自分用MATLABマニュアル

吉川周二 (Shuji Yoshikawa)
大分大学 理工学部 数理科学コース

December 19, 2017

Contents

Chapter 1

MATLABの基本

自分用のマニュアルで, 研究室内で共有利用するために公開はしますが, 信頼できる文書ではありません. 本文書を参考にしたことで生じる不利益などについて一切責任は負いません. 随時加筆・修正していくつもりですが, 特に更新情報などは表示しません.

(1) 短い計算ならコマンドウィンドウに直接書き込んでもよい.

(2) 長いプログラムは、「新規スクリプト」から「エディター」を開いてそちらに書き込む。実行は「F5」、拡張子は「.m」(Scilabでは実行は「Ctrl + L」, 拡張子は「.sce」)。関数ファイルも「.m」ファイル.

(3) 基本はScilabとほとんど同じ. 細かい違いはある. 変数名がScilabのように`%pi`でなく, `pi`だったりするので, 上書きに注意が必要. また, MATLABにある`eye(m)`はScilabにはないことによるエラーがよくある. あとファイルの入出力のコマンド名が`mopen`から`fopen`になるくらいの違い.

(4) 非線形方程式を解くのに使える`fsolve`は, Scilabではデフォルトで使えるが, MATLABではoptimization toolboxの関数なので使えない.

1.1 変数

変数は大文字・小文字を区別する. 使ってよい文字は, アルファベット・数字・「_（アンダースコア)」のみで, イニシャルはアルファベットでなければならない. Scilabでは組み込みの定数が「`%pi`」のなどだったが, MATLABでは「`pi`」のように小文字で始まるので, 自分で定義する変数は大文字にしておく方が無難.

1.2 マルチ・ステートメント

結果が表示されるマルチステートメント	, と「改行」
結果が表示されないマルチステートメント	;

1.3 定数

MATLAB の組み込み定数は, Scilab の「`%pi`」のように「`%`」をつけないため, 上書きに注意が必要. また自然対数の底 e は MATLAB には用意されていない. `exp(1)` のように使用すればよい. 真偽の t, f も MATLAB にはない.

円周率 π	`pi`
虚数単位 i	`i` または `j`
無限大 ∞	`inf` または `Inf`
値無し (Not a Number)	`nan`

1.4 スカラー間の基本演算

四則演算（+, −, ×, ÷）	`+, -, *, /`
べき乗 a^b	`a^b`

1.5 比較演算と論理演算

Scilab と同じ.

等しいならば真 (==)	`==`
異なるならば真 (=)	`~=`
不等号	`<=, >=, <, >`
「かつ」, 「または」(∩, ∪)	`&, |`
否定演算 (not P)	`~P`

注意 1.5.1. *Scilab* では真偽は変数`%t, %f` などで表されたが, *MATLAB* では真偽は 1, 0 で表す.

1.6 数列

a から始まる dx 刻みで b 以下で終わる数列	`a:dx:b`
真ん中の dx がないと $dx = 1$ として取り扱われる	`a:b`

例 1.6.1.

- `1:0.1:2` → *1, 1.1, 1.2, ..., 2.0*
- `1:2:20` → *1, 3, 5, ..., 19 (← 20* 以下だから*)*
- *(dx* は負でもよい*)* `5:-1:3` → *5, 4, 3*
- `1:3` → *1, 2, 3*

注意 1.6.2. このように生成された数列はベクトル（行列）の扱いになる．すなわち，
`a = 1:0.1:1.3` と `a = [1  1.1  1.2  1.3]` は同値である．また，
`--> a = 1:0.1:1.3; a(3)` と入力すると`--> ans = 1.2` が返される．

1.7 ベクトルと行列の基礎知識

この話は 2 章で詳しく述べる．MATLAB でも Scilab でも基本的にベクトルも行列も同じもの．ベクトルは一次元配列と考えればよい．ベクトル (数列)a の第 n 成分 (第 n 項) は `a(n)` で呼び出せる．

1.8 大きい数と小さい数

`6e+08` は 6×10^8 を意味する．Scilab ではこの `6D+08` の表記だった．

1.9 コメント

コメントは `%` で．Scilab では `//`だった．

1.10 数学関連の組み込み関数

Scilab と同じ．

$\log x$, $\log_{10} x$, $\log_2 x$	`log(x), log10(x), log2(x)`
$\sin x$, $\cos x$, $\tan x$	`sin(x), cos(x), tan(x)`
$\arcsin x$, $\arccos x$, $\arctan x$	`asin(x), acos(x), atan(x)`
$\sinh x$, $\cosh x$, $\tanh x$	`sinh(x), cosh(x), tanh(x)`
$\sinh^{-1} x$, $\cosh^{-1} x$, $\tanh^{-1} x$	`asinh(x), acosh(x), atanh(x)`
$\sqrt{x}$, $\lvert x\rvert$, sign$x$ など	`sqrt(x), abs(x), sign(x)`
小数点以下切り上げ (切り下げ) 四捨五入	`ceil(x), (floor(x))` `round(x)`
複素数の実部, 虚部, 複素共役	`real(x), imag(x), '`

1.11 算術演算子の優先順位

Chapter 2

行列操作

行列操作について述べる.

