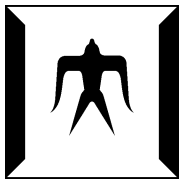
MATX (ver. 4.384)

リファレンス マニュアル



東京工業大学　古賀 雅伸

I(), 151
ONE(), 152
Z(), 151
微分方程式, 190
左特異ベクトル, 186
微分
多項式行列の—, 159
多項式の—, 138
有理多項式行列の—, 165
有理多項式の—, 144
評価
多項式行列の—, 156
多項式の—, 137
文字列の—, 130
有理多項式行列の—, 162
有理多項式の—, 143
標準偏差, 184, 185, 201, 202

ファイルディスクリプタ, 66, 119, 133, 210
ファイルのアクセス権, 68
ファディーブ, 216
フーリエ変換, 205
複数値関数, 106
複素オペレータ, 125
複素行列, 153, 182
複素数, 125
複素多項式, 136
複素値, 51
符合関数, 120
符合反転, 268
部分行列, 147, 167
プリプロセッサ, 53
プロセスディスクリプタ, 67
ブロック成分, 167
ブロック成分の代入, 171
ブロック成分の取り出し, 167
ブロック対角行列, 153
プロテクトモード, 276
プロテクトモード, 276
フロベニウスノルム, 182
プロンプト, 28
分子多項式, 142
分子配列, 166
分母多項式, 142
分母配列, 166

平均値, 184, 185, 201, 202
平方根行列, 186
ベクトル積, 186
ヘッセンベルグ分解, 198
ベル, 43
ヘルプ, 45
変数, 97
変数消去, 41
変数の存在, 41

ボード線図, 218
ポストスクリプト, 88

マシーン, 240
丸め関数, 120

4.2.2　一般データのファイル入出力 63
4.2.3　バイナリデータのファイル入出力 63
4.2.4　一般データの保存と読み込み 65
4.3　ファイル入出力 66
4.4　プロセスの終端の検出 67
4.5　プロセス間通信 67
4.6　ファイルのアクセス権を調べる 68
4.7　ディレクトリを変更する 69

第 5 章　行列エディタ (Matrix Editor)　71
5.1　実行列の編集 72
5.2　複素行列の編集 72
5.3　キーバインディング 74
5.4　行列エディタ (mated) 74

第 6 章　グラフィックス　77
6.1　gnuplot 77
6.1.1　基本的なプロット 78
6.1.2　複数の線 79
6.1.3　対数プロット 81
6.1.4　1 個のウィンドウに複数のグラフ 84
6.1.5　ウィンドウのクリアと消去 84
6.1.6　ラベル，タイトル，グリッド，テキスト，他のコマンド 86
6.1.7　(ポストスクリプト—FIG コード) ファイルに出力 . . 88
6.1.8　Windows95/NT で mgplot を使う 88
6.1.9　X ウィンドウを必要としないグラフィック表示 90
6.2　xplot 90
6.2.1　基本的なプロット 91
6.2.2　複数の線 92
6.2.3　対数プロット 93
6.2.4　ウィンドウ・マネ− ジャ− の設定 93

行列の—, 185
配列の—, 202
sum_row()
行列の—, 185
配列の—, 202
svd(), 198
switch, 115
system(), 42

tan()
実数の—, 120
配列の—, 205
複素数の—, 126
tanh()
実数の—, 120
配列の—, 205
複素数の—, 126
Tektronix, 90
TEX, 91
trace(), 182
多項式行列の—, 184
複素行列の—, 184
有理多項式行列の—, 184
trans()
行列の—, 187
実数の—, 120
整数の—, 119
多項式の—, 139
配列の—, 205
複素数の—, 126
有理多項式の—, 145
twm, 93

UNDEF, 208
UNIX, 27
unsinged char, 63
unsinged int, 63
unsinged short, 63

vec2diag()
行列の—, 187
配列の—, 203
VecChop(), 230
VecCont(), 230
vectorprod(), 186
void, 103, 105, 110
vt98, 90

warning(), 44
what, 40
which, 40
while, 114
who, 40
whos, 40
Windows95/NT, 88

X-Window System, 90
xplot, 18, 90
xplot_grid(), 91
xplot_semilogx(), 93
xplot_title(), 91
xplot_xlabel(), 91
xplot_ylabel(), 91

15.5.1 成分の取り出し . . . 167
15.5.2 ブロック成分の取り出し . . . 167
15.5.3 式行列の成分の取り出し . . . 168
15.5.4 0-1 配列を用いた成分の操作 . . . 168
15.5.5 成分の代入 . . . 170
15.5.6 ブロック成分の代入 . . . 171
15.5.7 行列成分の削除 . . . 172
15.5.8 行列の大きさの変更 . . . 172
15.5.9 指数を用いた行列成分の操作 . . . 173
15.5.10 行列の配列 (ピース行列の利用) . . . 175
15.6 行列の比較と配列の比較 . . . 177
15.7 行列とスカラの演算 . . . 181
15.8 行列関数 . . . 182
15.9 配列関数 . . . 200

第 16 章 リスト 207
16.1 リストの記述方法 . . . 207
16.2 リスト成分の参照と代入 . . . 208
16.3 リスト成分の付加 . . . 209
16.4 リストの合成と取り出し . . . 209
16.5 リスト関数 . . . 210

第 17 章 例題 211
17.1 連続系リカッティ方程式 . . . 212
17.2 離散系リカッティ方程式 . . . 214
17.3 線形システムの伝達関数行列 . . . 216
17.4 線形システムのボードとナイキスト周波数応答 . . . 218
17.5 連続系システムのシミュレーション . . . 220
17.5.1 状態フィードバック制御のシミュレーション (1) . . . 220
17.5.2 状態フィードバック制御のシミュレーション (2) . . . 223
17.5.3 動的フィードバック制御のシミュレーション . . . 225

実数の—, 120
整数の—, 119
配列の—, 201, 203
複素数の—, 126
minimum(), 182
minsing(), 182
mm, 27, 61, 65
mm ファイル, 19, 23, 28, 61, 65, 109
MS-DOS, 61
mx, 63
MX データフォーマット, 63, 240

NaN, 117
norm(), 182
Nu()
有理多項式配列の—, 166
有理多項式の—, 142

Ode(), 190
Ode45(), 193
Ode45Auto(), 194, 221, 225, 226
Ode45Hybrid(), 195
Ode45HybridAuto(), 196, 221, 296
OdeAuto(), 191
OdeHybrid(), 192
OdeHybridAuto(), 192
OdeStop(), 197
OdeXY(), 197
on_task(), 298
ONE(), 152
OS, 42
Outport(), 291
Outportb(), 291

p-ノルム, 182
PAGER, 45
pause, 46
pclose(), 67
PI, 117
PID, 117
PoArray, 147
PoArray(), 49
poles(), 145
PolyInit(), 36
poly_last, 37
Polynomial(), 48, 135
PoMatrix, 147
PoMatrix(), 49
popen(), 67
Postscript, 79
pow()
行列の—, 187
実数の—, 120
整数の—, 119
多項式の—, 139
配列の—, 205
複素数の—, 126
有理多項式の—, 145
print, 39, 62, 63
printf(), 39

表 目 次

1.1 MATX の動作環境 25

2.1 matx のオプション 31
2.2 matc のオプション 32
2.3 コマンドライン編集機能のキーバインディング 34
2.4 履歴の設定 34

3.1 メニューのキーバインディング 43

4.1 バイナリデータの型 64

5.1 mated のキーバインディング 75

6.1 グラフィック関数 1(gnuplot) 77
6.2 グラフィック関数 2(gnuplot) 80
6.3 グラフィック関数 3(gnuplot) 82
6.4 グラフィック関数 4(gnuplot) 83
6.5 グラフィック関数 5(gnuplot) 85
6.6 グラフィック関数 6(gnuplot) 85
6.7 グラフィック関数 7(gnuplot) 87
6.8 グラフィック関数 8(gnuplot) 89
6.9 グラフィック関数 9(gnuplot) 89
6.10 グラフィック関数 1(xplot) 91
6.11 グラフィック関数 2(xplot) 91

extern, 98

fact()
　実数の—, 120
　整数の—, 119
fclose(), 66, 119
feof(), 67
fft(), 205
FIG コード, 88, 91
find(), 168, 201
finite()
　行列の—, 182
　実数の—, 120
　配列の—, 200
　複素数の—, 126
fix()
　行列の—, 187
　実数の—, 120
　多項式の—, 139
　配列の—, 205
　複素数の—, 126
　有理多項式の—, 145
fliplr()
　行列の—, 187
　配列の—, 203
flipud()
　行列の—, 187
　配列の—, 203
float, 63
floor()
　行列の—, 187
　実数の—, 120
　多項式の—, 139
　配列の—, 205
　複素数の—, 126
　有理多項式の—, 145
fopen(), 66, 119, 133, 210
for, 114
fprintf(), 66, 67
　実数の—, 122
　整数の—, 119
　文字列の—, 133
fread(), 63
frobnorm(), 214
　行列の—, 182
　多項式行列の—, 184
　配列の—, 201
　有理多項式行列の—, 184
fscanf(), 66, 67, 210
　実数の—, 122
　整数の—, 119
　文字列の—, 133
Func, 105
fvwm, 93
fwrite(), 63

gcls(), 90
getch(), 119
getenv(), 42
gettimer(), 44
gnuplot, 18, 77, 90
gplot(), 77

第1章　はじめに

MATX は科学や工学に必要な数値及び数式計算をサポートする記述性に優れたプログラミング言語である [9, 10]。MATX は数値解析，行列計算，数式処理を使い易い形に統合化しているので，ノートに書いた数学的表現をほとんどそのままプログラムとして記述できる。このためユーザは他の計算機言語 (Fortran，Pascal，C，Basic) でプログラムを記述するのに比べてずっと短い時間で多くの計算問題を解くことができる。インタラクティビティと (Fortran や C 言語の最適化されたライブラリの) パフォーマンスを結合したいユーザは，MATX の柔軟性と便利さを非常に重要なものと考えるでしょう。

システム理論や信号処理の研究において，多項式や有理式そしてそれらを成分としてもつ多項式行列や有理多項式行列などの数式表現は，重要な役割を果たす。MATX を使えば，これらのデータの定義や操作を自然な表現で記述することができる。これにより，システム理論に必須である数値行列のさまざまな演算や，多項式，有理多項式，などの数式の処理を理論中の表記とほとんど同じ形式で記述でき，制御系の解析・設計だけでなく，システム理論の新しい設計アルゴリズムの開発や検証などの支援にも役立つ。

システム理論，プロセス同定，制御系設計などの研究には，対話型処理が有効な場合と一括型処理が威力を発揮する場合がある。対話型処理系は，解析や設計のためにコマンドライン上でいろいろなコマンドを組み合わせて使うためだけでなく，プログラムを作成する場合，開発時間を大幅に節約するのに役立つ。一方，一括型処理系はプログラムが完成した後で，計算時間を短縮するのに役立つ。特に，シミュレーションのように計算量の多い処理は，一括型処理によって飛躍的に計算速度を向上させることができる。MATX は，対話型処理を行なうインタプリタ (matx) とコンパイラ (matc) を提供する。

.&&, 180
.+, 181
.-, 181
./, 263
.<, 177
.<=, 177
.==, 177
.>, 177
.>=, 177
.^, 265
.||, 180
.~, 263
.fwmrc, 93
.matx_history, 33
.matxout, 58
.matxrc, 57
.twmrc, 93
/, 263
/*, 52
:, 152, 167
　複素数, 125
　複素有理多項式, 141
<, 269
　行列の比較, 177
　配列の比較, 179
<-, 63
<=, 269
　行列の比較, 177
　配列の比較, 179
==, 269
　行列の比較, 177
　配列の比較, 179
>, 269
　行列の比較, 177
　配列の比較, 179
>=, 269
　行列の比較, 177
　配列の比較, 179
[, 148
[], 148
||, 271
], 148
^, 265
_matxout, 58
_matxrc, 57
<<, 62
>>, 62
||
　配列の比較, 180
~, 263
0-1 配列, 168
0×0 行列, 148
1-ノルム, 182
10 進数, 117
16 進数, 117
1 で満たされた行列, 152
8 進数, 117

A/D 変換器, 281
abs(), 133
　行列の—, 186

ンタフェースを提供することである。データの保存に必要なメモリ領域を管理し，データの属性に応じて適当な低レベル・ライブラリの関数を呼び出す。

制御系の CAD が開発されるにつれ，制御系設計アルゴリズムの計算を数値的に安定に行なうことの重要性が認識され，多くの数値的に安定なアルゴリズムが提案された [11]。最適レギュレータやオブザーバなどの線形制御理論を中心とする制御系の解析・設計に必須である LU 分解，QR 分解，QZ 分解，固有値問題，一般固有値問題，等の行列演算を数値的に安定に行なうアルゴリズムをまとめたものに Linpack[2] や Eispack [4, 15] がある。 これらのパッケージは，Fortran で記述されており，簡単に入手できるので，多くの制御系 CAD の内部で利用されている [13, 17]。MATX は，Eispack に含まれるルーチンのうち固有値問題と一般固有値問題のルーチンを C 言語で書き直したものを使用している。

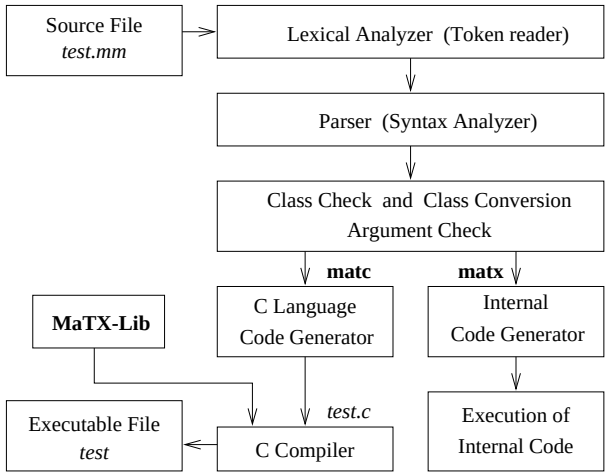


図 1.2: MATX の内部処理

図 1.2 は，MATX の内部処理を示している。MATX 言語で書かれたソースファイル (test.mm) を入力すると，入力文字列は，字句解析部によって MATX の認識する表現単位に分割される。分割された基本単位 (token) は，構文解析部に渡され，MATX の構文ルールにしたがって再構成される。ルー

E.2.2　matx の実行時のメッセージの説明

Call nested too deeply
関数の呼び出しが深すぎた。

Can't find the function
宣言された関数の定義が見つからなかった。

Can't open file
fprintf() によるファイルのオープンが出来ない。

Can't open help file
ヘルプファイルをオープンできない。

Can't open temporary file for help
ヘルプのための一時ファイルをオープンできない。

Confliction of the argument class
関数の引数の型が定義と異なる。

Data is broken or too few
<-で読み込んだデータが壊れている。

Duplicate switch label
１つの switch 文に同じラベルが２度使われた。

'extern' is used in argument definition
extern 宣言が関数の引数の宣言で使われた。

Function cann't be edited
edit コマンドで関数を編集しようとした。

Function cann't be read
read コマンドで関数を読み込もうとした。

表 1.1: MATX の動作環境

コンピュータ	OS
CRAY C916/12256	Cray UNICOS
JP4 (PowerPC604)	JCC_BSD+ 1.0
NEWS5000X	NEWS-OS 4.2.1R
HP 9000/755	HP-UX
SGI Origin2000	IRIX6.4
Mips RC6280	UMIPS4.52C
Sun4	SunOS4.1.x, Solaris 2.5.x
Sun3	SunOS4.1.x
PC/AT 互換機	Linux (1.2.x, 2.0.x)
PC/AT 互換機	FreeBSD 2.1.x, FreeBSD 2.2.x
PC/AT 互換機	Solaris 2.5.1 x86
PC/AT 互換機	BSD/OS 2.0
PC/AT 互換機	BSD on Windows
PC/AT 互換機	Windows 95/NT, (Visual C++)
PC/AT 互換機	Windows 95/NT, (Watcom C/C++)
PC/AT 互換機	Windows 95/NT, (DJGPP 2.01)
PC/AT 互換機	DOS/V, (DJGPP 2.01)
PC9801(NEC)	DOS, (DJGPP 2.01)
X68000(SHARP)	Human68k (gcc + libc)

label must be constant
　switch 文のラベルに変数が使用された。

label must be an integer
　switch 文のラベルに整数でないものが使用された。

Missing quote
　文字列の定義中に改行が行われた。

Not a complex matrix
　実行列の成分に複素数が代入された。

Not a number
　インクリメント演算やデクリメント演算に整数と実数以外のものが使われた。

Not a polynomial nor rational matrix
　多項式行列，有理関数行列以外の行列の成分に多項式や有理関数が代入された。

Not appropriate arguments for ”save”
　データを保存するファイルが指定されなかった。

Not appropriate arguments for ”print”
　データを保存するファイルが指定されなかった。

Not appropriate arguments for ”read”
　データを読み込むファイルが指定されなかった。

Not a variable
　インクリメント演算やデクリメント演算に定数が使われた。

Not suitable arguments class in exit()
　関数 exit() に整数以外のものが渡された。


```
% emacs sub3.mm
% matx

MaTX (1) load "main", "sub1", "sub2", "sub3"
```

5. 宣言だけされて定義されていない関数が呼び出されると，関数名と同じ mm ファイルが探索されて読み込まれるので，実行したい関数を宣言した後，“funcname();” と入力すれば，望みの関数を実行することができる。ただし，この機能を利用するには，関数名と mm ファイルのファイル名が一致していなければならない。

   [例] 実行したい関数 “main()” を宣言して，関数を直接呼び出す。

```
% emacs main.mm sub1.mm sub2.mm sub3.mm
% matx

MaTX (1) void main();
MaTX (2) main();
```

6. UNIX のシェルスクリプトや Perl のスクリプトと同じように、mm ファイルの先頭に “#! /usr/local/bin/matx” を追加して、ファイルに実行許可の属性を与えることによって、通常の UNIX のコマンドと同様に mm ファイルを実行させることができる。ただし、C プリプロセッサの機能を使うことができない。

   [例] 実行したい mm ファイルの先頭に “#! /usr/local/bin/matx” を追加して、実行許可の属性を与える。スクリプトの実行はコマンドラインから行なう。

```
"filename", in function(), for "variable(class)", message
"filename", in function(), line number, "sourcecode", message
```

matx によるプログラム解釈時のメッセージは，次の書式で表示される。

```
"filename", at or near line number, for "variable(class)", message
"filename", at or near line number, "sourcecode", message
```

上に示した一般的な名前および値は，エラーメッセージが出力されるときは，これらの名前は適切な実際の名前あるいは値に置き換えられる。

コンパイラ matc は，エラーを検出したソースファイルに対しては，C ファイルを生成しない。また，インタプリタ matx は，エラーがあるソースファイルに対しては，内部中間コードを生成しない。このようなエラーが発生した場合，コンパイラやインタプリタは，エラーからの回復を図り，可能であればソースファイルの処理を続け，エラーメッセージを表示する。エラーの数が多すぎる場合や非常に重大なエラーがある場合は，コンパイラやインタプリタや処理を中止する。

エラーメッセージの行番号については注意が必要である。コンパイラやインタプリタは検出した通りにメッセージを表示するだけである。MATX ではソースファイルの文の配置については強制の制限がないため，実際にエラーの原因は示された行番号より前にあることがある。

Mathematica や Maple の出力コードのように 1 個の式がとても長いとき，-Nc オプションで内部バッファのサイズを大きく，-Np オプションでプログラムコードのサイズを大きくする必要がある。内部バッファのデフォルトのサイズは 50000 バイト (MS-DOS 版は 10000 バイト)，プログラムコードのデフォルトのサイズは 50000 ステップ (MS-DOS 版は 10000 ステップ) である。[例] 4 つのソースファイルから実行可能型ファイル main を作成する。mm ファイルから一旦 C ファイルやオブジェクトファイルを生成し，コンパイル，リンクする。

```
% emacs main.mm sub1.mm sub2.mm sub3.mm
% matc -mm main.mm sub1.mm          // main.c, sub1.c 生成
% matc -c sub2.mm sub3.mm           // sub2.o sub3.o 生成
% matc -c main.c sub1.c             // main.o sub1.o 生成
% matc main.o sub1.o sub2.o sub3.o
% main
```