2.1 ベクトルと行列をつくる

行ベクトル $x = (1, 2, 3)$	`x = [1 2 3]` または `x = [1,2,3]`
列ベクトル $x = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ 列の場合は,「;」で区切ると改行. または転置演算「'」を用いる.	`[1;2;3]` `x = [1 2 3]'`
上記の規則を組み合わせると行列が定義できる. $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	`A = [1 2 3; 4 5 6; 7 8 9]` `A = [1,2,3;4,5,6;7,8,9]` でも勿論 OK.

2.2 行列を合成する

行列同士を組み合わせることも可能. ただし次数やサイズが合わないとエラーを吐く.

例 **2.2.1.**
```
1: X=[1 2 3]; Y=[4 5 6]; A=[X;Y]
2: A=
   1. 2. 3.
   4. 5. 6.
```

2.3 行列を生成するコマンド

n 行 m 列のゼロ行列 (n 次ゼロ行列)	`zeros(n,m), (zeros(n))`
n 行 m 列の 1 行列 (単位行列ではない)	`ones(n,m)`
n 次の単位行列, (n,n)-成分が 1 の n 行 m 列の行列	`eye(n), eye(n,m)`
数列 X を対角成分にもつ対角行列 オプションで対角線から上下に m ずらすことも可.	`diag(X)` `diag(X,m)`
一様乱数を成分とする行列 正規乱数を成分とする行列	`rand(n,m), rand(n)` `randn(n,m), rand(n)`

注意 2.3.1. *Scilab* には `eye(n)` がない *(*使えない. スカラー 1 を返す*)* ので注意. 同様に, `zeros(n)` や `ones(n)` や `rand(n)` などもつかえない. *Scilab* では `rand(n,m, 'uniform')` と `rand(n,m, 'normal')` だった.

例 2.3.2.

```
1: A=zeros(2,3)
  A=
   0. 0. 0.
   0. 0. 0.
2: B=ones(3,2)
  B=
   1. 1.
   1. 1.
   1. 1.
3: C=eye(2,4)
  C=
   1. 0. 0. 0.
   0. 1. 0. 0.
4: X=1:3; D=diag(X)
  D=
   1. 0. 0.
   0. 2. 0.
   0. 0. 3.
5: E=diag(X,-1), F=diag(X,+1)
  E=
   0. 0. 0. 0.
   1. 0. 0. 0.
   0. 2. 0. 0.
   0. 0. 3. 0.
  F=
```

```
   0. 1. 0. 0.
   0. 0. 2. 0.
   0. 0. 0. 3.
   0. 0. 0. 0.
```

2.4 行列の演算

行列の足し算・引き算	+, -
行列のかけ算とスカラー倍	*
行列の成分ごとのかけ算	.*

例 2.4.1.

```
1: A=[1 2; 3 4]; B=[0 2; 3 0];  C=A+B, D=A*B, E=A.*B
2: C=
   1. 4.
   6. 4.
3: D=
   6. 2.
   12. 6.
4: E=
   0. 4.
   9. 0.
```

行列の割り算	\@（バックスラッシュ，yen）または\verb/@ (スラッシュ)
行列の成分ごとの割り算	./

注意 2.4.2. 行列の割り算は逆行列を作用させることに相当する演算であり, そのため \ と/の 2 パターンある. すなわち, $AB = C$ に対して $B = A^{-1}C$ を求める演算は B=A\C で与えられ, $AB = C$ に対して $A = CB^{-1}$ を求める演算は A=C/B で与えられる.

その他, 行列のべき乗 (成分毎のべき乗)「^, .^」, 行列の成分毎の論理演算・関係演算「| , & , ~ , == , >= など」も同様に定義できる. また応用として, ベクトルの内積は行列の転置と積を組み合わせて定義できる. すなわちベクトル $A = (1, 2)$ とベクトル $B = (3, 4)$ の内積 $A \cdot B$ を求めたければ, 行ベクトル A と B に対して, A*B'$を計算すればよい.

```
1: A=[1 2]; B=[0 2];  A*B', dot(A,B)
2: ans =
   4
3: ans =
   4
```

注意 2.4.3. 上記のように, *MATLAB* には *(Scilab* にはない*)*, `dot(A,B)` という内積を計算する関数が準備されている. この `dot` は A と B が行ベクトル同士でも列ベクトル同士でもその混合したものであっても問題なく計算できる.

2.5 行列の全成分に施す関数

普通に関数に行列を作用させると各成分にその関数が施される.

例 **2.5.1.**

```
1: X=[1 2; 4 9]; sqrt(X)
2: ans=
   1.   1.4142135   2.   3.
```

この計算は C や Java だと for ループなどを用いることになり, MATLAB でも for ループで計算はできる. 必ず覚えておきたいのは, for ループで回すより, この演算の方が速い (プログラムが短いという意味でなく計算にかかる時間が実際に短縮される) ということである.

2.6 行列の関数

ベクトル・行列の成分の個数	`length(A)`
行列の大きさ (行の数と列の数) m をつけないと行と列の二つを返し $m = 1$ なら行 (縦) の数, $m = 2$ なら列 (横) の数	`size(A, m)`
行列の成分がゼロかどうか m をつけないと全ての成分がゼロであれば`F` を返す, $m = 1$ なら行 (縦) に対して, $m = 2$ なら列 (横) に対して検査.	`and(A, m)`
行列の成分がゼロかどうか m をつけないと成分に一つでもゼロであれば`F` を返す, $m = 1$ なら行 (縦) に対して, $m = 2$ なら列 (横) に対して検査.	`or(A, m)`
非ゼロ成分のインデックスを求める	`find(A)`

例 **2.6.1.** 大きさ

```
1: A=[0 0 3; 0 0 0]; length(A), size(A), size(A,1), size(A,2)
2: ans=
   6
3: ans=
   2.   3.
4: ans=
```

```
   2.
5: ans=
   3.
```

例 **2.6.2.** 論理積 *(and)*

```
1: A=[1 2 0; 3 2 1]; a1=and(A), a2=and(A,1), a3=and(A,2)
2: a1=
   F
3: a2=
   T  T  F
4: a3=
   F
   T
```

例 **2.6.3.** 論理和 *(or)*

```
1: A=[0 0 3; 0 0 0]; a1=or(A), a2=or(A,1), a3=or(A,2)
2: a1=
   T
3: a2=
   F  F  T
4: a3=
   T
   F
```

例 **2.6.4.** インデックス

```
1: A=[1 0 0 3; 0 2 0 0; 0 0 3 0]; find(A)
2: ans=
   1.  5. 9.  10.
```

2.7 線形代数にあらわれる行列の関数

逆行列	`inv(A)`
一般化逆行列	`pinv(A)`
行列式	`det(A)`
トレース	`trace(A)`
直交化	`orth(A)`
階数 (rank)	`rank(A)`
固有値と固有ベクトル	`spec(A)`
行列の平方根	`sqrtm(A)`
行列の指数関数	`expm(A)`
行列の対数関数	`logm(A)`
Cholesky 分解	`chol(A)`
LU 分解	`lu(A,m)`
QR 分解	`qr(A,m)`

詳しい使用法やオプションについてはヘルプを参照のこと.

2.8 抜き出す操作

A の (m,n)-成分	`A(m,n)`
A の m-行 (コロンを用いる)	`A(m,:)`
A の n-列 (コロンを用いる)	`A(:,n)`
上の 3 つの方法には応用がある. m などに行列を 入れてもよい. (例) n-列から l 行分抜き出す. 別の例は以下に...	`A(:, n:n+1)`
行列の対角成分を取り出す	`diag`

例 2.8.1.

```
1: A=[1 0 0 4; 0 2 0 0; 0 0 3 0]; A(1,4), diag(A), B=A(1,:), C=A(:,2)
2: ans=
   4.
3: ans=
   1.
   2.
   3.
4: B=
   1.  0.  0.  4.
5: C=
   0.
```

```
   2.
   0.
```

例 **2.8.2.**

```
1: A=[1 0 0 4; 0 2 0 0; 0 0 3 0]; Y=[1 3]; D=A(:, 2:4), E=A(Y,:)
2: D=
   0.  0.  4.
   2.  0.  0.
   0.  3.  0.
3: E=
   1.  0.  0.  4.
   0.  0.  3.  0.
```

2.9 加える操作

A にあらたな (m,n)-成分 x を付け加える. 追加された成分の行や列で指定されなかった部分には 0 が入る.	`A(m,n)`
行 Y を追加するには行列 A と一緒に行方向に並べればよい. 列 X を追加するには行列 A と一緒に列方向に並べればよい.	`[A;Y]` `[A X]`

例 **2.9.1.**

```
1: A=[1 2 3; 4 5 6]; A(3,4)=7
2: A=
   1.  2.  3.  0.
   4.  5.  6.  0.
   0.  0.  0.  7.
```