2.5　コマンドライン履歴・編集

MATX はコマンドラインの編集機能や履歴機能を提供する。インタプリタ matx を対話的に使う場合，この機能を用いると入力編集がとても簡単に行なえる。

コマンドライン編集機能のキーバインディングを以下の表に示す。記号 ^ は，Ctrl キーを押しながら続く文字を同時に押すことを表す。例えば，^C は，Ctrl キーを押したまま C キーを同時に押すことを意味する。

matx の終了時に，履歴は作業ディレクトリのファイル (.matx_history) に保存され，起動時に自動的に読み込まれる。すなわち，前回のセッションの履歴を利用できる。-nohist オプションで履歴機能を無効にできる。また，コマンドラインの先頭で Ctrl+H を入力すると，全ての履歴が番号と共に表示される。!番号のようにすれば，履歴を直接利用できる。履歴の長さなどを設定する環境変数を表 2.4 に示す。

D.7.1　倒立振子の制御のシミュレーション

関数 Ode45HybridAuto() を使ってディジタル制御のシミュレーションを行なうプログラムを示す。まず，メイン関数でパラメータの設定，プラントとコントローラの状態の初期値の設定を行なって，シミュレーションのための関数を呼び出す。

```
Real a32, a33, a34, a35, a42, a43, a44, a45;
Real b3, b4, alpha;
Matrix C, F, Ah, Bh, Ch, Dh, Jh, Xo;

Func void main()
{
  Real t0, tf, dt;
  Matrix TC, XC, UC, x0;
  void para_init(), diff_eqs(), link_eqs();

  para_init();                  // パラメータの初期化
  t0 = 0.0; tf = 3.0; dt = 0.005;

  x0 = [0 30.0/180.0*PI 0 0]'; // 初期状態
  Xo = Z(2,1);                  // 推定値の初期値

  {TC, XC, UC} =
    Ode45HybridAuto(t0, tf, dt, x0, diff_eqs, link_eqs);
}
```


2. 局所変数の占有した領域を関数終了時に解放するために，関数内で使用するクラスに応じて，関数の始めと終りに次の宣言と文を書く。

```
void foo()
{
  Matrix     mat_last = MAT_DEF;
  Polynomial poly_last = POLY_DEF;
  Rational   rat_last  = RAT_DEF;
  List       list_last = LIST_DEF;

  ......................
  MatFrees(mat_last);
  PolyFrees(poly_last);
  RatFrees(rat_last);
  ListFrees(list_last);
}
```

3. 局所変数の行列，多項式，有理関数，リストは，宣言と同時に初期化する。

```
void foo()
{
  ......................
  Matrix     aa = MAT_DEF,  bb = MAT_DEF;
  Polynomial cc = POLY_DEF, dd = POLY_DEF;
  Rational   ee = RAT_DEF,  ff = RAT_DEF;
  List       gg = LIST_DEF, hh = LIST_DEF;
  ......................
}
```

`Real rtSetClock(stime);` `  Real stime;`	サンプリング時間を秒単位で設定する。
`void rtStartRehearsal();`	on_task() の実行に必要なメモリを調べるため on_task() をテスト実行する前に呼び出す。
`void rtStopRehearsal();`	on_task() のテスト実行を終了する。on_task() の実行に必要なメモリがメモリ管理リストへ追加される。
`Integer rtIsTimeOut();`	サンプリング時間内にリアルタイム処理が終わったか調べる。終わらなければ 1 が，終われば 0 が戻り値となる。
`Integer rtIsRunning();`	リアルタイム処理中かどうか調べる。リアルタイム処理中なら 1，そうでなければ 0 が戻り値となる。
`Integer rtIsTesting();`	リアルタイム処理のテスト中かどうか調べる。テスト中なら 1 が，そうでなければ 0 が戻り値となる。
`Integer Inport(portid);` `  Integer portid;`	portid で指定したハードウェアポートから 1 ワードを読み込む。
`Integer Inportb(portid);` `  Integer portid;`	portid で指定したハードウェアポートから 1 バイトを読み込む。
`void Outport(portid, value);` `  Integer portid;`	portid で指定したハードウェアポートに value が指定する 1 ワード

登録されている変数の詳しい情報を表示するには whos コマンドを使用する。変数を指定しないと，登録されている全ての変数の情報が表示される。

```
whos
whos a, b, A, B
```

3.6　変数の消去

変数を消去するには，clear コマンド[2]に変数名をコンマで区切って指定する。

```
clear a, b, A, B
```

3.7　変数の存在を調べる (matx のみ)

ある変数が定義されているか調べるには，変数名を引数として関数 exist() を呼ぶ。局所変数として定義されている場合は，2，大域変数としてのみ定義されている場合は，1，定義されていない場合は，0 が返される。

```
i = exist("A");
if (i == 0)
  print "A dosen't exist.";
} else if (i == 1) {
  print "A is a global variable.";
} else if (i == 2) {
  print "A is a local variable.";
}
```

[2] clear コマンドは画面のクリアに使用する。

```
// カウンタの初期化
Func void count_init()
{
  Integer i, ch_port;

  void count_mode(), count_set();

  for (i = 1; i <= CNT_CH; i++) {
    ch_port = 5 * (i - 1);
    count_mode(ch_port);
    count_set(ch_port, 0);
  }
}
```

カウンタの値を読み出す関数の例を次に示す。

```
Func Integer count_get(ch)
  Integer ch;
{
  Integer ch_port;
  Integer d1, d2, d3, data;

  ch_port = 5 * (ch - 1);

  Outportb(CNT_COM, 0x14);
  Outportb(CNT_DAT, 0x7);
  Outportb(CNT_COM, ch_port);

  d1 = Inportb(CNT_DAT);
  d2 = Inportb(CNT_DAT);
  d3 = Inportb(CNT_DAT);

  data = d3*0x10000 + d2*0x100 + d1;

  if (data > 0x800000) {
    data = data - 0xffffff - 1;
  }

  return data;
}
```



```
  warning("WARNING: Poles lie near the imaginary axis.\n");
  warning("Poles lie on the imaginary axis at (%f, %f)\n",
          Re(p), Im(p));
  warning("");
```

3.16　エラー停止

error() 関数を使用すると，エラーメッセージが表示され，実行が中止される。メッセージが標準エラー出力に出力される。長さが 0 の文字列を引数にして error() を呼ぶと，エラーは無視される。引数がない場合は，単に実行が停止する。この関数には printf() と同じ書式を指定できる。

```
error("ERROR: Zero division\n");
error("Dimension error A(%d, %d), B(%d, %d)\n",
       Rows(A), Cols(A), Rows(B), Cols(B));
error("");
error();
```

3.17　ヘルプ

help コマンドを実行すると関数一覧が表示される。help topic を実行すると，指定したトピックスに関する情報が表示される。topic がヘルプファイル中で見つからない場合は，topic.mm という mm ファイルが探される。そして，そのファイルが存在した場合，ファイルの先頭部分のコメント (/* と*/で囲まれた部分) が画面に表示される。ヘルプメッセージが表示されるとき，環境変数 PAGER に設定されているページャーが使われる。何も設定されていないとき，more が使われる。なお，環境変数 LANG に ja，japanese，ja_JP.xxx が設定されていれば，日本語でヘルプが表示される。

D.4.5　センサとアクチュエータのインターフェース関数

センサやアクチュエータとのインターフェースは次の 5 つの関数を使って行なう。

`void sensor_init();`	センサの初期化
`void actuator_init();`	アクチュエータの初期化
`void actuator_stop();`	アクチュエータの停止
`void actuator();`	アクチュエータへの出力
`Matrix sensor();`	センサからの入力

センサの初期関数の例を次に示す。

```
Func void sensor_init()
{
  void count_init();

  // 変換係数の設定 (m/カウント数)
  c_k = [[0.07489506 * 0.001]
         [0.04600000 * 0.001]
         [0.07501935 * 0.001]];

  count_init(); // カウンタの初期化
}
```

アクチュエータの初期関数の例を次に示す。

`CM1 = CoMatrix(実行列|配列)`	引数を実部とする複素行列
`PM1 = PoMatrix(実行列|配列)`	引数を定数項とする多項式行列
`CPM1 = PoMatrix(複素行列|配列)`	引数を定数項とする複素多項式行列
`RM1 = RaMatrix(実行列|配列)`	引数を分子の定数項とする有理関数行列
`CRM1 = RaMatrix(複素行列|配列)`	引数を分子の定数項とする複素有理関数行列
`RM1 = RaMatrix(多項式行列|配列)`	引数を分子とする有理関数行列

`A1 = Array(行列)`	行列と同じ成分を持つ配列
`A1 = Array(多項式)`	多項式の係数を昇べきの順に並べた配列
`A1 = Array(実数 r)`	(1,1) 成分が r の 1x1 実配列
`CA1 = Array(複素数 c)`	(1,1) 成分が c の 1x1 複素配列

`CM1 = CoArray(実配列|行列)`	引数を実部とする複素配列
`PM1 = PoArray(実配列|行列)`	引数を定数項とする多項式配列
`CPM1 = PoArray(複素配列|行列)`	引数を定数項とする複素多項式配列
`RM1 = RaArray(実配列|行列)`	引数を分子の定数項とする有理関数配列
`CRM1 = RaArray(複素配列|行列)`	引数を分子の定数項とする複素有理関数配列
`RM1 = RaArray(多項式配列|行列)`	引数を分子とする有理関数配列

`I1 = Index(行列)`	整数に近似した成分を持つ指数
`I1 = Index(配列)`	整数に近似した成分を持つ指数
`I1 = Index(整数 i)`	第 1 成分が i の指数
`I1 = Index(実数 r)`	第 1 成分が r を近似した整数である指数

D.4.1　メイン関数

基本的なメイン関数 main() の例を次に示す。

```
Func void main()
{
  para_init();              // パラメータの設定
  var_init();               // 変数の初期化
  machine_ready();          // 実験装置（電源など）の準備

  rtSetClock(stime);        // サンプリング時間（秒単位）設定
  rtSetTask(on_task);       // リアルタイムタスクを設定
  rtSetBreak(break_task); // Ctrl-C による割込の処理を設定

  rtStart();                // リアルタイム処理を開始
  off_task_loop();          // 画面表示，コマンドの入力
  rtStop();                 // リアルタイム処理を停止

  machine_stop();           // 実験装置（電源など）の停止
}
```

コメントの説明を補足する。

- Real 型の変数 stime にはサンプリング時間が秒単位で設定されている。

- void 型の関数 break_task() はコントロール+C などの割り込みキーが押された時に呼ばれる関数であり，リアルタイム処理の停止などを記述しておく。
- 関数 rtStart() が実行されると，サンプリング時間毎に void 型の関数 on_task() が実行される。
- 関数 off_task_loop() では，変数の表示やコマンドの入力待ちのループが記述されており，この関数の実行が終わると，関数 rtStop() が呼ばれて，リアルタイム処理が停止する。

したがって，コメントはプログラムを理解しやすくするために自由に使うことができる。コメントを入れる場合は，空白 (blank) あるいは改行が書けるところならどこでもよい。ただし，“/*” と “*/” によるコメントの入れ子 (ネスティング) は許されない。

3.26　プリプロセッサ

MATX は，C プリプロセッサを呼び出すことによって，言語仕様の拡張を行なっている。拡張された言語仕様のうち，最も良く使われる機能は，文字綴を任意の文字列に置換するための#define と別のファイルを読み込むための#include である。デフォルトで呼び出される C プリプロセッサは，UNIX では cpp，Windows では各 C コンパイラ (例えば Visual C++の場合 cl) であるが，次に示すように別のプリプロセッサを環境変数 MATXCPP に設定することができる。

```
setenv MATXCPP /usr/local/lib/gcc-lib/cpp
```

オプション -nocpp を指定すると，C プリプロセッサが起動されなくなる。オプション -Dname=def によって C プリプロセッサに引数を渡せる。

また，インタプリタ matx の実行時には__MATX__が，コンパイラ matc の実行時には__MATC__が自動的に 1 に定義される。これらを用いるとプログラム中の実行される箇所をインタプリタとコンパイラで容易に変更できる。例えば関数のデバッグ時にその関数を matx でスクリプトとして実行したいとき以下のようすれば良い。この例では，matx を実行するとプログラムはスクリプトとして実行され，A, B, C は大域変数として定義されるので，コマンドラインからそれらを参照できる。matc を実行すれば，関数 afo() が定義される。

る DJGPP 版がある。Borland C 版が生成するリアルモードのプログラムは使用できるメモリ容量が 640KB 以下であり，ツールボックスの関数をほとんど組み込めないなどの制約をうけるが，タイムアウトエラーの検出が正しく行われる。一方，DJGPP の gcc が生成するプロテクトモードのプログラムはパソコンに実装されたメモリを全て使えるので，ほとんどメモリの制約がなく，ツールボックスの関数をたくさん組み込める。

また，DJGPP 版の RTMATX が生成したプログラムは，ハードリアルタイム性は保証されないものの，Windows95 の DOS 窓で使用できることが確認されている。DOS 窓でリアルタイム処理プログラムを実行するまえに，他の Windows のアプリケーションを終了することを勧める。

リアルタイム処理プログラムの作成手順は，DJGPP 版と Borland C 版で異なる。以下，それぞれについて説明する。

D.2.1　プロテクトモード RTMATX(DJGPP 版)

DJGPP 版[1]の RTMATX はプロテクトモードで動作するので，MS-DOS のメモリ制約を受けることなく，常駐させるデバイスドライバの選択に悩む必要がない。そして，パソコンに実装したメモリを全て使えるので大量のデータを保存することもできる。

リアルタイム処理プログラムを作成する手順例を次に示す。

```
A:> rtmatc sample.mm
```

rtmatc により実行可能プログラム sample.exe が生成される。複数のファイルから実行プログラムを生成する場合，以下のような Makefile を作成しておくと便利である。

[1] DJGPP は MATX の FTP サイトからフリーで入手できる


```
#if constant-expression
```

は，定数式がゼロでないかチェックする。次の制御文

```
#ifdef identifier
```

は，identifier が定義されているかチェックする。すなわち，identifier が既に #define によって定義されたかどうかをチェックする。次の制御文

```
#ifndef identifier
```

は，identifier が定義されていないかチェックする。これらの制御文の後には，任意のソース と制御文

```
#else
```

と

```
#endif
```

が続くことができる。もし，チェックされた条件が真のとき，#else と #endif の間の行は無視される。もし，チェックされた条件が偽のとき，テストと #else あるいは，#endif の間の行は無視される。

3.27　スタートアップファイル

matx 及び matc を起動すると，以下の順序でスタートアップファイルが検索されて，ファイルが存在していれば，読み込まれる。ただし，どれか 1 つのファイルが検出されると，ファイルの検索はその時点で中止される。つまり，スタートアップファイルは，1 つしか読み込まれない。

第4章　ファイル

4.1　mm ファイル の読み込み (matx のみ)

load コマンドの引数に mm ファイルの名前を文字列として指定することによって mm ファイルを読み込むことができる。拡張子を省略すると，拡張子は mm となる。コンマで区切ることにより，複数の mm ファイルを同時に指定できる。mm ファイルは，

1. カレントディレクトリ
2. 環境変数 MATXINPUTS で定義されているディレクトリ以下の全てのディレクトリ
   （コロン “:” で区切って複数のディレクトリを指定できる。MS-DOS バー ジョンでは，セミコロン “;” で区切る。）
3. デフォルト **mm** ファイルディレクトリ以下の全てのディレクトリ (コンパイル時に決定される)

の順番で探索される。環境変数 LANG に ja，japanese，ja_JP.xxx が設定されていて，各ディレクトリに ja というディレクトリが存在すれば，そこが先に検索される。

何も指定しないと，カレントディレクトリにファイル MaTXRC.mm が存在すれば，そのファイルが読み込まれる。

C.10 符合反転 (Negation)

整数	= -	整数
実数	= -	実数
複素数	= -	複素数
実多項式	= -	実多項式
複素多項式	= -	複素多項式
実有理多項式	= -	実有理多項式
複素有理多項式	= -	複素有理多項式
実 (行列|配列)	= -	実 (行列|配列)
複素 (行列|配列)	= -	複素 (行列|配列)
実多項式 (行列|配列)	= -	実多項式 (行列|配列)
複素多項式 (行列|配列)	= -	複素多項式 (行列|配列)
実有理多項式 (行列|配列)	= -	実有理多項式 (行列|配列)
複素有理多項式 (行列|配列)	= -	複素有理多項式 (行列|配列)

C.11 {ポスト, プリ}{インクリメント, デクリメント}

MATX には変数のインクリメントとデクリメントに対して，C 言語のそれと同様な 2 つの演算子がある。インクリメント演算子 ++ はその被演算数に 1 を加え，デクリメント演算子 -- は 1 を引くものである。演算子 ++ と -- は前置演算子と後置演算子のどちらにも使える。

```
整数++      // 整数 = 整数 + 1
実数++      // 実数 = 実数 + 1.0
整数--      // 整数 = 整数 - 1
実数--      // 実数 = 実数 - 1.0
++整数      // 整数 = 整数 + 1
++実数      // 実数 = 実数 + 1.0
--整数      // 整数 = 整数 - 1
--実数      // 実数 = 実数 - 1.0
```



```
// 行列 A を int 型のバイナリデータとして保存する
A = [1 2 3 4 5];
fd = fopen("data", "w");
fwrite(fd, A, "int");
fclose(fd);

// 5 個の int 型のバイナリデータを行列 B に読み込む
fd = fopen("data", "r");
B = fread(fd, 5, "int");
fclose(fd);

// 3 個の double 型のバイナリデータを Endian を変更
// して行列 C に読み込む
fd = fopen("data", r");
C = fread(fd, 3, "double", 1);
fclose(fd);
```

なお，使用中の計算機のバイナリの数値形式 (Endian) は，関数 machine_endian() で調べることができる。この関数は IEEE Little Endian (PC, 386, 486, etc.) のとき 0 を返し，IEEE Big Endian (SPARC, Motorola, etc.) のとき 1 を返す。

4.2.4 一般データの保存と読み込み

値を保存するには，クラスの区別なく save コマンドを使用する。保存する値をコンマで区切って並べ，最後にファイル名を指定する。データは，**mm** ファイルと同じ形式で保存される。拡張子を省略すると，拡張子は mm となる。

インタプリタ (matx) の場合，ファイル名だけを指定すると全ての広域変数が保存され，何も指定しないとファイル MaTX.mm に全ての広域変数が保存される。

```
        整数          = 整数 / 整数
        実数          = 実数 / 実数
        複素数        = 複素数 / 複素数

        実多項式      = 実多項式 / (整数|実数)
        複素多項式    = 実多項式 / 複素数
        複素多項式    = 複素多項式 / (整数|実数)
        複素多項式    = 複素多項式 / 複素数

        実有理多項式  = 実多項式 / 実多項式
        複素有理多項式 = 複素多項式 / 複素多項式
        実有理多項式  = 実有理多項式 / 実有理多項式
        複素有理多項式 = 複素有理多項式 / 複素有理多項式

実 (行列|配列)              = 実 (行列|配列) / 実 (行列|配列)
複素 (行列|配列)            = 複素 (行列|配列) / 複素 (行列|配列)
実多項式 (行列|配列)        = 実多項式 (行列|配列) / (整数|実数)
複素多項式 (行列|配列)      = 実多項式 (行列|配列) / 複素数
複素多項式 (行列|配列)      = 複素多項式 (行列|配列) / (整数|実数)
複素多項式 (行列|配列)      = 複素多項式 (行列|配列) / 複素数
実有理多項式 (行列|配列)    = 実有理多項式 (行列|配列) / (整数|実数)
複素有理多項式 (行列|配列) = 実有理多項式 (行列|配列) / 複素数
複素有理多項式 (行列|配列) = 複素有理多項式 (行列|配列) / (整数|実数)
複素有理多項式 (行列|配列) = 複素有理多項式 (行列|配列) / 複素数

    実行列          = 実行列 ./ 実行列
    複素行列        = 複素行列 ./ 複素行列
    有理多項式行列 = (実|複素|多項式|有理多項式) 行列 ./ 多項式行列

    整数            = 整数 \ 整数
    実数            = 実数 \ 実数
    複素数          = 複素数 \ 複素数

    実有理多項式    = 実多項式 \ (整数|実数)
    複素有理多項式 = 実多項式 \ 複素数
    複素有理多項式 = 複素多項式 \ (整数|実数)
    複素有理多項式 = 複素多項式 \ 複素数
```