例 **2.9.2.**

```
1: A=[1 2 3; 4 5 6]; B=[7 7 7]; C=[8;8], D=[A;B], E=[A C]
2: D=
   1.  2.  4.
   4.  5.  6.
   7.  7.  7.
3: E=
   1.  2.  3.  8.
   4.  5.  6.  8.
```

2.10 行列の加工

行列 A の各列を縦につないで縦ベクトル X を作る. 組み込み関数matrix も使用できる.	A(:) X = matrix(A,length(A),1)
逆にベクトル X から $m \times n$ 行列 A を作るときにも matrix を用いればよい.	A = matrix(X,m,n)

例 **2.10.1.**

```
1: A=[1 2 ; 4 5]; X1=A(:), X2=matrix(A,4,1)
2: X1=
   1.
   4.
   2.
   5.
3: X2=
   1.
   4.
   2.
   5.
```

例 **2.10.2.**

```
1: X=1:12; A=matrix(X,2,6)
2: A=
   1.  3.  5.  7.  9.  11.
   2.  4.  6.  8.  10. 12.
```

2.11 行や列を取り除く

行や列を取り除きたければ空の行か列をセットすればよい.	[] （以下の例参照）

例 **2.11.1.**

```
1: A=[1 2 3; 4 5 6]; A(:,2)=[]
2: A=
   1.  3.
   4.  6.
```

例 **2.11.2.**

```
1: A=[1 2 3; 4 5 6]; A(1,:)=[]
2: A=
   4.  5.  6.
```

例 **2.11.3.**

```
1: A=[1 2 3; 4 5 6], A(:,[1 3])=[]
2: A=
   2.
   5.
```

2.12 三角行列

行列 A を下三角行列に.	`tril(A)`
行列 A を上三角行列に.	`triu(A)`

例 **2.12.1.**

```
1: A=[1 2 3; 4 5 6; 7 8 9]; B=tril(A), C=tril(A)
2: B=
   1.  0.  0.
   4.  5.  0.
   7.  8.  9.
3: C=
   1.  2.  3.
   0.  5.  6.
   0.  0.  9.
```

2.13 その他

文字列を扱ったり加工したりする関数や, グラフィック表示で便利な `meshgrid` などについて記載されていたが, 自分には当面必要なさそう.

Chapter 3

関数

大きなプログラムを作る場合は, 関数を別ファイルに作るのが便利である. また, メインプログラムに関数を書くこともできる.

3.1 別ファイルで関数を定義

基本形は

```
1: function 出力引数=関数名 (入力引数)
2: H1 行
3: ヘルプテキスト
4: 本体
```

の形で, ファイルは拡張子が $.m$ ファイルで保存して, メインファイル (スクリプトファイルと呼ばれる) と同じフォルダ (ディレクトリ) に置く. 「H1 行」は MATLAB のヘルプで引用される行で, 組み込み関数 `lookfor` を用いてキーワード検索するとこの行が表示される. 「ヘルプテキスト」は追加のヘルプで組み込み関数 `help` でここまでが表示される. H1 行とヘルプテキストは書かなくても勿論大丈夫だが, 長いプログラムになると書いておいたほうが無難. 出力変数や入力変数がない場合もある.

Scilab のように `endfunction` で終わる必要はない. 本文のスクリプトファイルの中に関数を書くときについては, "@" を用いて関数ハンドルを作成する必要がある. これについては次節で.

例 3.1.1. 以下は平均と総和を 2 次元ベクトルとして返す関数である.

FunctEx1.m

```
1: function Out = AverageSum(X)
2:   % AverageSum: Average and Sum of a vector X
3:   % Out = [Arv  S]; Avr and S are respectively
4:   % the average and the sum of a given vector X.
```

```
5:    N = length(X);
6:    S = sum(X);
7:    Avr = S ./N;
8:    Out = [Avr S];
```

実際に使うときには,

```
1: N=10;
2: X=1:N;
3: Y=AverageSum(X)
--> Y =
    55.  5.5
```

のようにつかえばよい.

- 変数は, 英文字で始まり, 英数字とアンダーバーからなる文字列を用いる. 文字数には制限はない. Scilab にはすでに多くの組み込み関数が用意されておりそれらは小文字で書かれているので, 大文字から始める名前にすると区別しやすい.
- 二つ以上の引数をとるときは, ,(カンマ) で区切る.
- 一つの関数ファイルに複数の関数を書いてもよい.
- ファイル名と関数名が一緒である必要もない.

出力引数がないとき,

```
1: function 関数名 (入力引数)
2:    本体
```

だけでよい. 入力引数がないときは,

```
1: function 出力引数=関数名 ()
2:    本体
```

のようにかく. 入力引数も出力引数も両方ともないときは

```
1: function 関数名
2:    本体
```

となる. いずれも H1 行とヘルプテキストは省略した.

関数の中に出てくる変数と, メインプログラムの中の変数が同じ文字の時でもそれらは「別物」として区別される. 同じものとして判断させたいときは, `global` を用いる.

```
FunctEx1.sci
1: function AverageSum2 (←出力引数も入力引数もない関数)
2: global X AVR S
2:   N = length(X);
3:   S = sum(X);
4:   AVR = S ./N;

Main.sce
1. global X ARV S
2. N=10; X=1:N;
3. AverageSum2;
4. AVR, S
--> AVR = 5.5
S= 55
```

3.1.1 入力引数の数を可変に

たとえば, Sum2 を二つの引数 Sum2(X,Y) のときは X+Y を, 1 つの引数 Sum2(X) のときは成分の総和 sum(X) としたいときがあるとする. このときは, 入力引数の数を与えてくれる関数 argn をもちいればよい.

argn(1)	出力引数の個数
argn(2)	入力引数の個数
argn()	出力引数と入力引数を成分とするベクトル

例.

```
FunctEx1.sci
1: function S = Sum2(X,Y)
2:   if argn(2)==1
3:     S= sum(X);
4:   else
5:     S= X+Y;
6:   end
7: endfunction
```

3.2 メインプログラムで関数を定義する

これまでは関数をメインプログラムと別のファイルに書いてきたが, 簡単なものであればメインプログラムに直接書けばよい. 本文のスクリプトファイルの中に関数を書くときについては, "@"を用いて関数ハンドルを作成する必要がある.

例 **3.2.1.** メインプログラムの中で関数を定義する例

```
1: N=10;
2: X=1:N;
3: Y=AverageSum(X)
4: AverageSum=@(X) [sum(X)/N  sum(X)];
--> Y =
    55.  5.5
```

注意 3.2.2. *Scilab* では deff を用いていた．基本形は deff(’ 出力引数=関数名 (入力引数)’,[’ 本体 (複数行でもよい)’]) 注意するのは, 本体の実行文もすべて「’」でかこみ, 行列の各行に格納しておくこと.

Chapter 4

構文

主に [1] に従った.

4.1 ループ, 繰り返し処理

4.1.1 For 文

基本形は,

```
1: for 繰り返し部分
2: 実行文たち
3: end
```

一文で

```
1: for 繰り返し部分, 実行文たち end
```

と書くこともできる. 例としては

```
1: x=0;
2: for I=1:10
3:    x=x+I;
4: end
5: x
6: --> x=
        45
```

や

```
1: x=0;
2: for I=1:10, x=x+I; end
3: x
4: --> x=
        45
```

4.1.2 While 文

基本形は,

```
1: while 終了条件
2: 実行文たち
3: end
```

一文で

```
1: while 終了条件; 実行文たち end
```

と書くこともできる. 例としては

```
1: x=0; I=0;
2: while I<10
3:    I=I+1; x=x+I;
4: end
5: x
6: --> x=
        45
```

や

```
1: x=0; I=0;
2: while I<10; I=I+1; x=x+I; end
3: x
4: --> x=
        45
```

4.1.3 ループから抜け出る (break)・続ける (continue)

(1) `break`　ある条件が満たされたら中断したいときがある. 次のプログラムは 20 秒間整数の乱数を発生し続けるが, もし整数 2013 が得られれば探索を打ち切るというものである.

```
1: tic()
2: while toc()<20
3:   X=fix(rand()*10000);
4:   if X == 2013
5:       X, break
6:   end
7: end
```

rand() は 0 以上 1 未満の一様乱数. fix は小数部分を切り捨てて整数にする関数. 20 秒間の間ループが続くが, もし $X = 2013$ となると break によってループを抜け出してプログラムが終了する.

ループが何重にもなっているときは, break によって抜け出る場所はそれが置かれているループのすぐ外側になる.

(2) continue　ある条件が満たされても繰り返しをつづけたいときがある. 次の例は, 20 秒間乱数を発生し続け, もし整数 2013 が得られれば表示はするが, 探索を続けるプログラムである.

```
1: tic()
2: while toc()<20
3:   X=fix(rand()*10000);
4:   if X ~= 2013
5:       continue
6:   end
7:   X
8: end
```

3 行目で発生した乱数が 2013 でなければ, continue によって, 最後の 8 行目の end まで飛びループを続ける. もし 2013 なら, 普通に 7 行目が実行されるので, X が表示される.

ループが何重にもなっているときには, continue によって飛ぶ場所は, それが置かれているループの最後の部分になる.