4.7　ディレクトリを変更する

作業ディレクトリを変更するには，コマンド `chdir` を使用する。ただし，文字列中の `\t` や `\n` などは特殊文字として解釈されるので，バックスラッシュ `\` を重ねる必要がある。

```
chdir "/usr/local/tmp"     // UNIX
chdir "c:/tmp"             // DOS(1)
chdir "c:\\tmp"            // DOS(2)
```

C.2 差 (Subtraction)

```
整数                          = 整数 - 整数
実数                          = 実数 - 実数
複素数                        = 複素数 - 複素数
実多項式                      = 実多項式 - 実多項式
複素多項式                    = 複素多項式 - 複素多項式
実有理多項式                  = 実有理多項式 - 実有理多項式
複素有理多項式                = 複素有理多項式 - 複素有理多項式
実 (行列|配列)                = 実 (行列|配列) - 実 (行列|配列)
複素 (行列|配列)              = 複素 (行列|配列) - 複素 (行列|配列)
実多項式 (行列|配列)          = 実多項式 (行列|配列) - 実多項式 (行列|配列)
複素多項式 (行列|配列)        = 複素多項式 (行列|配列)
                                - 複素多項式 (行列|配列)
実有理多項式 (行列|配列)      = 実有理多項式 (行列|配列)
                                - 実有理多項式 (行列|配列)
複素有理多項式 (行列|配列)    = 複素有理多項式 (行列|配列)
                                - 複素有理多項式 (行列|配列)
(行列|配列|指数)              = (行列|配列|指数)
                                .- (整数|実数|複素数|多項式|有理多項式)
(行列|配列|指数)              = (整数|実数|複素数|多項式|有理多項式)
                                .- (行列|配列|指数)
```


示している．

```
 Name :  [ B ]
 Size :  [  30, 10 ]   Filename : B

          (  3)-Real     (  3)-Imag     (  4)-Real     (  4)-Imag
( 12)   5.763537E-01   3.227756E-01   2.890290E-01   8.225142E-01
( 13)   2.218456E-01   1.713277E-01   5.039103E-01   1.292042E-01
( 14)   2.112349E-01   7.294220E-02   6.336816E-01   4.309787E-01
( 15)   8.716954E-01   9.988188E-02   6.955673E-01   7.323241E-01
( 16)   6.781542E-01   8.113680E-01   5.772123E-01   8.804363E-01
( 17)   6.445914E-01   9.994860E-01   2.568782E-02   8.759036E-03
( 18)   6.485325E-01   5.361216E-01   2.513150E-01   9.766774E-01
( 19)   2.967033E-01   7.648637E-01   6.800593E-01   4.797411E-01
( 20)   8.320193E-01   2.827639E-01   3.725870E-01   1.078266E-01
( 21)   8.935846E-01   1.882765E-01   9.587078E-01   9.792556E-01
( 22)   5.605379E-01   2.117229E-01   6.072429E-01   9.438229E-01
( 23)   8.295992E-01   1.647209E-01   5.731654E-01   7.800235E-01
( 24)   2.509799E-01   8.080054E-01   2.564957E-01   7.973468E-01
( 25)   3.942465E-01   4.115652E-01   5.600023E-01   5.074672E-01
( 26)   9.020740E-01   6.695332E-02   2.406348E-01   4.392214E-01
```

図 5.2: 行列エディタ (複素行列)

```
 Name :  [ Re(B) ]
 Size :  [  30, 10 ]

            (  3)          (  4)          (  5)          (  6)
( 12)   5.763537E-01   2.890290E-01   3.210669E-01   2.613233E-01
( 13)   2.218456E-01   5.039103E-01   6.703021E-02   3.930634E-01
( 14)   2.112349E-01   6.336816E-01   5.356539E-02   7.834114E-01
( 15)   8.716954E-01   6.955673E-01   9.301896E-01   4.550905E-01
( 16)   6.781542E-01   5.772123E-01   2.731182E-01   9.353880E-01
( 17)   6.445914E-01   2.568782E-02   8.413805E-01   7.701880E-02
( 18)   6.485325E-01   2.513150E-01   2.286443E-01   2.509221E-01
( 19)   2.967033E-01   6.800593E-01   8.331472E-01   8.763075E-01
( 20)   8.320193E-01   3.725870E-01   7.570482E-01   1.092149E-01
( 21)   8.935846E-01   9.587078E-01   1.217351E-01   9.815443E-01
( 22)   5.605379E-01   6.072429E-01   8.164395E-01   4.465349E-01
( 23)   8.295992E-01   5.731654E-01   1.047806E-01   7.329764E-01
( 24)   2.509799E-01   2.564957E-01   3.381931E-01   3.882216E-01
( 25)   3.942465E-01   5.600023E-01   3.108211E-01   8.225768E-01
( 26)   9.020740E-01   2.406348E-01   3.731643E-01   7.522860E-01
```

図 5.3: 行列エディタ (実部行列)

第6章　グラフィックス

作成したデータをプロットするにはグラフィック関数を使って gnuplot や xplot を呼び出す。

6.1　gnuplot

gnuplot を使ってデータをプロットするには以下の関数を使う。

表 6.1: グラフィック関数 1(gnuplot)

関数名	引数	機能
gplot(Y)		
gplot(X, Y)	Array X, Y	線形 x–y プロット
gplot(X, Y, titles)	List titles	
gplot(X, Y, titles, cmd)	String cmd	
mgplot(win, Y)	Integer win	
mgplot(win, X, Y)	Array X, Y	線形 x–y プロット
mgplot(win, X, Y, titles)	List titles	
mgplot(win, X, Y, titles, cmds)	List cmds	

グラフィック関数 (`gplot()`, `mgplot()`) 使うと，自動的に gnuplot が起動され，データがプロットされる。X-Window で gnuplot を使うには，MATX を起動する前に環境変数 `DISPLAY` に使っている端末の名前を設定しておく必要がある。

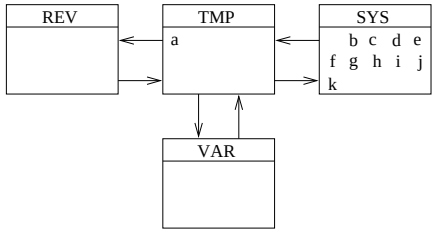


図 B.3: 行列のメモリ管理 (2)

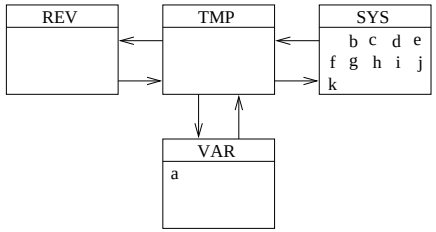


図 B.4: 行列のメモリ管理 (3)

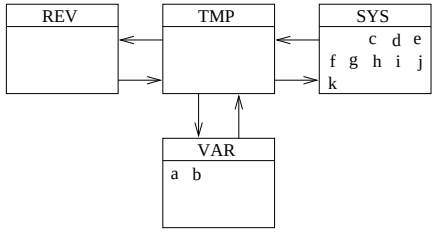


図 B.5: 行列のメモリ管理 (4)

算結果が入っている。式 (4) の右辺が左辺に代入されたときのメモリ状態は図 B.9 のようになる。$(A+B)*(A-B)$ の計算結果である f だけが VAR 状態となり，d と e は REV 状態となる。式 (5) の右辺が評価されたときのメモリの状態は図 B.10 のようになっている。d, e, g には，それぞれ $(C+D)$,

6.1.3　対数プロット

関数 gplot_semilogx() を使うと，X 軸が $\log_{10}$，Y 軸が線形スケール (片対数スケール) でデータがプロットされる。関数 gplot_semilogy() を使うと，Y 軸が $\log_{10}$，X 軸が線形スケール (片対数スケール) でデータがプロットされる。関数 gplot_loglog() を使うと，X 軸と Y 軸が共に $\log_{10}$スケール (両対数スケール) でデータがプロットされる。

対数プロットでは，最初からグリッド線が表示される。グリッド線を消去するには，関数 gplot_grid(0) や関数 mgplot_grid(win,0) を使えばよい。

関数 gplot_semilogx()(gplot_semilogy(), gplot_loglog()) の使い方は，関数 (gplot()) のそれと同じである。

関数 mgplot_semilogx()(mgplot_semilogy(), mgplot_loglog()) の使い方は，関数 mgplot() のそれと同じである。

1 つのグラフ上に複数の対数プロット描くには，関数 greplot() や mgreplot() と同様に，次に示す関数を使えばよい。これらの関数を使うと，古い線に新しい線が重ねてプロットされる。

表 6.5: グラフィック関数 5(gnuplot)

関数名	引数	機能
gplot_subplot(rows,cols,num)	Integer rows Integer cols Integer num	ウィンドウを rows x cols に分割 num 番目をカレント面に設定。
mgplot_subplot(win,rows,cols,num)	Integer win Integer rows Integer cols Integer num	ウィンドウを rows x cols に分割 num 番目をカレント面に設定。

を呼び出すと，ラベル，タイトル，グリッド等が初期状態に戻る。

表 6.6: グラフィック関数 6(gnuplot)

関数名	引数	機能
gplot_clear()		ウィンドウのクリア
gplot_reset()		ウィンドウのリセット
gplot_quit()		ウィンドウの消去
mgplot_clear(win)	Integer win	ウィンドウのクリア
mgplot_reset(win)	Integer win	ウィンドウのリセット
mgplot_quit(win)	Integer win	ウィンドウの消去
mgplot_quit()	Integer win	全ウィンドウの消去

5. 5 番目の整数は，データの名前の長さを表す。

データ型	名前	内容
Matrix	namelen	名前の長さ
Integer	namelen	名前の長さ
Real Number	namelen	名前の長さ
Complex Number	namelen	名前の長さ
String	namelen	名前の長さ
Polynomial	namelen	名前の長さ
Rational	namelen	名前の長さ
List	namelen	名前の長さ
Array	namelen	名前の長さ
Index	namelen	名前の長さ

20 バイトのヘッダーの次に，namelen 長の変数の名前が続く。最後の文字は，(0 でコード化される)NUL 文字である。この後のデータは，データ型によってことなる。

整数　　データの精度に応じた整数 (2 バイトまたは，4 バイト)。

実数　　データの精度に応じた実数 (4 バイトまたは，8 バイト)。

複素数　　2 個のデータの精度に応じた実数 (4 バイトまたは，8 バイト)。

文字列　　stringlen 長の文字列。最後の文字は，NUL 文字である。

多項式　　実多項式　　degree + 1 個の精度に応じた実数 (4 バイトまたは，8 バイト)。

表 6.8: グラフィック関数 8(gnuplot)

関数名	引数	機能
gplot_psout(file, cmd)	String file String cmd	ポストスクリプト ファイルに保存
mgplot_psout(win, file, cmd)	Integer win String file String cmd	ポストスクリプト ファイルに保存 cmd = オプション
gplot_figcode(file, cmd)	String file String cmd	FIG コード ファイルに保存
mgplot_figcode(win, file, cmd)	Integer win String file String cmd	FIG コード ファイルに保存 cmd = オプション

```
w = [-PI:PI];
mgplot_hold(1, 1);
mgplot_subplot(1, 2, 1, 1);
mgplot(1, w, sin(w));
mgplot_subplot(1, 2, 1, 2);
mgplot(1, w, cos(w));
mgplot_hold(1, 0);
```

表 6.9: グラフィック関数 9(gnuplot)

関数名	引数	機能
gplot_hold(on_off)	Integer on_off	on_off = 0: ホールドオフ on_off = 1: ホールドオン
mgplot_hold(win, on_off)	Integer win Integer on_off	on_off = 0: ホールドオフ on_off = 1: ホールドオン

をファイルに保存すると，

```
# 2 3 C
  1.0000   2.0000   7.0000   8.0000
  3.0000   4.0000   9.0000   10.000
  5.0000   6.0000   11.000   12.000
```

となる。フォーマット規則の 3 と 4 より，次のようなファイルを読み込むこともできる。

```
# 2 2
#
# This file contains data of simulation (1)
#
1.00 2.00  # This is the end of first column
# This line is commented out.
3.00 4.00  # This is the end of second column
#
# End of file
#
```

A.2　MX データフォーマット

データファイルは，1 個以上の変数を含み，変数は，連続的に保存されている。それぞれの変数は，変数の属性を記述した情報を含む 20 バイトのヘッダーから始まる。20 バイトのヘッダーは，5 個の 4 バイト長の整数から構成されている。

1. 変数のタイプを表す整数。この整数を，[M|O|P|C] と表すと，

   M は，ファイルを書いたマシーンを表す。

- もし xplot(Y) のように X が指定されなかったら，Y の各成分とその成分の指数を対応させて異なった線種で線を描く。

1 つのグラフ上に複数の線をプロットする 2 つ目の方法は，関数 xplot() に複数の引数を渡す方法である。

```
xplot(X1, Y1, X2, Y2, ..., X5, Y5);
```

変数 X1, Y1, X2, Y2 等は，行列の対である。それぞれの x-y 対が 1 つのグラフ上にプロットされる。この方法を使えば，異なった長さの線を同一グラフ上にプロットすることができる。

6.2.3　対数プロット

関数 xplot_semilogx() を使うと，片対数スケールでデータがプロットされる。すなわち，X 軸が $\log_{10}$，Y 軸が線形スケールとなる。

表 6.12: グラフィック関数 3(xplot)

関数名	引数	機能
xplot_semilogx(X, Y)	Array X, Y	セミログ x–y プロット (x 軸 ログ)

関数 xplot_semilogx() の使い方は，関数 xplot() のそれと同じである。

6.2.4　ウィンドウ・マネ– ジャ– の設定

X ウィンドウ・システムのウィンドウ・マネ– ジャ– として twm を使う場合，以下の記述を各自のウィンドウ・マネ– ジャ– の設定ファイル .twmrc に加えることによって，xplot を使用するときに，キャンバス・ウィンドウだ

[9] Masanobu Koga and Katsuhisa Furuta. MaTX : A high-performance interactive software package for scientific and engineering computation. *Proc. of CADCS'91, Swansea, U.K.*, pages 39–44, 1991.

[10] Masanobu Koga and Katsuhisa Furuta. MaTX : A high-performance programming language (interpreter and compiler) for scientific and engineering computation. *Proc. of CACSD'92, Napa, U.S.A.*, pages 15–22, 1992.

[11] A. J. Laub. A schur method for solving algebraic riccati equations. *IEEE Trans. on AC*, 24(6), 1979.

[12] C. Moler, J. Littel, and S. Bangert. *PC-MATLAB – User's Guide*. N. Main St., Sherborn, MA 01770, USA, 3.1 edition, 1987.

[13] Cleve Moler. *MATLAB – User's Guide*. Alberquerque, USA, 1980.

[14] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Veterling. *Numerical Recipes in C (The Art of Scientific Computing)*. Cambridge University Press, 1988.

[15] B.T. Smith, J.M. Boyle, J.J. Dongarra, B.S. Garbow, Y. Ikebe, V.C. Klema, and C.B. Moler. *Matrix Eigensystem Routines – EISPACK Guide*. Springer-Verlag, 1976.

[16] G.W. Stewart. *Introduction To Matrix Computations*. Academic Press, 1973.

[17] Robert Walker, Charles Gregory Jr., and Sunil Shah. MATRIXx: A data analysis, system identification, control design and simulation package. *Control Systems Magazine*, pages 30–37, 1982.

[18] 近藤 嘉雪. *yacc* による *C* コンパイラプログラミング. ソフトバンク, 1990.

第7章　変数

コンパイラ (matc) を使う場合，全ての変数は使う前に宣言しておかなければならない。変数には局所変数と大域変数がある。 局所変数は関数の先頭の実行可能文の前で宣言し，関数内でのみ有効である。局所変数が占有したメモリ領域は，その関数が終了すると自動的に解放される。一方，大域変数は関数 `main()` が定義される前に宣言されなければならない。大域変数は，大域的にアクセスできるから，それらは関数間のデータのやりとりについては，関数の引数と戻り値の代用にすることができる。その名前が何らかの形で宣言されていれば，大域変数は，それを名前で参照することによって，どの関数からもアクセス可能である。そうした関数の間で数多くの変数を共用しなければならない時には，長い引数リストよりも大域変数の方が便利である。しかしながら，この方法は注意して用いねばならない。なぜなら，それはプログラム構造に悪い影響を与え，関数間で多くのデータが結合したプログラムを作ることになるからである。

なお，大域変数は 0 に初期化されることが保証されるので，ユーザが陽に初期化する必要はない。リストは成分が 0 個のリストとして初期化される。ただし，行列は 0 x 0 の行列として初期化されるので，`length()` 関数を用い大きさを調べるなどして，陽に初期化する必要がある。一方，局所変数は使用する前に必ず初期化する必要がある。

7.1　変数の宣言

変数はクラス名に続けてそのクラスの変数を 1 個以上コンマ ",” で区切って宣言する。変数名は，英文字または下線 _ で始まる任意長の英数字文字列

```
Func void diff_eqs(DX, t, X, U)
  Real t;
  Matrix X, DX, U;
{
  Matrix xp, xd, xc, xf;
  Matrix dxp, dxd, dxc, dxf;
  Matrix up, ud, uc, uf;

  {xp, xd, xc, xf} = VecChop(X, state_idx);
  {up, ud, uc, uf} = VecChop(U, input_idx);

  dxp = Ap*xp + Bp*up;
  dxd = Ad*xd + Bd*ud;
  dxc = Ac*xc + Bc*uc;
  dxf = Af*xf + Bf*uf;
  DX = [[dxp][dxd][dxc][dxf]];
}

Func void link_eqs(U, t, X)
  Real t;
  Matrix U, X;
{
  Matrix xp, xd, xc, xf;
  Matrix up, ud, uc, uf;
  Matrix yp, yd, yc, yf;

  {xp, xd, xc, xf} = VecChop(X, state_idx);

  yp = Ap*xp;
  yd = Ad*xd;
  yc = Ac*xc;
  yf = Af*xf;

  up = yc - yf;
  ud = up;
  uc = u;
  uf = yp + yd;
  U = [[up][ud][uc][uf]]';
}
```

List 8:　複雑なシステムのシミュレーション

実数 = 整数	整数は実数に変換される。
複素数 = 整数	整数は実数，複素数に変換される。
複素数 = 実数	実数は複素数に変換される。
有理多項式 = 整数	整数は実数，有理多項式に変換される。
有理多項式 = 実数	実数は有理多項式に変換される。
有理多項式 = 複素数	複素数は有理多項式に変換される。
有理式 = 整数	整数は実数，有理多項式，有理式に変換される。
有理式 = 実数	実数は有理多項式，有理式に変換される。
有理式 = 複素数	複素数は有理多項式，有理式に変換される。
有理式 = 有理多項式	有理多項式は有理式に変換される。

```
Func void main()
{
  Matrix x0;
  Array TC, XC, UC;

  void diff_eqs(), link_eqs();

  x0 = [1 0 0]';
  {TC, XC, UC} =
     Ode45Auto(0.0, 20.0, x0, diff_eqs, link_eqs);
  print [[TC][XC][UC]] >> "cont.mat";
}

Func void diff_eqs(DX, t, X, UY)
  Real t;
  Matrix X, DX, UY;
{
  static Integer init;
  static Matrix Ap, bp, Ac, bc;
  Matrix dxp, dxc, xp, xc, up, uc;

  if (init == 0) {
    init = 1;
    Ap = [[0  1]        // 制御対象のパラメータ
          [2  1]];
    bp = [0 1]';
    Ac = [5];           // 制御器のパラメータ
    bc = [1];
  }

  xp = X(1:2,1);            // 制御対象の状態
  xc = X(3:3,1);            // 制御器の状態
  up = UY(1:1,1);           // 制御対象への入力
  uc = UY(2:2,1);           // 制御器への入力
  dxp = Ap*xp + bp*up;      // 制御対象の状態方程式
  dxc = Ac*xc + bc*uc;      // 制御器の状態方程式
  DX = [[dxp][dxc]];
}
```