4.2　分岐

4.2.1　if ... end

基本形は,

```
1: if 判定条件
2: 実行文たち
3: end
```

判定条件が T なら実行され, F なら何も実行されない. 何重にネストすることもできる. 一文で

```
1: if 判定条件，実行文たち end
```

と書くこともできる. 例としては

```
1: x=2;
2: if x>0
```

```
3:    disp('x is positive');
4: end
5: --> x is positive
```

4.2.2 if ... else ... end

2 つのことがらを条件によって分けて実行するにはこれをもちいる. 基本形は,

```
1: if 判定条件
2: 実行文たち 1
3: else
4: 実行文たち 2
3: end
```

判定条件が T なら実行文たち 1 が実行され, F なら実行文たち 2 が実行される. 何重にネストすることもできる.

4.2.3 if ... elseif ... else ... end

3 つのことがらを条件によって分けて実行するにはこれをもちいる. 基本形は,

```
1: if 判定条件 1
2: 実行文たち 1
3: elseif 判定条件 2
4: 実行文たち 2
5: else
6: 実行文たち 3
3: end
```

判定条件 1 が T なら実行文たち 1 が実行され, F なら判定条件 2 が判定される. この判定条件 2 が T なら実行文たち 2 が実行され, F なら実行文たち 3 が実行される. 何重にネストすることもできる. `elseif` のあとにさらに `elseif` を続けることによって, 4 つ以上の幾通りもの場合分けが可能だが, このような場合は `select` (Scilab のみ) を用いた方がよい. MATLAB には `select` はない.

Chapter 5

テクニック

5.1 高速化のコツ

主に [1](3 章) に従った.

5.1.1 実行速度を測る

まずは, 実行速度を測るコマンドは `tic , toc` である. `tic()` でタイマーをスタートさせ, `toc()` でタイマーを止める.

例としては

```
1: x=0;
2: tic()
3: for I=1:10
4:    x=x+I;
5: end
6: T1=toc()
7: x
8: --> T1=0.015.
9: --> x=45.
```

5.1.2 ループより

たとえば和の計算を行う場合は, `while , for` を使ってループを回すより, `sum` を用いた方が格段にはやい. また, $f(n\Delta x)$ を各 n に対して計算するときも, ループで計算していくより, `f(0:\Delta x: N)` で計算した方がはやい.

[1][P91]

5.1.3 領域の確保

N 個の配列 $a[n] = sin(n\Delta x)$ を計算するときに, 突然計算させるより, ループの前にあらかじめ `a = zeros(1,N)` とでも設定しておくとメモリが確保され速い. もしこれをしないと, 毎回の計算ごとにメモリを確保しながら計算するため時間のロスが多くなってしまう. Scilab や MATLAB は C や JAVA のように型の宣言をしないでよいかわりにこういったことに気を配るとよい.

Chapter 6

ファイルの読み書き

MATLAB にはバイナリファイルへの読み書きとテキストファイルへの読み書きに大きく分けられるが, 通常後者しか使わないので, テキストファイルへの読み書きのみについて説明する. 主に [1] に従った.

6.1 テキストファイルへ書き込む

ファイルにデータを書き込むのは, Java などと同じで, まずファイルをオープンして, 次にデータを書き込んで, 最後にファイルをクローズする.

```
1: X=1:5, Y=X .^3
2: Fid = fopen('ex6201.txt','wt')
3: fprintf(Fid, '%2d%5d\n', X', Y');
4: fclose(Fid)
--> X=
1. 2. 3. 4. 5.
    Y=
1. 8. 27. 64. 125.
```

2 行目でファイルをオープンして, 3 行目でファイルに書き込むのだが, `%2d%5d\n` は, 最初の数値が整数で二桁の幅に, 次の数値が整数で 5 桁の幅に書きここで改行する. ことを意味している. 生成されたテキストファイル `ex6201.txt` を開いてみると以下のようになっている.

```
1 1
2 8
3 27
4 64
5 125
```

fopen のオプション'wt' はほかに

読み込み	'rt'
読み込みと書き込み	'rt+'
書き込み (もし同名のファイルがあればそれを削除)	'wt'
読み込みと書き込み (もし同名のファイルがあればそれを削除)	'wt+'
追加書き込み	'at'
追加書き込みと読み込み	'at+'

Scilab では, fopen, fprintf, fclose, wt, rt, rt+, ... がそれぞれ, mopen, mfprintf, mclose, w, r, r+, ... であるだけの違いである. この語尾の t は text ファイルからの読み込みであることを強調するためにつけられている. バイナリファイルの読み書きの場合はこの t はつかない.

\%2d%5d\n の部分だが, フラグとしては,

マイナスの符号を左端に寄せる.	-, 例: \%-5.2f
つねにプラスの符号を表示する.	+, 例: \%+5.2f
数字の前にスペースの代わりに 0 を詰める	例: \%05.2f

以上は全て Scilab と共通である.