List 7-3:　動的フィードバック制御のシミュレーション (静的変数使用)

poleasgn.mm に保存できるので，DOS のようにファイル名に厳しい制限のある OS でも，自由に関数名を決められる。

宣言されていて定義されていない関数が呼び出されると，その関数が定義されているファイルが，環境変数 MATXINPUTS で定義されているディレクトリ群やデフォルト mm ファイルディレクトリから再帰的に検索される。ファイルやディレクトリの数が多くなると検索に時間がかかるが，requrie 修飾子で関数が定義されているファイルを指定すれば，検索時間が短縮される。例えば，

```
List bode(...) requrie "control/bode.mm";
```

によって，関数 bode() は検索対象のディレクトリから相対パスが control/bode.mm であるファイルに定義されていることが分かるので短い時間で発見される。

8.2　関数の定義

関数は，関数定義子 Func に続けて K&R の関数定義と同様に記述する。ClassName にはその関数の返す値のクラス名を書く。値を返さない関数の ClassName には void を書く。関数内の個々の文の終りはセミコロン ";" で区切っておく。void 型以外の関数の場合，return 文を省略できない。

```
Func void plant(dx, t, x, u)
  Real t;
  Matrix x, dx, u;
{
  Matrix A, b;

  A = [[ 0   1   0]   // システム行列
       [ 0   0   1]
       [-2, -3, -4]];

  b = [0 0 1]';       // 入力行列

  dx = A*x + b*u;     // 状態方程式
}

Func Matrix controller(t, x)
  Real t;
  Matrix x;
{
  Matrix f, u;

  f = [-1, -1, -1];   // 状態フィードバック則
  u = f*x;
  return u;
}
```

List 6:　状態フィードバックのシミュレーション (2)

8.5　関数呼び出し (matx)

関数は名前で呼ぶが，その後には丸括弧内に引数を並べたものを付ける。引数がなくても丸括弧は省略できない。宣言されているが定義されていない関数が呼び出されれると，

1. その関数が require 修飾子付きで宣言されている場合，指定されたファイルが読み込まれる。ただし，ファイルが相対パスで指定されている場合，ファイルは，カレントディレクトリ，環境変数 MATXINPUTS が定義されていれば，そのディレクトリ群，デフォルト **mm** ファイルディレクトリの順番に探索される。

2. その関数が require 修飾子付きで宣言されていない場合，関数名と同じ mm ファイルが探索されて読み込まれる。mm ファイルは，カレントディレクトリ，環境変数 MATXINPUTS が定義されていれば，そのディレクトリ群，デフォルト **mm** ファイルディレクトリの順番に探索される。

MATXINPUTS で指定されたディレクトリ群とデフォルト **mm** ファイルディレクトリは再帰的に検索される。環境変数 LANG に ja，japanese，ja_JP.xxx が設定されていて，各ディレクトリに ja というディレクトリが存在すれば，そこが先に検索される。環境変数 MATXINPUTS には，コロン ":" で区切って複数のパスを指定することができるが，この環境変数を指定した場合，デフォルトの mm ファイルディレクトリは，探索されなくなる。デフォルトの mm ファイルディレクトリも探索して欲しい場合は，環境変数にデフォルトの mm ファイルディレクトリを指定する必要がある。

8.6　関数の引数渡し

関数の間でデータをやりとりする 1 つの方法は，呼び出しを行なう関数から相手の関数に対し，引数と呼ばれる値のリスト渡すやり方である。整数，

17.5　連続系システムのシミュレーション

17.5.1　状態フィードバック制御のシミュレーション (1)

連続系線形システム

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -2 & -3 & -4 \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u$$

に状態フィードバック

$$u = \begin{bmatrix} -1 & -1 & -1 \end{bmatrix} x$$

を用いた連続時間制御とハイブリッド制御のシミュレーションを 4 次のルンゲ- クッタの公式にしたがって，きざみを自動調節しながら行なう [14]。

関数 main() はパラメータと初期状態を設定してシミュレーションを行ない，結果をファイル cont.mat に保存する。diff_eqs() は動的要素の微分方程式を記述するための関数であり，link_eqs() は動的要素の接続関係を記述するための関数である。

関数 Ode45Auto() と Ode45HybridAuto() は陽の非線形関数の積分を行なう。関数 Ode45Auto() は連続時間制御のシュミレーションを行ない，関数 Ode45HybridAuto() はサンプル値系 (ディジタルコントローラによる連続時間プラントの制御) のシュミレーションを行なう。

関数 Ode45Auto() は，5 個の引数をとる。引数は，初期時刻 (0.0)，最終時刻 (5.0)，初期状態ベクトル (x0)，微分係数ベクトル (dx) を計算する関数の参照 (diff_eqs)，入力ベクトル (u) を計算する関数の参照 (link_eqs) の 5 個である。関数 Ode45HybridAuto() は，上述の 5 個の引数に加えて，一定入力 (u) がプラントに与え続けられるサンプリング時間 (dt) を引数にとる。

微分係数ベクトルを計算する関数 diff_eqs() と入力ベクトルを計算する関数 link_eqs() は，特別なフォーマットでなければならない。関数 diff_eqs() は 4 個の引数を持つ。第 1 引数 (dx) は，微分係数ベクトルを返

第 9 章　制御の流れ

MATX の制御の流れは，C 言語 [7, 20, 8, 22] の制御の流れとよく似ている。

9.1　If-Else

判定を行なう時は，if-else 文を用いる。if-else 文を用いる場合は，if-else の else 部は省略可能である。したがって，入れ子 (ネスト) になった if の連続で else が省略された場合，構文があいまいになってしまう。else を if に正しく関係付けるために大カッコ “{” と “}” を用い，省略することはできない。インタプリタは 1 行毎に解釈を行なうので，else 部が存在する時，else 部は if 部の最終行から始めなければならない。

```
if (expr) {
    statement-1;
} else {
    statement-2;
}
```

この if では，まず expr が計算される。それが真 (すなわち expr が 0 でない) であれば statement-1 が実行される。それが偽 (すなわち expr が 0) であり，かつ else 部が存在すれば statement-2 が実行される。

17.3　線形システムの伝達関数行列

線形システム

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \end{aligned}$$

の伝達関数行列

$$\begin{aligned} G(s) &= C(sI - A)^{-1}B \\ &= C\frac{1}{\det(sI - A)}(\Gamma_{n-1}s^{n-1} + \Gamma_{n-2}s^{n-2} + \cdots + \Gamma_0)B \\ \det(sI - A) &= s^n + \alpha_{n-1}s^{n-1} + \alpha_{n-2}s^{n-2} + \cdots + \alpha_0 \end{aligned}$$

をファディーブのアルゴリズムにより求める。ファディーブのアルゴリズムによりΓ_iとα_iは，

$$\begin{array}{ll} \Gamma_{n-1} = I & \alpha_{n-1} = -tr(A\Gamma_{n-1}) \\ \Gamma_{n-2} = A\Gamma_{n-1} + a_{n-1}I & \alpha_{n-2} = -tr(A\Gamma_{n-2})/2 \\ \Gamma_{n-3} = A\Gamma_{n-2} + a_{n-2}I & \alpha_{n-3} = -tr(A\Gamma_{n-3})/3 \\ \cdots\ \cdots\ \cdots & \cdots\ \cdots\ \cdots \\ \Gamma_i = A\Gamma_{i+1} + \alpha_{i+1}I & \alpha_i = -tr(A\Gamma_i)/(n-i) \\ \cdots\ \cdots\ \cdots & \cdots\ \cdots\ \cdots \\ \Gamma_0 = A\Gamma_1 + \alpha_1 I & \alpha_0 = -tr(A\Gamma_0)/n \end{array}$$

と与えられる。ただし， $tr(X)$ は，行列 X の対角成分の和を意味する。

第10章　整数と実数

MATX は，整数と実数を区別して処理する。1 は整数であり，1.0 は実数である。MATX では他の多くの言語と同様，整数の割算では切捨てが行なわれる。そして，実数は全て倍精度で計算される。

[宣言]

```
Integer a, b, c;
Real aa, bb, cc;
```

[例]

```
0, 3, -4                    // 整数
0., 0.0, 3., 3.0, -4. -4.0  // 実数
3E4, 3.E4., 3e4, 3.e4.      // 実数
```

以下の実数定数があらかじめ定義されている。

```
PI    =  3.14159265358979323846; // Circumf./Diameter
PID   =  ??????;                 // Process ID
EPS   =  pow(2.0, -54.0);        // Machine Epsilon
Inf   =  Infinity;               // Infinity
                                 // 正の無限大であり
                                 // 1.0/0.0 等の結果
NaN   =  NaN;                    // Not a Number
                                 // 0.0/0.0 等の結果
```

10.1　整数定数

整数型の定数である。数字の 0(ゼロ) 以外の数字で始まる数字の列は 10 進数を表す。0 または 0 で始まる数字の列は 8 進数を，数字のゼロとエックス 0x または 0X で始まる数字の列は 16 進数を表す。

[例]

17.1　連続系リカッティ方程式

連続時間の線形システムのリカッティ方程式の解法はハミルトン行列の固有値問題に帰着され [6]，そのプログラムの例を以下に示す。関数 eigvec() は固有値の実部の大きさの降順に固有ベクトルを並べて返すので，20–21 行目は，安定な固有値に対応する固有ベクトルを取り出す。以下は，リストの説明である。

- 1 行と 11 行: 関数の始まり。
- 3 行と 14 行: 局所変数の宣言。
- 6 行: キーボードから行列を読み込む。
- 7 行: 関数 CRiccati() を呼ぶ。
- 8 行: 解行列 P を表示する。
- 12 行: 関数の引数の宣言。
- 16 行: ハミルトン行列を作る [6]。
- 19 行: 固有ベクトルの計算。
- 22 行と 23 行: 行列 vec から安定な固有値に対する固有ベクトルを取り出す。vec(1:n,n+1:2*n) は vec の 1 行から n 行，n+1 列から 2n 列の部分行列，vec(n+1:2*n,n+1:2*n) は vec の n+1 行から 2n 行，n+1 列から 2n 列の部分行列である。
- 25 行と 26 行: 関数 isreal() は与えられた行列が実行列なら 1 を，そうでなければ 0 を返す。
- 27 行: 関数 Re() は行列の実部を返す。

`Real rem(i1, r2)`	割算の余り (i1 - r2*fix(i1/r2))
`Real rem(r1, i2)`	割算の余り (r1 - i2*fix(r1/i2))
`Real max(r1, r2)`	最大値
`Real min(r1, r2)`	最小値
`Real sin(r1);`	正弦関数
`Real sinh(r1);`	双曲線正弦関数
`Real asin(r1);`	逆正弦関数
`Real asinh(r1);`	逆双曲線正弦関数
`Real cos(r1);`	余弦関数
`Real cosh(r1);`	双曲線余弦関数
`Real acos(r1);`	逆余弦関数
`Real acosh(r1);`	逆双曲線余弦関数
`Real tan(r1);`	正接関数
`Real tanh(r1);`	双曲線正接関数
`Real atan(r1);`	逆正接関数
`Real atanh(r1);`	逆双曲線正接関数
`Real atan2(y, x);`	逆正接関数
`Real abs(r1);`	絶対値
`Real exp(r1);`	指数関数
`Real sqrt(r1);`	平方根
`Real log(r1);`	自然対数
`Real log10(r1);`	常用対数
`Real rand();`	0.0 と 1.0 の間の一様乱数
`Integer rand("seed")`	一様乱数の種
`Integer rand("seed", il)`	一様乱数の種の変更
`Real randn();`	平均 0，分散 1 の正規乱数
`Integer randn("seed")`	正規乱数の種
`Integer randn("seed", il)`	正規乱数の種の変更

makelist() を使って，任意個の成分を持つリストを定義することができる。この時，リストの成分は未定義 (UNDEF) である。

```
x = makelist(3);
```

16.2　リスト成分の参照と代入

リストのそれぞれの成分は，丸括弧（ と ）の間に指数とその成分のクラス名を書くことにより参照することができる。(指数は 1 から始まることに注意すること)

```
x = {sqrt(3.0), ((1+2)*4,:), [1 2] + [3 4]};
a = x(1, Real);                              // a = 1.732
b = x(2, Complex);                           // b = (12,0)

a2 = {sqrt(3.0), ((1+2)*4,:)}(1, Real);    // a2 = 1.732
b2 = {sqrt(3.0), ((1+2)*4,:)}(2, Complex); // b2 = (12,0)
```

丸括弧（ と ）の間に指数を入れて左辺に書くことによって，右辺の任意の値をリストの成分に代入することができる。(指数は 1 から始まる。) また，丸括弧（　と ）の間に成分の範囲を指定して，左辺に書くことによって，右辺のリストをその位置に代入することができる。右辺のリストの長さは，指定された範囲の長さ以上でなければならない。もし，右辺のリストの方が長い場合は，前の方の成分だけが左辺のリストに代入され，残りの成分は捨てられる。

```
x = makelist(4);
x(1) = (3,4);
a = x(1, Complex);  // a = (3,4)
x(2:3) = {sqrt(3.0), (5,6), [1 2] + [3 4], "Hello"};
// x = {(3,4), 1.732, (5,6), [4 6]}
```


第11章　複素数

[宣言]

```
Complex a, b, c;
```

[例]

```
(1, 2), (3.0, 4.0), (0, 1)
```

11.1　複素数の記述方法

一対の実数をコンマ，で区切って丸括弧（ と ）で囲むことによって複素数を記述することができる。実部または虚部だけを指定し，もう一方に * を指定して複素数を記述することもできる。成分が整数の時，自動的に実数に変換される。丸括弧（ と ）とコンマ “,” を合わせて複素オペレータと呼ぶ。複素オペレータを用いた複素値の記述方法については，複素値の章を参照。

```
c1 = (r1,r2)  // c1 = r1 + r2*i;  (r1 と r2 は，実数)
c1 = (r1, *)  // c1 = r1 + 0*i;   (r1 は，実数)
c1 = (*, r1)  // c1 = 0  + ri*i;  (r1 は，実数)
```

複素数の実部と虚部は，関数 Re() と Im() を用いて取り出すことができる。

```
r1 = Re((r1, r2));  // 実部 = Re(複素数)
r2 = Im((r1, r2));  // 虚部 = Im(複素数)
```

`Array diag2vec(A1);`	対角配列から縦ベクトル
`Array diag2vec(A1, k);`	対角配列から縦ベクトル (k だけ主対角より上)
`Array fliplr(A1);`	列を逆順に並べる
`Array flipud(A1);`	行を逆順に並べる
`List  sort(A1);`	成分を昇べきの順にソートする 複素配列の場合，絶対値でソート
`{A2, idx} = sort(A1);`	A2:　ソートした結果の配列 idx: ソートに使った指数 A2 = A1(idx)
`List size(A1);`	行数と列数のリスト {row, col} = size(A1);
`Array rand(m);`	m x m の一様乱数配列
`Array rand(m, n);`	m x n の一様乱数配列
`Array rand(A1);`	A1 と同じ大きさの一様乱数配列
`Array randn(A1);`	A1 と同じ大きさの正規乱数配列
`Array randn(m);`	m x m の正規乱数配列
`Array randn(m, n);`	m x n の正規乱数配列
`Array max(A1, A2);`	A1 と A2 の大きい方の成分 からなる配列（複素配列の場合， 絶対値によって比較する
`Array min(A1, A2);`	A1 と A2 の小さい方の成分 からなる配列（複素配列の場合， 絶対値によって比較する
`Array rem(A1, i1);`	A1 の各成分に対して rem() を 計算する
`Array rem(A1, r1);`	A1 の各成分に対して rem() を 計算する
`Array rem(A1, A2);`	A1 と A2 の成分毎に rem() を

第12章　文字列

任意個の文字の列を二重引用符 " と " で囲んで記述する。タブと改行はそれぞれ \t と \n で表す。+ 演算子で文字列を結合できる。

[宣言]

```
String a, b, c;
```

[例]

```
"FileName", "File\tName\n"
"Hello" + " " + "world"              // "Hello world"
```

12.1　文字列の成分の取り出し

`S(1)`	S の 1 番目の文字からなる文字列
`S(2:4)`	S の 2 番目から 4 番目の文字からなる文字列

12.2　文字列の成分の代入

`S(1) = "A"`	S の 1 番目に右辺の文字を代入
`S(2:4) = "AB"`	S の 2 番目から 4 番目に右辺の文字を代入

12.3　文字列の比較

関係演算子 ==, != を使って，2 つの文字列を比較することができ，結果は，1(真) または 0(偽) となる。

15.9　配列関数

配列関数の一覧を以下に示す。

`Integer  Rows(Array A1);`	行の数
`Integer  Cols(Array A1);`	列の数
`Integer  length(Array A1);`	行と列の大きい方の数
`Integer  isreal(Array A1);`	実配列の有無（真 = 1，偽 = 0）
`Integer  iscomplex(Array A1);`	複素配列の有無（真 = 1，偽 = 0）
`Integer  all(Array A1);`	全ての成分がゼロでない（真 = 1，偽 = 0）
`Integer  all(Matrix M1);`	全ての成分がゼロでない（真 = 1，偽 = 0）
`Array  all_col(Array A1);`	列毎に全ての成分がゼロでない（真 = 1，偽 = 0）の横ベクトル
`Array  all_col(Matrix M1);`	列毎に全ての成分がゼロでない（真 = 1，偽 = 0）の横ベクトル
`Array  all_row(Array A1);`	行毎に全ての成分がゼロでない（真 = 1，偽 = 0）の縦ベクトル
`Array  all_row(Matrix M1);`	行毎に全ての成分がゼロでない（真 = 1，偽 = 0）の縦ベクトル
`Integer  any(Array A1);`	ゼロでない成分がある（真 = 1，偽 = 0）
`Integer  any(Matrix M1);`	ゼロでない成分がある（真 = 1，偽 = 0）
`Array any_col(Array A1);`	列毎にゼロでない成分がある（真 = 1，偽 = 0）の横ベクトル
`Array any_col(Matrix M1);`	列毎にゼロでない成分がある（真 = 1，偽 = 0）の横ベクトル
`Array any_row(Array A1);`	行毎にゼロでない成分がある（真 = 1，偽 = 0）の縦ベクトル

であり，行列 a には [656667] が代入される。

12.7　文字列関数

文字列関数の一覧を以下に示す。書式付き出力関数 printf(), sprintf(), fprintf() を使って，整数，実数，文字列を書式を指定して出力することができる。書式文字列には C 言語の printf() のそれと同じものを使う。関数 fprintf() と fscanf() の第 1 引数は，fopen() でオープンしたファイルのファイルディスクリプタを用い，sscanf() の第 1 引数は，読み取りを行う文字列を指定する。

`Integer printf(String format, ...);`	画面表示（matx）
`Integer fprintf(Integer fd, String format, ...);`	ファイル出力 fd は fopen() の返り値
`Integer fprintf(String file, String format, ...);`	ファイル出力 file に追加される
`List  scanf(String format);`	キーボードから入力
`List  fscanf(Integer fd, String format);`	ファイル入力
`List  sscanf(String str, String format);`	文字列からの読み取
`String  sprintf(String format, ...);`	文字列変換
`Integer length(String str);`	文字列の長さ
`String  int2str(Integer c);`	ASCII コード c の文字を文字列に変換する。
`Matrix  abs(String str);`	ASCII コードからなる行列

```
List Ode45HybridAuto(Real t0, Real t1, Real dt, Matrix x,
                     void dx())
                     : eps=1.0E-6, dtsav=1.0E-3,
                       dtmin=EPS, dtmax=abs(t1-t0)/5
List Ode45HybridAuto(Real t0, Real t1, Real dt, Matrix x,
                     void dx(), void input())
                     : eps=1.0E-6, dtsav=1.0E-3
                       dtmin=EPS, dtmax=abs(t1-t0)/5
List Ode45HybridAuto(Real t0, Real t1, Real dt, Matrix x,
                  void dx(), void input(), Real eps)
                : dtsav=1.0E-3, dtmin=EPS, dtmax=abs(t1-t0)/5
List Ode45HybridAuto(Real t0, Real t1, Real dt, Matrix x,
                     void dx(), void input(), Real eps,
                     Real dtsav)
                    : dtmin=EPS, dtmax=abs(t1-t0)/5
List Ode45HybridAuto(Real t0, Real t1, Real dt, Matrix x,
                     void dx(), void input(), Real eps,
                     Real dtsav, Real dtmin)
                    : dtmax=abs(t1-t0)/5
List Ode45HybridAuto(Real t0, Real t1, Real dt, Matrix x,
                     void dx(), void input(), Real eps,
                     Real dtsav, Real dtmin, Real dtmax)
```

きざみを自動調節しながら RKF45 の公式にしたがって
指定された誤差内で常微分方程式の [t0,t1] 時刻の解を求める。
サンプリング間で外部入力は一定値をとる。
ただし，きざみの最小値と最大値は dtmin と dtmax であり，
dtmax < dtsav のとき，dtmax = dtsav に設定される。

```
{T, X, U} = Ode45HybridAuto();
```

T: 時間の系列
X: 状態ベクトルの時系列

13.4　多項式の評価

関数 eval() を用いて，多項式の変数にいろいろなクラスの値を代入した結果を求めることができる。

```
s = $;
P1 = 3*s^2 + 4*s + 5;
r1 = eval(P1, 2.0);    // 2.0 を代入
P2 = eval(P1, s + 1); // 多項式"s + 1"代入
A2 = eval(P1, [1:3]); // (1, 2, 3) を代入
M2 = eval(P1, M);      // 3*M^2 + 4*M + 5*I(M)
```

多項式の評価に伴うクラス変換の規則を以下に示す。

```
実数       = eval(実多項式, 実数)
複素数     = eval(実多項式, 複素数)
複素数     = eval(複素多項式, 実数)
複素数     = eval(複素多項式, 複素数)
実多項式   = eval(実多項式, 実多項式)
複素多項式 = eval(複素多項式, 実多項式)
複素多項式 = eval(実多項式, 複素多項式)
実行列     = eval(実多項式, 実行列)
複素行列   = eval(複素多項式, 実行列)
複素行列   = eval(実多項式, 複素行列)
複素行列   = eval(複素多項式, 複素行列)
実配列     = eval(実多項式, 実配列)
複素配列   = eval(複素多項式, 実配列)
複素配列   = eval(実多項式, 複素配列)
複素配列   = eval(複素多項式, 複素配列)
```

X: 状態ベクトルの時系列
U: 入出力ベクトルの時系列

```
List OdeHybrid(Real t0, Real t1, Real dt, Matrix x,
        void dx())               : h=4.0E-2, dtsav=1.0E-3
List OdeHybrid(Real t0, Real t1, Real dt, Matrix x,
        void dx(), void input()) : h=4.0E-2, dtsav=1.0E-3
List OdeHybrid(Real t0, Real t1, Real dt, Matrix x,
        void dx(), void input(), Real h) : dtsav=1.0E-3
List OdeHybrid(Real t0, Real t1, Real dt, Matrix x, void dx(),
        void input(), Real h, Real dtsav)
```

4 次のルンゲ- クッタの公式にしたがって，指定されたきざみ幅 h で，

常微分方程式の [t0,t1] 時刻の解を求める。ただし，
サンプリング間で外部入力は一定値をとる。
また，きざみ幅 h が省略されて，dt < 4.0E-2 のとき，
h = dt である．
{T, X, U} = OdeHybrid();
T: 時間の系列
X: 状態ベクトルの時系列
U: 入出力ベクトルの時系列

```
List OdeHybridAuto(Real t0, Real t1, Real dt, Matrix x,
                   void dx())
                  : eps=1.0E-6, dtsav=1.0E-3,
                    dtmin=EPS, dtmax=abs(t1-t0)/5
List OdeHybridAuto(Real t0, Real t1, Real dt, Matrix x,
                   void dx(), void input())
                  : eps=1.0E-6, dtsav=1.0E-3,
                    dtmin=EPS, dtmax=abs(t1-t0)/5
```


第14章　有理多項式

実数と複素数を係数とする 1 変数の有理多項式の数式処理を行なうことができる。

[宣言]

```
Rational   a, b, c;      // 実有理多項式
CoRational a2, b2, c2;   // 複素有理多項式
```

[例]

```
(s + 2)/(2*s + 3),  (1,2)/((3,4)*s + (5,6))
```

14.1　有理多項式の記述方法

有理多項式は多項式の割算として記述する。出力される有理多項式の変数名は，常に s である。

```
s = $;
R1 = (s + 2)/(2*s + 3);         // 1 次の実有理多項式
CR1 = (1,2)/((3,4)*s + (5,6)); // 1 次の複素有理多項式
```

14.2　複素有理多項式

複素有理多項式の入力方法には，複素数を係数として記述する方法と一対の実有理多項式をコンマ “,” で区切って丸括弧（ と ）で囲む方法がある。実部または虚部だけを指定し，もう一方に * を指定して複素有理多項式を記述することもできる。

```
                          縦ベクトルから対角行列
                          (k だけ主対角より上)
Matrix  diag2vec(M1);     対角行列から縦ベクトル
Matrix  diag2vec(M1, k);
                          対角行列から縦ベクトル
                          (k だけ主対角より上)
Matrix  fliplr(M1);       列を逆順に並べる
Matrix  flipud(M1);       行を逆順に並べる
Matrix  shiftLeft(M1, k);
                          全ての列を左に k シフトし，
                          右から 0 を代入する
                          k 省略すると，シフト数は 1 となる
Matrix  shiftRight(M1, k);
                          全ての列を右に k シフトし，
                          左から 0 を代入する
                          k 省略すると，シフト数は 1 となる
Matrix  shiftUp(M1, k);
                          全ての列を上に k シフトし，
                          下から 0 を代入する
                          k 省略すると，シフト数は 1 となる
Matrix  shiftDown(M1, k);
                          全ての列を下に k シフトし，
                          上から 0 を代入する
                          k 省略すると，シフト数は 1 となる
Matrix  rotateLeft(M1, k);
                          全ての列を左に k 回転する
                          k 省略すると，シフト数は 1 となる
Matrix  rotateRight(M1, k);
                          全ての列を右に k 回転する
                          k 省略すると，シフト数は 1 となる
```


14.7　有理多項式の係数のシフト

有理多項式の係数を高次方向と低次方向にシフトするには，関数 higher() と lower() をそれぞれ用いる。第 2 引数にシフトする回数を指定するが，回数を省略すると 1 回のシフトを意味する。

```
s = $;
R1 = (s + 2)/(2*s + 3);
R2 = higher(R1);          // 係数を高次方向に 1 回シフト
R2 = higher(R1,2);        // 係数を高次方向に 2 回シフト
R3 = lower(R2);           // 係数を低次方向に 1 回シフト
R3 = lower(R2,2);         // 係数を低次方向に 2 回シフト
```

14.8　有理多項式の比較

関係演算子 ==, != を使って，2 つの有理多項式を比較することができ，結果は 1(真) または 0(偽) となる。

表 記	意 味
R1 == R2	有理多項式 R1 と R2 は等しい
R1 != R2	有理多項式 R1 と R2 は等しくない

14.9　有理多項式関数

有理多項式関数の一覧を以下に示す。

```
Rational R1;
Integer i1;
Real tol;

Matrix  zeros(R1);          有理多項式の零点
Matrix  poles(R1);          有理多項式の極
Integer isreal(R1);         実有理多項式の有無
                            (真 = 1, 偽 = 0)
Integer iscomplex(R1);      複素有理多項式の有無
```

```
Real norm(M1);          norm(M1, 2)

Complex max(Matrix M1);       絶対値が最大の成分
Complex min(Matrix M1);       絶対値が最小の成分
List maximum(Matrix M1);      絶対値が最大の成分とその位置
                              {c1, i, j} = maximum(M1);
                              c1 == M1(i,j)
List minimum(Matrix M1);      絶対値が最小の成分とその位置
                              {c1, i, j} = mininum(M1);
                              c1 == M1(i,j)
Complex det(CMatrix M1);      複素行列の行列式
Complex trace(CMatrix M1);    複素行列のトレース

Polynomial det(PMatrix M1);      多項式行列の行列式
Polynomial trace(PMatrix M1);    多項式行列のトレース
Polynomial frobnorm(Matrix M1); 多項式行列のフロベニウスノルム

Rational  det(RMatrix M1);       有理多項式行列の行列式
Rational  trace(RMatrix M1);     有理多項式行列のトレース
Rational  frobnorm(Matrix M1); 有理多項式行列のフロベニウスノルム

Real sum(Matrix M1);           全ての成分の和
Complex sum(CoMatrix M1);      全ての成分の和
Polynomail sum(PoMatrix M1);   全ての成分の和
Rational sum(RaMatrix M1);     全ての成分の和
Real prod(Matrix M1);          全ての成分の積
Complex prod(CoMatrix M1);     全ての成分の積
Polynomial prod(PoMatrix M1); 全ての成分の積
Rational prod(RaMatrix M1);    全ての成分の積
```


1. 成分の整数は全て実数に変換される。
2. 成分に複素数又は複素行列が存在する時，
   全ての成分は複素数に変換される。
3. 成分に多項式又は多項式行列が存在する時，
   全ての成分は多項式に変換される。
4. 成分に有理多項式または有理多項式行列が存在する時，
   全ての成分は有理多項式に変換される。

例外　成分が全て文字列の時，それらを結合した
　　文字列に変換される。
　　(Matlab との互換性のための規則であり，
　　一般ユ- ザは，使用すべきでない。)

ただし，成分中に式がある場合，式の解釈が行なわれた後，自動変換が行なわれる。次のような間違いを犯す可能性があるので，注意すること。

```
M1 = [1/2, 3/4, 4/5, 5/6];
```

この結果は，ユーザの予想と反して，1×4 の零行列となる。望みの結果を得るためには，

```
M1 = [1./2., 3./4., 4./5., 5./6.];
```

としなければならない。

15.1.2　行ベクトル

成分をスペース，タブ，改行，コンマ “,” のいずれかで区切って書き並べ，四角ブラケット [と] で囲む。

表 記	意 味
配列 1 < 配列 2	各成分の比較結果 (0|1) の配列
配列 1 > 配列 2	各成分の比較結果 (0|1) の配列
配列 1 <= 配列 2	各成分の比較結果 (0|1) の配列
配列 1 >= 配列 2	各成分の比較結果 (0|1) の配列
配列 1 == 配列 2	各成分の比較結果 (0|1) の配列
配列 1 != 配列 2	各成分の比較結果 (0|1) の配列
配列 <（整数|実数）	各成分の比較結果 (0|1) の配列
配列 >（整数|実数）	各成分の比較結果 (0|1) の配列
配列 <=（整数|実数）	各成分の比較結果 (0|1) の配列
配列 >=（整数|実数）	各成分の比較結果 (0|1) の配列
配列 ==（整数|実数）	各成分の比較結果 (0|1) の配列
配列 !=（整数|実数）	各成分の比較結果 (0|1) の配列
（整数|実数） < 配列	各成分の比較結果 (0|1) の配列
（整数|実数） > 配列	各成分の比較結果 (0|1) の配列
（整数|実数） <= 配列	各成分の比較結果 (0|1) の配列
（整数|実数） >= 配列	各成分の比較結果 (0|1) の配列
（整数|実数） == 配列	各成分の比較結果 (0|1) の配列
（整数|実数） != 配列	各成分の比較結果 (0|1) の配列


```
// from から to まで by づつ増加する数列を成分とする横配列
A1 = [from:by:to];
// from から to まで 1 づつ増加する数列を成分とする横配列
A1 = [from:to];
```

```
A1 = [0.1:0.1:0.5];  // M1 = [0.1 0.2 0.3 0.4 0.5]
A2 = [0.5:-0.1:0.1]; // M2 = [0.5 0.4 0.3 0.2 0.1]
A3 = [4:12];         // M3 = [4 5 6 7 8 9 10 11 12]
```

15.2.5　対角行列とブロック対角行列

対角行列及びブロック対角行列を定義する時，全ての成分のクラスは一致しなければならない。整数，実数，多項式等のスカラーと行列を混在させてブロック対角行列を記述するには，スカラーを 1 x 1 の行列として記述する。

```
M1 = diag(1.0, 2.0, 3.0);         // 実対角行列
CM1 = diag((1,2), (3,4), (5,6)); // 複素対角行列
M4 = diag(M1, M2, M3);            // ブロック対角行列
M3 = diag(M1, M2, [1]);           // ブロック対角行列
```

15.2.6　複素行列

複素行列の記述方法には，複素数を成分として記述する方法と一対の実行列をコンマ “,” で区切って丸括弧（ と ）で囲む方法がある。実部または虚部だけを指定し，もう一方に * を指定して複素行列を記述することもできる。

めには，リストを利用しなければならない。ピース行列は，行列の配列の大きさの単位を指定するために使われる。たとえば，

```
A = [[1 2][3 4]];
AA = Z(2, 3, A);
```

によって，行列 A を縦に 2 個横に 3 個並べた 4 x 6 の零行列ができ，

```
AA(0, 0, A) = [[1 2][3 4]];
AA(1, 2, A) = [[5 6][7 8]];
```

によって，

$$\mathtt{AA} = \begin{bmatrix} 1 & 2 & \mathtt{Z(2)} & \mathtt{Z(2)} \\ 3 & 4 & & \\ \mathtt{Z(2)} & \mathtt{Z(2)} & 5 & 6 \\ & & 7 & 8 \end{bmatrix}$$

が得られる。ここで，A 行列は AA 行列のピース行列 (**Piece Matrix**) と呼ばれる。別の行列 B = [1] を AA 行列のピース行列として使って，

```
B = [1];
AA = Z(2, 3, A);
AA(0, 0, B) = [[1 2][3 4]];
AA(1, 2, B) = [[5 6][7 8]];
```

とすると，

$$\mathtt{AA} = \begin{bmatrix} 1 & 2 & 0 & 0 & 0 & 0 \\ 3 & 4 & 5 & 6 & 0 & 0 \\ 0 & 0 & 7 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

となる。

の結果，

$$\mathtt{B} = \begin{bmatrix} \mathtt{eval(s+1,\ A)} & \mathtt{eval(s+2,\ A)} \\ \mathtt{eval(s+3,\ A)} & \mathtt{eval(s+4,\ A)} \end{bmatrix} = \begin{bmatrix} 2 & 2 & 3 & 2 \\ 3 & 5 & 3 & 6 \\ 4 & 2 & 5 & 2 \\ 3 & 7 & 3 & 8 \end{bmatrix}$$

となるが，配列を用いると

```
s = $;
RM1 = [[(s+1) (s+2)]
       [(s+3) (s+4)]];

A = [[1 2][3 4]];

B = eval(RM1, Array(A));
```

の結果，

$$\mathtt{B} = \begin{bmatrix} \mathtt{eval(RM1,\ 1)} & \mathtt{eval(RM1,\ 2)} \\ \mathtt{eval(RM1,\ 3)} & \mathtt{eval(RM1,\ 4)} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 3 & 4 \\ 4 & 5 & 5 & 6 \\ 4 & 5 & 5 & 6 \\ 6 & 7 & 7 & 8 \end{bmatrix}$$

となる。多項式行列の評価に伴うクラス変換の規則を以下に示す。

```
M1(0,0,M2) = M3;        M2 を縦横に並べた時，1 行 1 列に代入する
M1(1,2,M2) = M3;        M2 を縦横に並べた時，2 行 3 列に代入する
M(1,:,M2) = M3;         M2 を縦横に並べた時，2 行に代入する
M(:,2,M2) = M3;         M2 を縦横に並べた時，3 列に代入する
M1(1:2,1:2,M2) = M3;    M2 を縦横に並べた時，2-3 行,2-3 列に代入する
M1(1,2:3,M2) = M3;      M2 を縦横に並べた時，2 行,3-4 列に代入する
M1(:,2:3,M2) = M3;      M2 を縦横に並べた時，3-4 列に代入する
M1(1,2:,M2) = M3;       M2 を縦横に並べた時，2 行,3 列以降に代入する
M1(2:3,1,M2) = M3;      M2 を縦横に並べた時，3-4 行,2 列に代入する
M1(2:3,:,M2) = M3;      M2 を縦横に並べた時，3-4 行に代入する
M1(2:,1,M2) = M3;       M2 を縦横に並べた時，3 行以降,2 列に代入する
```

15.5.7　行列成分の削除

代入式の左辺に，削除する成分を指定した行列を書き，右辺に 0 x 0 の行列を用いれば，左辺の行列から行単位あるいは列単位で成分を削除できる。

```
M1(1,:)   = [];   M1 の 1 行を削除する
M1(2:3,:) = [];   M1 の 2 ~ 3 行を削除する
M1(:,1)   = [];   M1 の 1 列を削除する
M1(:,2:3) = [];   M1 の 2 ~ 3 列を削除する
M1(2:3)   = [];   M1 の (1,2),(1,3) 成分を削除する
                  (1,i) 成分の指定する時，1 を省略できる。
```

15.5.8　行列の大きさの変更

行列の大きさは，関数 reshape() を使って変更することができる。だだし，変更前の行列の成分の数と変更後の行列の成分の数は，等しくなければならない。行列の成分は，左から右へ，上から下へ，並べられる。

$$\left(\ 2\right)\ \frac{s+5}{s+6}\quad\frac{s+7}{s+8}$$

となる。また，配列オペレータを使うことによって，有理多項式行列を簡単に記述することができ，

```
s = $;
X = [[1 3][5 7]];  Y = [[2 4][6 8]];
B = (ONE(2)*s + X) ./ (ONE(2)*s + Y);
```

によって，有理多項式行列 B は，A と一致する。ただし，記号 ./ は，行列の成分毎の割算を行なう配列オペレータを意味し，ONE(2) は，全ての成分が 1 である 2 x 2 の行列である。

複素有理多項式行列の入力方法には，複素有理多項式を成分として記述する方法と一対の実有理多項式行列をコンマ “,” で区切って丸括弧（ と ）で囲む方法がある。実部または虚部だけを指定し，もう一方に * を指定して複素有理多項式行列を記述することもできる。

```
CRM1 = (RM1,RM2); // CRM1 = RM1 + RM2 i,
                  // RM1 と RM2 は，実有理多項式行列
CRM1 = (RM1,*);   // CRM1 = RM1 + Z(RM1) i,
                  // RM1 は，実有理多項式行列
CRM1 = (*,RM1);   // CRM1 = Z(RM1) + RM1 i,
                  // RM1 は，実有理多項式行列
```

実部有理多項式行列と虚部有理多項式行列は，関数 Re() と Im() を用いて取り出すことができる。

```
RM1 = Re((RM1, RM2));  // 実部有理多項式行列
RM2 = Im((RM1, RM2));  // 虚部有理多項式行列
```

[例] 2 個の実有理多項式行列 RM1 と RM2 から複素有理多項式行列 CRM1 を作る。

M1(0,1,M2)	M2 を縦横に並べた時，1 行 2 列のブロック成分
M1(1,2,M2)	M2 を縦横に並べた時，2 行 3 列のブロック成分
M1(1,:,M2)	M2 を縦横に並べた時，2 行のブロック成分
M1(:,1,M2)	M2 を縦横に並べた時，2 列のブロック成分
M1(1:2,2:3,M2)	M2 を縦横に並べた時，2-3 行，3-4 列のブロック成分
M1(2:3,4,M2)	M2 を縦横に並べた時，3-4 行，5 列のブロック成分
M1(2:3,:,M2)	M2 を縦横に並べた時，3-4 行のブロック成分
M1(2:,4,M2)	M2 を縦横に並べた時，3 行以降，5 列のブロック成分
M1(4,2:3,M2)	M2 を縦横に並べた時，5 行，3-4 列のブロック成分
M1(:,2:3,M2)	M2 を縦横に並べた時，3-4 列のブロック成分
M1(4,2:,M2)	M2 を縦横に並べた時，5 行，3 列以降のブロック成分