変換文字としては

文字	c
10 進整数	d
指数表示 3.1415e+00	e
指数表示 3.1415E+00	E
固定小数点表示 3.1415	f
指数表示または固定小数点表示 のいずれか簡潔な方	g(指数が小文字) G(指数が大文字)
8 進数表示 (符号なし)	o
文字列	s
符号なしの 10 進数整数	u
16 進数表示	x, X

以上は全て Scilab と共通である.

制御文字 (エスケープ文字) としては

改行	\n
タブ	\t
バックスペース	\b
キャリッジリターン	\r
フォームフィード	\f
バックスラッシュ	\ \
クォーテーションマーク	\"
パーセンテージ	%%

最後のパーセンテージ以外は全て Scilab と共通である.

6.2 ファイルから読み込む

```
1: Fid = fopen('ex6201.txt','r')
2: N=fscanf(Fid, '%d%d', [2,inf]);
3: fclose(Fid);
4: N'
--> ans =
1. 1.
2. 8.
3. 27.
4. 64.
5. 125.
```

1 行目でファイルをオープンし, 2 行目でデータを読み込み, 3 行目にファイルをクローズしている. 2 行目の `[2,inf]` は読み込んだデータを 2 行の形にして変数 `N` に格納し, `inf` はデータのある限りファイルの終わりまで読み込むことを意味している. `'%d%d'` は 2 個の整数を読み込むことを意味している. 注意しなくてはいけないのは, ファイルから読み込んだデータは行列の列方向に格納されること. (Scilab では行方向だった. そのため, 転置の操作は必要なかった)

まとめると組み込み関数 `fscanf` は

```
fscanf(Fid, format, Size)
```

の形をしており, 引数 `Size` は次の形で指定できる:

N 個のデータを列ベクトルに格納	`N`
ファイルの終わりまでのデータを列ベクトルに格納	`inf`
M 行 N 列の行列に列方向にデータを格納 (N は`inf` でもよい)	`[M,N]`

Scilab では 2 行目は `mfscanf(%inf, Fid, '%d%d');`, 4 行目は `N, [X Y]` と書く.

Chapter 7

グラフ

ここでは MATLAB のもつグラフ描画機能について説明する. [1] に従う.

まず $y = \sin x$ などの既知の関数を描画するだけなら `ezplot, ezsurf` などを用いればよい.

7.1 二次元グラフ

(x, y)-平面上の点の列 (x_n, y_n) $(n = 1, 2, \ldots, N)$ を線で繋いで得られる 1 次元の線図形 (曲線, 折れ線) は `plot(X,Y)` で描ける, ただし, `X, Y` は同じ次数のベクトル (数列) とする.

```
1: A = [1 2 3 4 4 3 2 1 1]; B = [1 0 0 1 2 3 3 2 1]; plot(A,B)
```

とすると, 下図が得られる. 数は得られた図を eps ファイルの形式で保存して TeX でコンパイルしている.

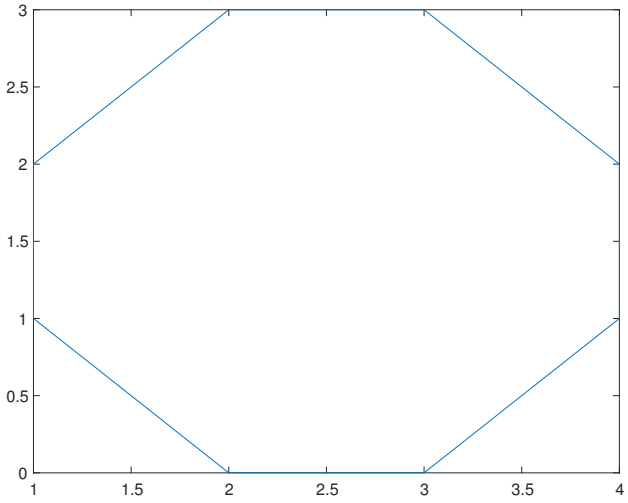

Figure 7.1: 図の例

7.2　三次元グラフ

7.2.1　曲線

(x, y, z)-空間内の点の列 (x_n, y_n, z_n) $(n = 1, 2, \ldots, N)$ を線で繋いで得られる 1 次元の線図形 (曲線, 折れ線) は `plot3(X,Y,Z)` で描ける, ただし, `X, Y` は同じ次数のベクトル (数列) とする.

```
1: A = 1:5; B = [3 2 1 2 3]; C = [1 2 1 2 1]; plot3(A,B,C)
```

とすると, 下図が得られる. 数は得られた図を eps ファイルの形式で保存して TeX でコンパイルしている.