15.5.3　式行列の成分の取り出し

行列の式を四角ブラケット [と] で囲むことによって変数行列と同様に成分を取り出すことができる。

[M1+M2](1,1)	M1+M2 の (1,1) 成分
[M1+M2](3:5)	M1+M2 の 3-5 番目成分からなるベクトル
[M1+M2](3:)	M1+M2 の 3 番目以降の成分からなるベクトル
[M1+M2](1,:)	M1+M2 の第 1 行ベクトル
[M1+M2](1:2,2:3)	M1+M2 の 1-2 行，2-3 列のブロック成分
[M1+M2](2:3,4)	M1+M2 の 2-3 行，4 列成分よりなる列ベクトル
[M1+M2](2:,4)	M1+M2 の 2 行以降，4 列成分よりなる列ベクトル

15.5.4　0-1 配列を用いた成分の操作

比較演算の結果生成される 0-1 配列を用いれば小行列を作成できる。m x 1 の縦ベクトル (配列)A から負の要素を取り去る例を次に示す。

実配列	= eval(実有理多項式行列，実配列)
複素配列	= eval(実有理多項式行列，複素配列)
複素配列	= eval(複素有理多項式行列，実配列)
複素配列	= eval(複素有理多項式行列，複素配列)
実有理多項式配列	= eval(実有理多項式行列，実多項式配列)
実有理多項式配列	= eval(実有理多項式行列，実有理多項式配列)
複素有理多項式配列	= eval(実有理多項式行列，複素多項式配列)
複素有理多項式配列	= eval(複素有理多項式行列，実多項式配列)
複素有理多項式配列	= eval(実有理多項式行列，複素有理多項式配列)
複素有理多項式配列	= eval(複素有理多項式行列，実有理多項式配列)
複素有理多項式配列	= eval(複素有理多項式行列，複素有理多項式配列)

15.4.3　有理多項式行列の微分

有理多項式行列の微分は，関数 derivative() を用いる。有理多項式行列の積分は，今のところ用意されていない。第 2 引数に求める導関数の階数を指定するが，階数を省略すると 1 階の導関数を求める。

```
CRM2 = derivative(CRM1);    // CRM1 の 1 階導関数
CRM3 = derivative(CRM1,2);  // CRM1 の 2 階導関数
```

15.4.4　有理多項式行列の各要素の係数のシフト

有理多項式行列の各要素の係数を高次方向と低次方向にシフトするには，関数
higher() と lower() をそれぞれ用いる。第 2 引数にシフトする回数を指定するが，回数を省略すると 1 回のシフトを意味する。

実行列　　= eval(実有理多項式行列，実数)
複素行列　　= eval(実有理多項式行列，複素数)
複素行列　　= eval(複素有理多項式行列，実数)
複素行列　　= eval(複素有理多項式行列，複素数)
実有理多項式行列　= eval(実有理多項式行列，実多項式)
実有理多項式行列　= eval(実有理多項式行列，実有理多項式)
複素有理多項式行列 = eval(実有理多項式行列，複素多項式)
複素有理多項式行列 = eval(複素有理多項式行列，実多項式)
複素有理多項式行列 = eval(実有理多項式行列，複素有理多項式)
複素有理多項式行列 = eval(複素有理多項式行列，実有理多項式)
複素有理多項式行列 = eval(複素有理多項式行列，複素有理多項式)

実行列　　= eval(実有理多項式行列，実行列)
複素行列　　= eval(実有理多項式行列，複素行列)
複素行列　　= eval(複素有理多項式行列，実行列)
複素行列　　= eval(複素有理多項式行列，複素行列)
実有理多項式行列　= eval(実有理多項式行列，実多項式行列)
実有理多項式行列　= eval(実有理多項式行列，実有理多項式行列)
複素有理多項式行列 = eval(実有理多項式行列，複素多項式行列)
複素有理多項式行列 = eval(複素有理多項式行列，実多項式行列)
複素有理多項式行列 = eval(実有理多項式行列，複素有理多項式行列)
複素有理多項式行列 = eval(複素有理多項式行列，実有理多項式行列)
複素有理多項式行列 = eval(複素有理多項式行列，複素有理多項式行列)


```
A = Array([-1, 1, -2, 2]);
L = A > 0;          // L = [0 1 0 1]
B = A(L);           // B = [1 2]
```

ここで，L は比較演算の結果作られる 0 と 1 からなる m x 1 の配列である。次のように，L を用いずに直接正の成分を取り出すこともできる。

```
A = Array([-1, 1, -2, 2]);
B = A(A > 0);       // B = [1 2]
```

関数 find() を使って指数を求めてから，正の成分を取り出してもよい。

```
A = Array([-1, 1, -2, 2]);
idx = find(A > 0);  // idx = [2 4]
B = A(idx);         // B = [1 2]
```

次に，A の負のようその符合を反転する例を示す。

```
A = Array([-1, 1, -2, 2]);
L = A < 0;      // L = [1 0 1 0]
B = -A(L);      // B = [1 2]
A(L) = B;       // A = [1 1 2 2]
```

次のように，L を用いないで直接負の成分の符合を反転することのできる。

```
A = Array([-1, 1, -2, 2]);
A(A < 0) = -A(A < 0);      // A = [1 1 2 2]
```

関数 find() を使って指数を求めてから，負の成分の符合を反転しても良い。

```
CPM2 = higher(CPM1);    // CPM1 の高次方向への 1 回シフト
CPM3 = higher(CPM1,2);  // CPM1 の高次方向への 2 回シフト
CPM4 = lower(CPM2);     // CPM2 の低次方向への 1 回シフト
CPM5 = lower(CPM3,2);   // CPM3 の低次方向への 2 回シフト
```

15.4　有理多項式行列

15.4.1　有理多項式行列の記述方法

有理多項式行列は，有理多項式を成分とする行列である。有理多項式行列の操作は実行列や複素行列といった数値行列と同じである。

```
s = $;

RM1 = [[(s+1)/(s+2)  (s+3)/(s+4)]
       [(s+5)/(s+6)  (s+7)/(s+8)]];

RM2 = diag((s+1)/(s+2), (s+3)/(s+4), (s+5)/(s+6));
```

有理多項式行列を記述するには，陽に有理多項式を行列の成分として用いる。例えば，

```
s = $;
A = [[(s+1)/(s+2) , (s+3)/(s+4)]
     [(s+5)/(s+6) , (s+7)/(s+8)]];
```

の結果，

```
=== [A] : (  2,  2) ===

      [     (  1)     ]  [     (  2)     ]
            s + 1              s + 3
(  1) ----------------   ----------------
            s + 2              s + 4
```



```
M1 = reshape([1 2 3 4], 2, 2); // M1 = [[1 2][3 4]]
```

15.5.9　指数を用いた行列成分の操作

指数は，整数を 1 次元的に並べたものであり，クラス変換関数 Index() を使って行列や配列から作ることができる。指数は，ある行列から任意の成分を取り出して，任意の順に並べた部分行列を作ったり，ある行列に任意の順で成分を代入する時に，使われる。行列の成分の取り出しや代入のために使われるコロン : で区切った整数の並びも指数と同様な働きをする。

例えば，X が n 次元ベクトルで V が n 次元指数とすると，X(V) は，

```
[X(V(1)), X(V(2)), X(V(3)), ..., X(V(n))]
```

となる。また，代入式において両辺で指数を使うと，もっと複雑なことができる。例えば，

```
A(:, Index([3 5 10])) = B(:, 1:3)
```

によって，行列 A の 3 列，5 列，10 列に行列 B の最初の 3 列が代入される。もっと一般的に，もし v と w が指数なら，

```
A(v, w)
```

は，v と w の成分を行番号と列番号としてもつ A の成分からなる部分行列を意味する。したがって，

```
A(:, n:-1:1)
```

実部多項式行列と虚部多項式行列は，関数 Re() と Im() を用いて取り出すことができる。

```
PM1 = Re((PM1, PM2));  // 実部多項式行列
PM2 = Im((PM1, PM2));  // 虚部多項式行列
```

[例] 2 個の実多項式行列 PM1 と PM2 から複素多項式行列 CPM1 を作る。

```
  s = $;

 PM1 = [[s + 1   s + 2]     PM2 = [[s + 5   s + 6]
        [s + 3   s + 4]];          [s + 7   s + 8]];

  CPM1 = (PM1, PM2);
  CPM2 = [[(1,1)*s+(1,5) (1,1)*s+(2,6)]
          [(1,1)*s+(3,7) (1,1)*s+(4,8)]];
```

15.3.2　多項式行列の評価

関数 eval() を用いて，多項式行列の変数にいろいろなクラスの値を代入した結果を求めることができる。配列は，多項式行列の各成分の変数に代入されるので，評価結果はブロック行列になる。例えば，変数に行列を代入して

```
  s = $;
  RM1 = [[(s+1) (s+2)]
         [(s+3) (s+4)]];

  A = [[1 2][3 4]];

  B = eval(RM1, A);
```


15.6　行列の比較と配列の比較

行列の正定性や 2 つの行列の比較は，関係演算子 >, >=, <, <= を使って調べることができる。配列を関係演算子を用いて比較した場合，各成分毎の比較が行なわれ，結果が (0 または 1) を成分とする配列として返される。この結果は，関数 all()，all_col()，all_row()，any()，any_col()，any_row() の引数として利用できる。

表 記	意 味
行列 < 0	maxeigval(行列) < 0
行列 > 0	mineigval(行列) > 0
行列 <= 0	maxeigval(行列) <= 0
行列 >= 0	mineigval(行列) >= 0
行列 == 0	行列は零行列である
行列 != 0	行列は零行列でない
0 > 行列	maxeigval(行列) < 0
0 < 行列	mineigval(行列) > 0
0 >= 行列	maxeigval(行列) <= 0
0 <= 行列	mineigval(行列) >= 0
0 == 行列	行列は零行列である
0 != 行列	行列は零行列でない
行列 1 < 行列 2	maxeigval(行列 1 - 行列 2) < 0
行列 1 > 行列 2	mineigval(行列 1 - 行列 2) > 0
行列 1 <= 行列 2	maxeigval(行列 1 - 行列 2) <= 0
行列 1 >= 行列 2	mineigval(行列 1 - 行列 2) >= 0
行列 1 == 行列 2	行列 1 は行列 2 と等しい
行列 1 != 行列 2	行列 1 は行列 2 と等しくない

maxeigval(行列) は固有値の実部の最大値を，mineigval(行列) は固有値の実部の最小値を意味する。

```
M1 = I(3, 4, P);  3x4 の実多項式行列,        P = 実多項式
M1 = I(3, 4, CP); 3x4 の複素多項式行列,      CP = 複素多項式
M1 = I(3, 4, R);  3x4 の実有理多項式行列,      R = 実有理多項式
M1 = I(3, 4, CR); 3x4 の複素有理多項式行列,   CR = 複素有理多項式
M2 = I(M1);       M1 と同じ大きさで同じクラスの行列
M2 = I(3, M1);    M1 を横 3 個並べた大きさで同じクラスの行列
M2 = I(2, 3, M1); M1 を縦に 2 個, 横に 3 個並べた大きさで
                  M1 と同じクラスの行列
```

15.2.3　1 で満たされた行列

多項式行列の場合，成分の全てが定数項が 1 である多項式になる。有理多項式行列の場合，成分の全てが分子多項式と分母多項式の定数項が 1 である有理多項式になる。ピース行列を大きさの単位として使い，ピース行列の整数倍の大きな 1 で満たされた行列を簡単に作ることができる。

```
M1 = ONE(3);         3x3 の実行列
M1 = ONE(3, c);      3x3 の複素行列,          c = 複素数
M1 = ONE(3, P);      3x3 の実多項式行列,      P = 実多項式
M1 = ONE(3, CP);     3x3 の複素多項式行列,    CP = 複素多項式
M1 = ONE(3, R);      3x3 の実有理多項式行列,    R = 実有理多項式
M1 = ONE(3, CR);     3x3 の複素有理多項式行列, CR = 複素有理多項式
M1 = ONE(3, 4);      3x4 の実行列,
M1 = ONE(3, 4, c);   3x4 の複素行列,          c = 複素数
M1 = ONE(3, 4, P);   3x4 の実多項式行列,      P = 実多項式
M1 = ONE(3, 4, CP);  3x4 の複素多項式行列,    CP = 複素多項式
M1 = ONE(3, 4, R);   3x4 の実有理多項式行列,    R = 実有理多項式
M1 = ONE(3, 4, CR);  3x4 の複素有理多項式行列, CR = 複素有理多項式
M2 = ONE(M1);        M1 と同じ大きさで同じクラスの行列
M2 = ONE(3, M1);     M1 を横に 3 個並べた大きさで同じクラスの行列
M2 = ONE(2, 3, M1);  M1 を縦に 2 個, 横に 3 個並べた大きさで
                     M1 と同じクラスの行列
```

15.2.4　等差数列配列

等差数列配列は行列ではなく配列として処理される。

表 記	意 味
配列 && 配列	論理演算の結果 (0|1) の配列
行列 .&& 行列	論理演算の結果 (0|1) の配列
行列 .&& 配列	論理演算の結果 (0|1) の配列
配列 .&& 行列	論理演算の結果 (0|1) の配列
配列 || 配列	論理演算の結果 (0|1) の配列
行列 .|| 行列	論理演算の結果 (0|1) の配列
行列 .|| 配列	論理演算の結果 (0|1) の配列
配列 .|| 行列	論理演算の結果 (0|1) の配列
! 配列	論理演算の結果 (0|1) の配列
.! 行列	論理演算の結果 (0|1) の配列

次の式は-A の成分が正である (0 より大きい) かどうか調べる意図で書かれているが，< と- の間にスペースが無いので，まとめて <- として解釈され，文法エラーになる。< と- の間にスペースを入れると意図通りに解釈される。

```
print 0<-A;
```

15.7　行列とスカラの演算

演算子 .+や.- によって定義される (行列—配列—指数) とスカラ (整数—実数—複素数など) の演算では、スカラはその値で満たされた (行列—配列—指数) と同じ大きさの (行列—配列—指数) に自動的に変換された後、通常の演算が行われる。

(行列|配列|指数) .+ (整数|実数|複素数|多項式|有理多項式)
(行列|配列|指数) .- (整数|実数|複素数|多項式|有理多項式)
(整数|実数|複素数|多項式|有理多項式) .+ (行列|配列|指数)
(整数|実数|複素数|多項式|有理多項式) .- (行列|配列|指数)

15.1　行列の記述方法

小さな行列の最も簡単な記述方法は，行列の成分をスペース，タブ，改行，コンマ “,” のいずれかで区切って書き並べ，四角ブラケット [と] で囲むことである。ただし，マイナス単項演算子 $(-)$ を含む成分は，コンマ “,” で区切るか，括弧で囲まなければならない。A = [] は，0×0 の行列を意味する。四角ブラケット [と] は，行の始まりと終りを示すためにも使われる。各成分の間には，任意個のスペースと改行を挿入することができるので，プログラムが読み易くなるように記述することができる。一方，大きな行列は，小さな行列を成分として構成する。

例えば，3×3 の実数行列は，次のように

```
A = [[1 2 3][4 5 6][7 8 9]];
```

あるいは，もっと自然な様式

```
A = [[1 2 3]
     [4 5 6]
     [7 8 9]];
```

で記述でき，結果は

```
=== [A] : (  3,  3) ===
            (  1)          (  2)          (  3)
(  1)  1.000000E+00  2.000000E+00  3.000000E+00
(  2)  4.000000E+00  5.000000E+00  6.000000E+00
(  3)  7.000000E+00  8.000000E+00  9.000000E+00
```

となる。

15.1.1　行列成分の変換規則

行列の成分に異なるクラスの値が存在する時，以下の規則に従って自動変換が行なわれる。


```
Real mean(Matrix M1);          全ての成分の平均値
Complex mean(CoMatrix M1);     全ての成分の平均値
Polynomial mean(PoMatrix M1); 全ての成分の平均値
Ratioanal mean(RaMatrix M1);   全ての成分の平均値

Real std(Matrix M1);           全ての成分の標準偏差
Complex std(CoMatrix M1);      全ての成分の標準偏差
Polynomial std(PoMatrix M1);   全ての成分の標準偏差
Rational std(RaMatrix M1);     全ての成分の標準偏差

Matrix cumsum(Matrix M1);      全ての成分の累積和
Matrix cumprod(Matrix M1);     全ての成分の累積積


Matrix M1;

Matrix sum_col(M1);       (縦・横) ベクトルの成分の和
                          列毎の和からなる横ベクトル
Matrix sum_row(M1);       (縦・横) ベクトルの成分の和
                          行毎の和からなる縦ベクトル
Matrix prod_col(M1);      (縦・横) ベクトルの成分の積
                          列毎の積からなる横ベクトル
Matrix prod_row(M1);      (縦・横) ベクトルの成分の積
                          行毎の積からなる縦ベクトル
Matrix mean_col(M1);      (縦・横) ベクトルの成分の平均値
                          列毎の平均値からなる横ベクトル
Matrix mean_row(M1);      (縦・横) ベクトルの成分の平均値
                          行毎の平均値からなる縦ベクトル
Matrix std_col(M1);       (縦・横) ベクトルの成分の標準偏差
                          列毎の標準偏差からなる横ベクトル
Matrix std_row(M1);       (縦・横) ベクトルの成分の標準偏差
```

実数	= eval(実有理多項式，実数)
複素数	= eval(実有理多項式，複素数)
複素数	= eval(複素有理多項式，実数)
複素数	= eval(複素有理多項式，複素数)
実有理多項式	= eval(実有理多項式，実多項式)
実有理多項式	= eval(実有理多項式，実有理多項式)
複素有理多項式	= eval(実有理多項式，複素多項式)
複素有理多項式	= eval(複素有理多項式，実多項式)
複素有理多項式	= eval(実有理多項式，複素有理多項式)
複素有理多項式	= eval(複素有理多項式，実有理多項式)
複素有理多項式	= eval(複素有理多項式，複素有理多項式)
実行列	= eval(実有理多項式，実行列)
複素行列	= eval(複素有理多項式，実行列)
複素行列	= eval(実有理多項式，複素行列)
複素行列	= eval(複素有理多項式，複素行列)
実配列	= eval(実有理多項式，実配列)
複素配列	= eval(複素有理多項式，実配列)
複素配列	= eval(実有理多項式，複素配列)
複素配列	= eval(複素有理多項式，複素配列)

14.6　有理多項式の微分

有理多項式の微分は，関数 derivative() を用いる。有理多項式の積分は，今のところ実現されていない。第 2 引数に求める導関数の階数を指定するが，階数を省略すると 1 階の導関数を求める。

```
s = $;
R1 = (s + 2)/(2*s + 3);
R2 = derivative(R1);        // 実有理多項式の 1 階導関数
R2 = derivative(R1,2);      // 実有理多項式の 2 階導関数
```


Matrix rotateUp(M1, k);	
	全ての列を上に k 回転する
	k 省略すると，シフト数は 1 となる
Matrix rotateDown(M1, k);	
	全ての列を下に k 回転する
	k 省略すると，シフト数は 1 となる
Matrix reshape(M1, m, n);	
	行列 M1 から行方向に成分を取り出して m x n の行列を作る。
List sort(M1);	成分を昇べきの順にソートする
	複素行列の場合，絶対値でソート
{M2, idx} = sort(M1);	M2: ソートした結果の行列
	idx: ソートに使った指数
	M2 = M1(idx)
List sort(Array A1);	成分を昇べきの順にソートする
	複素行列の場合，絶対値でソート
{A2, idx} = sort(A1);	A2: ソートした結果の配列
	idx: ソートに使った指数
	A2 = A1(idx)
Matrix max(M1, M2);	M1 と M2 の大きい方の成分からなる行列（複素行列の場合，絶対値によって比較する
Matrix min(M1, M2);	M1 と M2 の小さい方の成分からなる行列（複素行列の場合，絶対値によって比較する
Matrix rem(M1, i1);	M1 の各成分に対して rem() を計算する
Matrix rem(M1, r1);	M1 の各成分に対して rem() を計算する
Matrix rem(M1, M2);	M1 と M2 の成分毎に rem() を

```
List OdeHybridAuto(Real t0, Real t1, Real dt, Matrix x,
                   void dx(), void input(), Real eps)
                   : dtsav=1.0E-3, dtmin=EPS, dtmax=abs(t1-t0)/5
List OdeHybridAuto(Real t0, Real t1, Real dt, Matrix x,
                   void dx(), void input(), Real eps, Real dtsav)
                   : dtmin=EPS, dtmax=abs(t1-t0)/5
List OdeHybridAuto(Real t0, Real t1, Real dt, Matrix x,
                   void dx(), void input(), Real eps,
                   Real dtsav, Real dtmin)
                   : dtmax=abs(t1-t0)/5
List OdeHybridAuto(Real t0, Real t1, Real dt, Matrix x,
                   void dx(), void input(), Real eps,
                   Real dtsav, Real dtmin, Real dtmax)
```