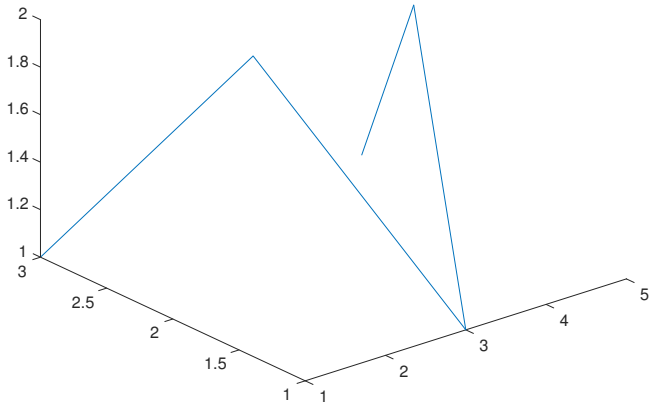

Figure 7.2: 図の例

7.2.2　曲面

x 軸方向に n 個の分割点 $x_1, x_2, \ldots, x_n$, y 軸方向に m 個の分割点 $y_1, y_2, \ldots, y_m$ を与えたときに, あらわれる $n \times m$ 個の格子点 (x_k, y_ℓ) $(k = 1, 2, \ldots, n,\ \ell = 1, 2, \ldots, m)$ を考える. この格子点に対して $z_{k\ell} = z(x_k, y_\ell)$ を対応させる曲面を描画したい.

そのために, まず `meshgrid` という関数を用いて, ベクトル $(x_1, x_2, \ldots, x_n)$, $(y_1, y_2, \ldots, y_m)$ から

$$\left.\begin{pmatrix} x_1 & x_2 & \ldots & x_n \\ x_1 & x_2 & \ldots & x_n \\ \vdots & \vdots & \vdots & \vdots \\ x_1 & x_2 & \ldots & x_n \end{pmatrix}\right\} m \qquad \underbrace{\begin{pmatrix} y_1 & y_1 & \ldots & y_1 \\ y_2 & y_2 & \ldots & y_2 \\ \vdots & \vdots & \ldots & \vdots \\ y_m & y_m & \ldots & y_m \end{pmatrix}}_{n}$$

という m 行 n 列の 2 つの行列を作成する.

```
1: XX = [x_1, x_2, ... , x_n]; YY = [y_1, y_2, ..., y_m];
2: [X Y] = meshgrid(XX,YY);
```

とすればよい．これで [X Y] の $m \times n$ 個の成分が (x_k, y_ℓ) の $m \times n$ 個の成分になる．この $z_{k\ell}$ が描く曲面はここで生成された行列 X，Y とこれらに対応する Z=Z(X,Y) に対して mesh(X, Y, Z) とすればよい．

```
1: XX = -2:0.2:2; YY = -1.4:0.2:1.4;
2: [X Y] = meshgrid(XX,YY);
3: Z=exp(-(X.^2+Y.^2));
4: mesh(X,Y,Z);
```

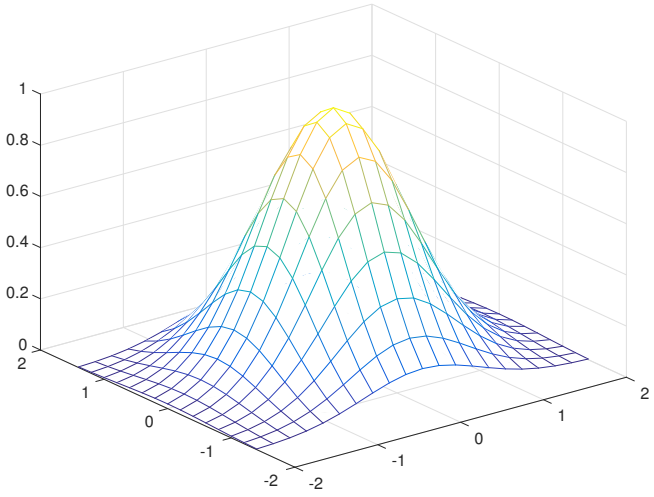

Figure 7.3: 図の例

7.3 様々なオプション

様々なオプションについては, [1] 以外に [5] などを用いるとよい．

Chapter 8

その他

8.1 GUI

GUI をプログラムすることもできる ([1]).

8.2 数式処理

簡単な数式処理であれば, 計算できる. ([3, 12 章]).

References

[1] 上坂吉則, MATLAB プログラミング入門, 牧野書店, 2000.

[2] 櫻井鉄也, MATLAB/Scilab で理解する数値計算, 東京大学出版会, 2003.

[3] 大石進一, MATLAB による数値計算, 培風館, 2001.

[4] 北村達也, はじめての MATLAB, 近代科学社, 2016.

[5] 上坂吉則, MATLAB 可視化プログラミング, 牧野書店, 2013.