きざみを自動調節しながら 4 次のルンゲ- クッタの公式にしたがって
指定された誤差内で常微分方程式の [t0,t1] 時刻の解を求める。
サンプリング間で外部入力は一定値をとる。
ただし，きざみの最小値と最大値は dtmin と dtmax であり，
dtmax < dtsav のとき，dtmax = dtsav に設定される。
{T, X, U} = OdeHybridAuto();
T: 時間の系列
X: 状態ベクトルの時系列
U: 入出力ベクトルの時系列

```
Matrix  rkf45(Real t, Matrix x, void dx())
                : dt = 1.0E-7, u = Z(0);
Matrix  rkf45(Real t, Matrix x, void dx(), Matrix u)
                : dt = 1.0E-7
Matrix  rkf45(Real t, Matrix x, void dx(), Matrix u,
                Real dt)
```

RKF45 の公式にしたがって常微分方程式の解を求める。

13.2　複素多項式

複素多項式の入力方法には，複素数を係数として記述する方法と一対の実多項式をコンマで “,” 区切って丸括弧（ と ）で囲む方法がある。実部または虚部だけを指定し，もう一方に * を指定して複素多項式を記述することもできる。

```
CP1 = (P1,P2);  // CP1 = P1 + P2*i; (P1, P2: 実多項式)
CP1 = (P1,*);   // CP1 = P1 + 0*i;  (P1: 実多項式)
CP1 = (*,P1);   // CP1 = 0  + P1*i; (P1: 実多項式)
```

複素多項式の実部と虚部は，関数 Re() と Im() を用いて取り出すことができる。

```
P1 = Re((P1,P2));  // 実部多項式 = Re(複素多項式)
P2 = Im((P1,P2));  // 虚部多項式 = Im(複素多項式)
```

13.3　多項式の係数の参照

多項式の名前の次に次数を丸括弧（ と ）で囲むことによって多項式の係数を参照することができる。ただし，複素多項式の係数には実数と複素数を代入できるが，実多項式の係数には実数しか代入できない。また，定義されている次数以上の係数を参照することはできない。

```
r1 = P1(1);   // 実多項式 P1 の 1 次の係数
c1 = CP1(2);  // 複素多項式 CP1 の 1 次の係数
P1(0) = r2;   // 実数 r1 を多項式 P1 の 0 次の係数に代入
CP1(3) = c2;  // 複素数 c1 を多項式 CP1 の 3 次の係数に代入
```


U: 入出力ベクトルの時系列

`void OdeStop();`	Ode(), OdeAuto(), OdeHybrid(), OdeHybridAuto(), Ode45(), Ode45Auto(), Ode45Hybrid(), Ode45HybridAuto() の実行を強制的にストップする．
`List OdeXY(Real t);`	Ode(), OdeAuto(), OdeHybrid(), OdeHybridAuto(), Ode45(), Ode45Auto(), Ode45Hybrid(), Ode45HybridAuto() の実行中に時刻 t の状態と入出力を返す。 `{X_d, Y_d} = OdeXY(t)` 遅れのあるシステムのシミュレーションに用いる。

```
Matrix M1, M2;
Integer row, col;
```

`List size(M1);`	行数と列数のリスト `{row, col} = size(M1);`

`List eig(M1);`	固有値問題
	`{val, vec} = eig(M1);`
	`M1 * vec = vec * val;`
	val: 固有値（対角行列）
	vec: 固有ベクトル

`List eig(M1, M2);`	一般固有値問題
	`{val, vec} = eig(M1, M2);`
	`M1 * vec = M2 * vec * val;`

12.6　文字列への変換

MATX のいろいろなオブジェクト (整数，実数，複素数，多項式) を文字列に変換する方法を紹介する。まず，型変換関数 String() により 1 個のオブジェクトを文字列に変換できる。以下に例を示す。

```
is = String(1);                    // 整数
rs = String(3.14);                 // 実数
cs = String((3,4));                // 複素数
ps = String(2.1*$^2 + 3.2*$ + 4);  // 多項式
```

そして，関数 sprintf() を用いると複雑な文字列変換が可能となる。以下に例を示す。

```
G = ($+1)/($^2 + 3*$ + 4);
Gs = sprintf("(%s)/(%s)", String(Nu(G)), String(De(G)));
```

関数 int2str() は ASCII コードを文字列に変換する。例えば，文字 A，B，C の ASCII コードは 65, 66, 67 であるから

```
str = int2str(65) + int2str(66) + int2str(67);
```

の結果，str には文字列 ABC が代入される。型変換関数 String() を用いれば，ASCII コードからなる行列を文字列に変換できる。上の例は

```
str = String([65 66 67]);
```

のように置き換えられる。また，型変換関数 Matrix() を用いれば，文字列を ASCII コードからなる行列に変換できる。上の例の逆変換は

```
a = Matrix("ABC");
```


`Array any_row(Matrix M1);`	行毎にゼロでない成分がある (真 = 1，偽 = 0) の縦ベクトル
`Array finite(Array A1);`	各成分に対して有限の有無を調べる 無限大あるいは NaN である成分の所が 0，それ以外の成分の所が 1 である 配列を返す
`Array isnan(Array A1);`	各成分に対して NaN の有無を調べる NaN である成分の所が 1， それ以外の成分の所が 0 である 配列を返す
`Index find(Array A1);`	ゼロでない成分の指数 A1 がベクトルでないとき， 行順に成分を調べる
`Index find(Matrix M1);`	ゼロでない成分の指数 M1 がベクトルでないとき， 行順に成分を調べる

`Real max(Array A1);`	最大成分
`Real min(Array A1);`	最小成分
`Real frobnorm(Array A1);`	実配列フロベニウスノルム
`Real frobnorm(CoArray A1);`	複素配列フロベニウスノルム
`Polynomial frobnorm(PoArray A1);`	多項式配列フロベニウスノルム
`Rational frobnorm(RaArray A1);`	有理多項式配列フロベニウスノルム

`Real sum(Array A1);`	全ての成分の和
`Complex sum(CoArray A1);`	全ての成分の和
`Polynomail sum(PoArray A1);`	全ての成分の和
`Rational sum(RaArray A1);`	全ての成分の和
`Real prod(Array A1);`	全ての成分の積

計算する

```
Array data;
Integer n;

Array fft(data);        高速フーリエ変換
Array fft(data, n);     高速フーリエ変換
                        n = 2^m
Array ifft(data);       高速逆フーリエ変換
Array ifft(data, n);    高速逆フーリエ変換
                        n = 2^m


Array A1;
Integer i1;
Real tol;

Array sgn(A1);          各成分の sgn() の結果からなる配列
Array abs(A1);          各成分の abs() の結果からなる配列
Array arg(A1);          各成分の arg() の結果からなる配列
Array exp(A1);          各成分の exp() の結果からなる配列
Array sqrt(A1);         各成分の sqrt() の結果からなる配列
Array log(A1);          各成分の log() の結果からなる配列
Array log10(A1);        各成分の log10() の結果からなる配列
Array conj(A1);         各成分の conj() の結果からなる配列
Array inv(A1);          各成分の inv() の結果からなる配列 (A1~)
Array sin(A1);          各成分の sin() の結果からなる配列
Array asin(A1);         各成分の asin() の結果からなる配列
Array sinh(A1);         各成分の sinh() の結果からなる配列
Array asinh(A1);        各成分の asinh() の結果からなる配列
Array cos(A1);          各成分の cos() の結果からなる配列
```

16.3　リスト成分の付加

リストに新しい成分を付加するには関数 appendlist() を使うか，+オペレータを使う。

```
x = {sqrt(3.0), ((1.0+2.0)*4.0,:)};
appendlist(x, "Hello");
// x ---> {sqrt(3.0), ((1.0+2.0)*4.0,:), "Hello"}

y = {sqrt(3.0), ((1.0+2.0)*4.0,:)};
y = y + {"Hello"};
// y ---> {sqrt(3.0), ((1.0+2.0)*4.0,:), "Hello"}
```

16.4　リストの合成と取り出し

長いリストは，短いリストを成分として作ることができる。リスト x とリスト y から長いリスト z を作るには，

```
x = {4, (3,4), [1 2]};
y = {"Hello", 2.0};
z = {x, y};
// z = {{4, (3,4), [1 2]}, {"Hello", 2.0}}
```

のように書く。また，成分の範囲を指定することにより，リストの任意の部分を取り出すことができる。例えば，

```
x2 = x(1:2);                    // x2 = {4, (3,4)}

x3 = {4, (3,4), [1 2]}(1:2);  // x3 = {4, (3,4)}
```

により，リスト x から最初の 2 個の成分を取り出すことができる。

```
String path, format, mode;

String int2str(c)           ASCII コード  c  の文字を文字列に
                            変換する。
Integer kbhit()             標準入力にデータがあるかどうか
                            調べる。1: 有, 0: 無
Integer getch()             標準入力からエコー無しで 1 文字取
                            り込み , その文字コードを返す。
Integer fopen(path, mode);   ファイルオープン
Integer fclose(fd);               ファイルクローズ
Integer printf(format, ...);       画面表示 (matx)
Integer fprintf(fd, format, ...)   ファイル出力
                                   fd は fopen() の返り値
Integer fprintf(file, format, ...);
                                   ファイル出力
                                   file に追加される
List    scanf(format, ...);        キーボードから入力
List    fscanf(fd, format, ...);   ファイル入力
String  sprintf(format, ...);      文字列変換
```

10.4　実数関数

実数関数の一覧を以下に示す。関数の引数は実数でなければならないので , 整数を引数として渡すにはクラス変換関数 Real() によって実数に変換する。

```
Integer i1;
Real r1, r2, x, y, tol;

Real rem(r1, r2)            割算の余り
                            (r1 - r2*fix(r1/r2))
```



```
   Func void main()
   {
     Matrix A, B, Q, R, P;      // 局所変数の宣言
     Matrix CRiccati();         // 関数の宣言
05:
     read A, B, Q, R;           // 行列エディタを起動
     P = CRiccati(A, B, Q, R); // CRiccati() を呼ぶ
     print P;                   // P を表示
   }
10:
   Func Matrix CRiccati(A, B, Q, R)
     Matrix A, B, Q, R;
   {
     Integer n;
15:  Matrix H, U, V, vec, P;

     n = Rows(A);
     H = [[ A   , -B*R~*B#]      // ハミルトン行列
          [-Q   , -A#     ]];
20:
     vec = eigvec(H);           // H の固有ベクトル
     U = vec(    1:n, n+1:2*n); // 安定な固有値に対応
     V = vec(n+1:2*n, n+1:2*n); // する固有ベクトルを
                                // 取り出す
25:  if (isreal(A) && isreal(B) &&
         isreal(Q) && isreal(R)) {
         P = Re(V*U~);
     } else {
         P = V*U~;
30:  }

     return P;
   }
```

List 1:　連続時間の線形システムのリカッティ方程式の解

9.5　Break 文と Continue 文

先頭または最後でテストする以外，時によっては，ループからの脱出を制御できる方が便利な場合がある。

```
break;
```

break 文は，for，while から早く抜け出るための方法である。この break 文を使うと，最も内側のループから直ちに抜け出すことができる。

```
continue;
```

次に continue 文は break 文と関係しているが，使われることは少ない。これは，囲んでいる for あるいは while ループの次の繰り返しを開始させるためのものである。while ループにおいては，これはテスト部分を直ちに実行することを意味する。


```
   Func void main()
   {
     Matrix A, B, C, G;     // 変数の宣言
     Matrix Faddeev();
05:
     read A, B, C;          // 行列の読み込み
     G = Faddeev(A, B, C); // Faddeev() を呼ぶ
     print G;               // G を表示
   }
10:
   Func Matrix Faddeev(A, B, C)
     Matrix A, B, C;
   {
     Matrix Gamma, N;
15:  Polynomial s, ch;
     Integer i, n;

     s = $;             // 多項式の変数
     n = Rows(A);       // システムの次数
20:
     Gamma = Z(n,A);
     ch = Polynomial(Z(1,n+1));
                        // 特性多項式
     ch(n) = 1.0;       // 最高次の係数は 1
25:
     Gamma(n-1,A) = I(n);
     ch(n-1) = - trace(A);

     for (i = n-2; i >= 0; i--) {
30:      Gamma(i,A) = A*Gamma(i+1,A) + ch(i+1)*I(n);
         ch(i) = - trace(A*Gamma(i,A))/(n - i);
     }

     N = Z(Rows(C), Cols(B));
35:  for (i = 0; i < n; i++) {
         N = N + C*Gamma(i,A)*B * s^i;
     }

     return N/ch;       // 伝達関数を返す
40:}
```

List 3:　ファディーブのアルゴリズムによる伝達関数の計算

```
Func void main(argc, argv)
    Integer argc;
    List argv;
{
    Integer a, b;

    print "Command Name is", argv(1, String), ".\n";
    a = Integer(argv(2, String));
    b = Integer(argv(3, String));
    print a + b;
}
```

は，コマンド名と第 1 引数と第 2 引数の和を計算して表示する。

すために使われ，状態ベクトル (x) と同じ次元である。第 2 引数 (t) は，現時刻を表す実数である。第 3 引数 (x) は，現時刻の状態ベクトルであり，第 4 引数 (u) は，動的要素への入力ベクトルである。関数 link_eqs() は 3 個の引数を持つ。第 1 引数 (u) は，入力ベクトルを返すために使われる。第 2 引数 (t) は，現時刻を表す実数である。第 3 引数 (x) は，現時刻の状態ベクトルである。

8.4.2　一般的な可変長引数関数

```
Func void foo()
{
  Matrix a, b, c;
  Matrix afo(...);

  read a, b, c;

  print afo();
  print afo(a);
  print afo(a, b);
  print afo(a, b, c);
}

Func Matrix afo(a, b, c, ...)
  Matrix a, b, c;
{
  if (nargs == 0) {
    return Z(0);
  } else if (nargs == 1) {
    return a;
  } else if (nargs == 2) {
    return a + b;
  } else if (nargs == 3) {
    return a + b + c;
  }
}
```

可変長引数関数の定義は，引数リストの最後に “...” を追加することを除いて普通の関数の定義と変わらない。また，可変長引数関数を呼び出す側では，普通の関数を呼び出す時と同じように，関数を定義しなければならない。宣言の方法は，関数の型に続いて “関数名 (...)” を書く。可変長引数関数が呼び出されると，呼出時に渡された引数の個数が，特別な変数 nargs にセットされる。

17.5.3　動的フィードバック制御のシミュレーション

下に示すようなシステムに動的フィードバックを施す場合の一般的なシミュレーションを行なう。関数 init_params() はプラントとコントローラのパラメータを設定する関数であり，関数 main() はパラメータと初期状態を設定してシミュレーションを行ない，結果をファイル cont.mat に保存する。diff_eqs() は動的要素の微分方程式を記述するための関数であり，link_eqs() は動的要素の接続関係を記述するための関数である。

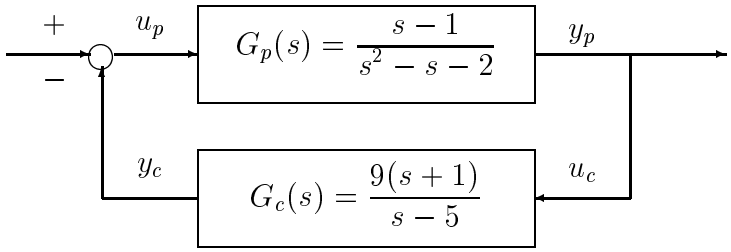


図 17.1: 制御対象とコントローラ

ここでは，制御対象と制御器のパラメータを大域変数として宣言するプログラムとパラメータを静的な局所変数として宣言するプログラムを示す。

ファイルから読み込みやキーボードから入力でパラメータを変更しながらシミュレーションを繰り返す場合は，大域変数を用いる方が都合が良い。しかし，大域変数はプログラム中の任意の場所で修正される可能性があり，変数名が衝突するなどの問題があるので，大規模なプログラムには用いない方が良い。一方，静的変数はそれが定義された関数内でのみ有効なので，他の関数の局所変数や大域変数と衝突することがないので大規模プログラムに有効である。

字と小文字は区別される。関数名として，組み込み関数の名前や，コマンド名等の予約語を使用することはできない。

```
Integer    func_int1(), func_int2();
Real       func_real1(), func_real2();
Complex    func_cmp1(), func_cmp2();
String     func_str1(), func_str2();
Polynomial func_poly1(), func_poly2();
Rational   func_rat1(), func_rat2();
Matrix     func_mat1(), func_mat2();
Array      func_arr1(), func_arr2();
Index      func_ind1(), func_ind2();
List       func_list1(), func_list2();
void       func_void1(), func_void2();
```

8.1.1　require 修飾子付き関数宣言

関数宣言を require 修飾子付きで行うと，その関数が定義されているファイルを指定することができる。例えば，

```
void mg_main(), mg_sub1(), mg_sub2() require "mg.mm";
```

は，関数 mg_main() ，mg_sub1() ，mg_sub2() がファイル mg.mm で定義されていることを宣言する。この宣言により，3 個の中のどの関数が呼ばれても，自動的にファイル mg.mm が読み込まれる。ファイルが 1 度読み込まれれば，別の関数が呼ばれても mg.mm が再読み込みされることはない。

従来は，関数名とファイル名が一致しなければ，宣言された関数が呼び出されても必要なファイルが自動的に読み込まれなかった。require 修飾子を用いれば関数名と異なる名前のファイルに関数を定義できる。例えば，

```
Matrix PoleAssignSISO() "poleasgn.mm";
```

によって，長い名前の関数 PoleAssignSISO() を短い名前のファイル


```
Func void link_eqs(UY, t, X)
  Real t;
  Matrix UY, X;
{
  static Integer init;
  static Matrix cp, cc, dc;
  Matrix xp, up, yp, xc, yc, uc;

  if (init == 0) {
    init = 1;
    cp = [-1, 1];  // 制御対象のパラメータ
    cc = [54];     // 制御器のパラメータ
    dc = [9];
 }

  xp = X(1:2,1);           // 制御対象の状態
  xc = X(3:3,1)            // 制御器の状態

  yp = cp*xp;              // 制御対象の出力
  uc = yp;                 // yp を uc に接続

  yc = cc*xc + dc*uc;      // 制御器の出力
  up = - yc;               // yc を up に接続

  UY = [up uc]';
}
```

List 7-4:　動的フィードバック制御のシミュレーション (静的変数使用)

```
Func void afo()
{
  static Integer i;    // 0 に初期化される
  static Real r;       // 0.0 に初期化される
  static Complex c;    // (0,0) に初期化される
  static Polynomial P; // 0 に初期化される
  static Rational R;   // 0 に初期化される
  static List l;       // {} に初期化される

  i = i + 1;
  r = r + 1;
  c = c + (1,0);
  P = P + 1;
  R = R + 1;
  appendlist(l, 1);
}
```

ただし，行列は 0 x 0 の行列として初期化されるので，length() 関数を用い大きさを調べるなどして，陽に初期化する必要がある。

```
Func void afo()
{
  static Matrix A;

  if (length(A) == 0) {
    A = [1];
  } else {
    A = A + [1];
  }
}
```

7.4　暗黙の型変換

異なる型のスカラーデータの代入時，次に示すルールに従って自動的に型変換が行なわれる。

謝辞

MATX の開発において御助言，御指導を賜わりました東京工業大学情報理工学研究科情報環境学専攻古田勝久教授に感謝致します。

また，東京工業大学情報理工学研究科情報環境学専攻古田研究室，三平研究室，同工学部制御システム工学科山北研究室，寄附講座の人々の貴重なフィードバックに感謝します。特に，開発当初より多くのバグを指摘して下さり，多くのアドバイスを賜わりました東京工業大学工学部制御システム工学科山北昌毅助教授に感謝致します。

最後になりました，MATX の開発に利用させて頂いた多くのフリーウェアの作者，移植者の方々に感謝致します。

[19] 古賀 雅伸 and 古田 勝久. 制御系解析・設計支援ツール. 第２９回計測自動制御学会学術講演会, pages 467–468, 1990.

[20] 石田 晴久 監訳. プログラミング言語 *C*. 共立出版株式会社, 1981.

[21] 石田 晴久 監訳. *UNIX* プログラミング環境. アスキー 出版局, 1985.

[22] 石田 晴久 監訳. プログラミング言語 *C* 第 *2* 版. 共立出版株式会社, 1989.

[23] 中島 信行. パソコンで学割り込みプログラミング, pages 100–201. CQ 出版社, 1992.

```
Y = [0. .48 .84 1. .91 .6 .14];
xplot(Array(Y));
```

この結果 xplot が起動し，データが自動的にスケールされてキャンバス上に表示され，X 軸と Y 軸が描かれる。データが表示された後で，マウスを使ってラベル，タイトル，スケール，フォントを変更したり，画面のハードコピーをネットワークに接続されているプリンタに出力したり，FIG コードに落して TEX の文書に取り込んだりすることができる。

もし，X と Y が同じ長さのベクトルなら，xplot(X,Y) によって X の成分と Y の成分を対応させた線形プロットが作成される。例えば，

```
t = [0.0:0.05:4*PI];
Y = sin(t);
xplot(t, Y);
```

によって，時刻 0.0 から 4*PI までの正弦波を描くことができる。

6.2.2　複数の線

1 つのグラフ上に複数の線をプロットする方法は 2 つある。1 つ目の方法は，関数 xplot() に xplot(X,Y) のように 2 個の引数を渡す方法である。ただし，X と Y の行の長さは同じでなければならない。X と Y がベクトルか行列かに応じて以下のようにプロットされる線が異なる。

- もし Y が行列で X がベクトルなら，xplot(X,Y) は，ベクトル X に対して Y の各行を対応させて異なった線種で線を描く。
- もし X が行列で Y がベクトルなら，xplot(X,Y) は，ベクトル Y に対して X の各行を対応させて異なった線種で線を描く。
- もし X と Y の両方が行列なら，xplot(X,Y) は，Y の各行に対して X の各行を対応させて異なった線種で線を描く。

マシーン	M
Sun	0x10
Appolo	0x10
Macintosh	0x10
PC	0x20
Other Intel	0x30
VAX D-float	0x40
VAX G-float	0x50

O は，行列データの保存される順序を表す。O が 0 なら，列毎にデータが保存され O が 1 なら，行毎にデータが保存される。

P は，データ精度を表す。

データ精度	P
倍精度 (64 ビット) 浮動小数点	0
単精度 (32 ビット) 浮動小数点	1
32 ビット符合付き整数	2
16 ビット符合付き整数	3
16 ビット符合なし整数	4
8 ビット符合なし整数	5

C は，データの型を表す。

6.1.7 (ポストスクリプト—FIG コード) ファイルに出力

関数 gplot_psout() や mgplot_psout() を使えば，ウィンドウに表示されたグラフを簡単にポストスクリプトファイル (EPS フォーマット) に保存できる。例えば，正弦波と余弦波をウィンドウに表示した後，ポストスクリプトファイル (sin_cos.ps) に出力するには次のようにする。

```
gplot(sin([0.0:0.1:2*PI]));
gplot_replot(cos([0.0:0.1:2*PI]));
gplot_psout("sin_cos.ps");
```

関数 gplot_figcode() や mgplot_figcode() を使えば，ウィンドウに表示されたグラフを簡単に FIG コードファイルに保存できる。例えば，正弦波と余弦波をウィンドウに表示した後，FIG コードファイル (sin_cos.fig) に出力するには次のようにする。

```
gplot(sin([0.0:0.1:2*PI]));
gplot_replot(cos([0.0:0.1:2*PI]));
gplot_figcode("sin_cos.fig");
```

6.1.8 Windows95/NT で mgplot を使う

Windows95/NT の gnuplot はパイプに対応してないので，mgplot はコマンドをファイルに書き込み，そのファイル名を引数として gnuplot を起動する。mgplot は gnuplot に渡した全てのコマンドを記録しているが，グラフを描画する毎に gnuplot を起動し直さなければならない。起動回数を減らすには mgplot_hold() 関数が役に立つ。この関数でホールドされたウィンドウに対応する gnuplot は，ホールドが解除されるまで起動されることはない。以下の例では，最終行でホールドが解除されて初めて，gnuplot が起動される。

	複素多項式	2×(degree + 1) 個の精度に応じた実数 (4 バイトまたは，8 バイト)。
有理多項式	実有理多項式	ndegree + 1 個の精度に応じた実数 (4 バイトまたは，8 バイト)，ddegree + 1 個の精度に応じた実数 (4 バイトまたは，8 バイト)。
	複素有理多項式	2×(ndegree + 1) 個の精度に応じた実数 (4 バイトまたは，8 バイト)，2×(ddegree + 1) 個の精度に応じた実数 (4 バイトまたは，8 バイト)。
行列	実行列	rows×cols 個の精度に応じた実数 (4 バイトまたは，8 バイト)。
	複素行列	2×(rows×cols) 個の精度に応じた実数 (4 バイトまたは，8 バイト)。
	実多項式行列	rows×cols 個の実多項式。(degree + 1 個の精度に応じた実数 (4 バイトまたは，8 バイト))。
	複素多項式行列	rows×cols 個の複素多項式。(degree + 1 個の精度に応じた実数 (4 バイトまたは，8 バイト))。
	実有理多項式行列	rows×cols 個の実有理多項式。(ndegree + 1 個の精度に応じた

6.1.4 1個のウィンドウに複数のグラフ

mgplot_subplot(win,m,n,p) は，ウィンドウ win を m 行 n 列の長方形に面に分割し，p 番目の面をカレント面にする。これ以後，mgplot() 関連の全ての操作はこの面に対して行われる。面はウィンドウの一番上の行の左から右に，上から下の行に向かって数える。mgplot_subplot(win,m,n,p) は，指定した位置にすでに面が存在していれば，その面をカレント面にする。mgplot(win,1,1) または mgplot(win,1,1,1) はウィンドウに存在している全ての面を消去し，ウィンドウ全体を 1 個の面にする。1 個ウィンドウを 4 分割し，4 個の異なるグラフをプロットする例を以下に示す。

```
w = linspace(-PI,PI);
s = sin(w);
c = cos(w);
mgplot_subplot(1,2,2,1); mgplot(1,w,s);
mgplot_subplot(1,2,2,2); mgplot(1,w,c);
mgplot_subplot(1,2,2,3); mgplot(1,w,2*s);
mgplot_subplot(1,2,2,4); mgplot(1,w,2*c);
```

この例では，左上に sin(x)，右上に cos(x)，左下に sin(2x)，右したに cos(2x) が表示される。gnuplot_subplot() を使用するには，gnuplot 3.5 (pre 3.6)[1]が必要である。

6.1.5 ウィンドウのクリアと消去

ウィンドウをクリアするには，関数 gplot_clear() や関数 mgplot_clear() を使う。ウィンドウをクリアしても，関数 greplot() や関数 mgreplot() を呼び出すと，前に表示されていた線が再び表示される。一方，ウィンドウを消去するには，関数 gplot_quit() や関数 mgplot_quit() を使う。一旦，ウィンドウを消去してしまうと，関数 greplot() や関数 mgreplot() を呼び出してもデータは表示されない。関数 mgplot_reset() や関数 gplot_reset()

[1] MATX の FTP サーバから入手可能。

付 録B　メモリ管理

前節で述べたように MATX はクラスライブラリ MaTX-Lib を利用している。この節では，クラスライブラリ MaTX-Lib のメモリ管理機構について述べる。行列クラス，多項式クラス，有理多項式クラス，リストクラスは別々にメモリを管理しているが，全て同じ方式なので，ここでは行列クラスについて説明する。

B.1 行列クラスのメモリ管理

図 B.1 に示すように，メモリ管理はメモリの状態遷移という概念に基づいて行なわれる。メモリは関数呼び出しによって

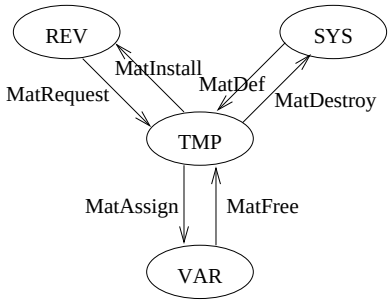


図 B.1: MATX のメモリ管理

1. SYS 状態 (OS によって管理されている状態)

- もし X と Y の両方が行列なら，gplot(X,Y)(mgplot(win,X,Y)) は，Y の各行に対して X の各行を対応させて異なった線種で線を描く。
- もし gplot(Y)(mgplot(win,Y)) のように X が指定されなかったら，Y の各成分とその成分の指数を対応させて異なった線種で線を描く。

1 つのグラフ上に複数の線をプロットする 2 つ目の方法は，関数 greplot()，関数 mgreplot() を使う方法である。これらの関数を使うと，古い線に新しい線が重ねてプロットされる。 例えば，

表 6.2: グラフィック関数 2(gnuplot)

関数名	引数	機能
greplot()		gplot_replot()
greplot(Y)	Array X, Y	
greplot(X, Y)		線形 x–y プロット
greplot(X, Y, titles)	List titles	
greplot(X, Y, titles, cmds)	List cmds	
mgreplot(win)	Integer win	mgplot_replot(win)
mgreplot(win, Y)		
mgreplot(win, X, Y)	Array X, Y	線形 x–y プロット
mgreplot(win, X, Y, titles)	List titles	
mgreplot(win, X, Y, titles, cmds)	List cmds	

```
gplot(X1, Y1);
greplot(X2, Y2);
...............;
greplot(X5, Y5);
```

変数 X1,Y1,X2,Y2 などは，行列の対である。それぞれの x-y 対が 1 つのグラフ上にプロットされる。この方法を使えば，異なった長さの線を同一グラフ上にプロットすることができる。

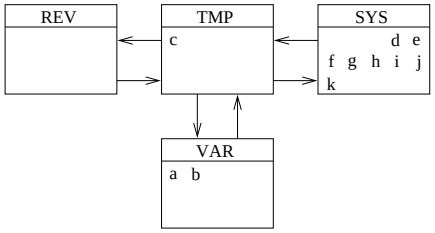


図 B.6: 行列のメモリ管理 (5)

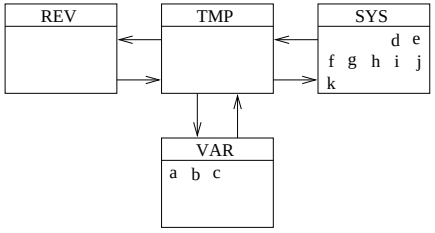


図 B.7: 行列のメモリ管理 (6)

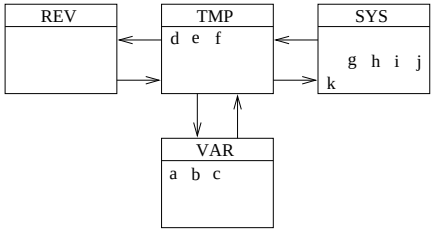


図 B.8: 行列のメモリ管理 (7)

$(C-D)$, $(C+D)*(C-D)$ の計算結果が入っている。d と e は REV 状態から g は SYS 状態から TMP 状態に移った。式 (5) の右辺が左辺に代入されたときのメモリ状態は図 B.11 のようになる。$(C+D)*(C-D)$ の計算結果である g だけが VAR 状態となり，d と e は REV 状態となる。関数が終了し

付 録C　演算

二項算術演算子は+，-，*，.*，/，./，\，.\，^，.^である。+と-は同じ優先度を持つが，それは，同じ優先度の*，.*，/，./，\，.\，^，.^よりも低い。さらに，後者の優先度は単項演算子+，-，~，'，#の優先度より低い。算術演算子は左から右へまとめられて評価される。表 1 は，全演算子の優先度と結合規則をまとめたものである。 同一行の演算しは同じ優先度

表 C.1: 演算子の優先度と結合規則

演 算 子	結合規則
( ) \$ [ ]	左から右
++ --	右から左
^ ~ ' # .^ .~	右から左
@ ! -	右から左
* / \ .* ./ .\	左から右
+ -	左から右
< <= > >=	左から右
== !=	左から右
&&	左から右
||	左から右
=	右から左

をもつものである。各行は優先度が減少する順に並べてある。例えば，*，/ は同じ優先度もち，2 項の+と-よりも高い。

二項演算においてクラスレベルの低い方の値は，下記の規則にしたがって

5.1　実行列の編集

図 5.1 は，行列エディタを使って，(50×10) の実行列の (18 行 ~ 32 行)(3 列 ~ 6 列) を入力している様子を示している。

```
 Name :  [ A ]
 Size :  [  50, 10 ]    Filename : A

           (  3)         (  4)         (  5)         (  6)
( 18)  -1.957854E-01  9.554574E-02 -1.957854E-01  9.554574E-02
( 19)  -1.953817E-01  9.751419E-02 -1.953817E-01  9.751419E-02
( 20)  -1.949779E-01  9.948264E-02 -1.949779E-01  9.948264E-02
( 21)  -1.945741E-01  1.014511E-01 -1.945741E-01  1.014511E-01
( 22)  -2.084542E-01  8.378548E-02 -2.084542E-01  8.378548E-02
( 23)  -2.080504E-01  8.575394E-02 -2.080504E-01  8.575394E-02
( 24)  -2.076466E-01  8.772239E-02 -2.076466E-01  8.772239E-02
( 25)  -2.072429E-01  8.969085E-02 -2.072429E-01  8.969085E-02
( 26)  -2.068391E-01  9.165930E-02 -2.068391E-01  9.165930E-02
( 27)  -2.064353E-01  9.362776E-02 -2.064353E-01  9.362776E-02
( 28)  -2.060315E-01  9.559621E-02 -2.060315E-01  9.559621E-02
( 29)  -2.056277E-01  9.756466E-02 -2.056277E-01  9.756466E-02
( 30)  -2.052239E-01  9.953312E-02 -2.052239E-01  9.953312E-02
( 31)  -2.048201E-01  1.015015E-01 -2.048201E-01  1.015015E-01
( 32)  -2.044164E-01  1.034700E-01 -2.044164E-01  1.034700E-01
```

図 5.1: 行列エディタ (実行列)

5.2　複素行列の編集

C コマンドによって，実行列を複素行列に変換したり，実部行列や虚部行列を複素行列に戻すことができる。(ただし，matx のみ) 実部行列や虚部行列を取り出して編集する時，行列のファイルへの書き込みやファイルから読み込み，1 行削除，1 列削除，大きさの変更は，できない。図 5.2 は，行列エディタを使って，(30×10) の複素行列の (12 行 ~ 26 行)(3 列 ~ 4 列) を入力している様子を示している。図 5.3 と図 5.4 は，R コマンドと I コマンドによって，複素行列の実部行行列と虚部行列を取り出して編集している様子を

C.3　積 (Multiplication)

行列の積の結果が 1x1 の行列になっても，その値はスカラーではなく，行列として扱われる。

整数	= 整数 * 整数
実数	= 実数 * 実数
複素数	= 複素数 * 複素数
実多項式	= 実多項式 * 実多項式
複素多項式	= 複素多項式 * 複素多項式
実有理多項式	= 実有理多項式 * 実有理多項式
複素有理多項式	= 複素有理多項式 * 複素有理多項式
実 (行列|配列)	= 実 (行列|配列) * 実 (行列|配列)
複素 (行列|配列)	= 複素 (行列|配列) * 複素 (行列|配列)
実多項式 (行列|配列)	= 実多項式 (行列|配列) * 実多項式 (行列|配列)
複素多項式 (行列|配列)	= 複素多項式 (行列|配列) * 複素多項式 (行列|配列)
実有理多項式 (行列|配列)	= 実有理多項式 (行列|配列) * 実有理多項式 (行列|配列)
複素有理多項式 (行列|配列)	= 複素有理多項式 (行列|配列) * 複素有理多項式 (行列|配列)
実行列	= 実行列 .* 実 (行列|配列)
複素行列	= 複素行列 .* 複素行列
実多項式行列	= 実多項式行列 .* 実多項式行列
複素多項式行列	= 複素多項式行列 .* 複素多項式行列
実有理多項式行列	= 実有理多項式行列 .* 実有理多項式行列
複素有理多項式行列	= 複素有理多項式行列 .* 複素有理多項式行列

C.4　スカラ倍 (Scalar Multiplication)

実 (行列|配列)	= (整数|実数) * 実 (行列|配列)
実 (行列|配列)	= 実 (行列|配列) * (整数|実数)
複素 (行列|配列)	= (整数|実数) * 複素 (行列|配列)
複素 (行列|配列)	= 複素 (行列|配列) * (整数|実数)
複素 (行列|配列)	= 複素数 * 実 (行列|配列)
複素 (行列|配列)	= 実 (行列|配列) * 複素数
複素 (行列|配列)	= 複素数 * 複素 (行列|配列)
複素 (行列|配列)	= 複素 (行列|配列) * 複素数

```
  if ((pid = popen("gnuplot -title sample")) < 0) {
      error("Can't open %s", "gnuplot");
  }
  fprintf(pid, "plot sin(x)\n");
  pause;
  fprintf(pid, "quit\n");
  pclose(pid);
}
```

4.6　ファイルのアクセス権を調べる

ファイルが存在しているか (mode==0)，そのファイルを実行できるか (mode==1)，そのファイルに書き込めるか (mode==2)，そのファイルを読み込めるか (mode==4)，ファイルのアクセス権を調べるには，ファイルのパス (ファイル名) を引数として関数 access() を呼ぶ。アクセス権がある場合，0 が返され，アクセス権がない場合，0 以外の値が返される。第 2 引数に調べたいアクセス権を整数の論理和で指定できる。モードを省略した場合，読み込みできるかを調べる。

```
  i = access("afo.mm");
  if (i == 0)
    print "afo.mm exists.";
  } else {
    print "afo.mm doesn't exist.";
  }

  i = access("afo.mm", 2);
  if (i == 0)
    print "afo.mm can be changed";
  } else {
    print "afo.mm cann't be changed.";
  }
```



```
実有理多項式      = 実多項式 \ 実多項式
複素有理多項式    = 複素多項式 \ 複素多項式
実有理多項式      = 実有理多項式 \ 実有理多項式
複素有理多項式    = 複素有理多項式 \ 複素有理多項式

実 (行列|配列)    = 実 (行列|配列) \ 実 (行列|配列)
複素 (行列|配列)  = 複素 (行列|配列) \ 複素 (行列|配列)

実行列            = 実行列 .\ 実行列
複素行列          = 複素行列 .\ 複素行列
有理多項式行列    = 多項式行列 .\ (実|複素|多項式|有理多項式) 行列
```

C.7　累乗 (Power)

指数が負の時，逆数 / 逆行列の指数の絶対値乗を意味する。特に多項式の場合，演算結果が有理多項式になることに注意すること。

```
整数                          = 整数 ^ 整数
実数                          = 実数 ^ 実数
複素数                        = 複素数 ^ 複素数
実多項式                      = 実多項式 ^ 整数
複素多項式                    = 複素多項式 ^ 整数
実有理課数                    = 実有理多項式 ^ 整数
複素有理課数                  = 複素有理多項式 ^ 整数
実 (行列|配列)                = 実 (行列|配列) ^ 整数
複素 (行列|配列)              = 複素 (行列|配列) ^ 整数
実多項式 (行列|配列)          = 実多項式 (行列|配列) ^ 整数
複素多項式 (行列|配列)        = 複素多項式 (行列|配列) ^ 整数
実有理多項式 (行列|配列)      = 実有理多項式 (行列|配列) ^ 整数
複素有理多項式 (行列|配列)    = 複素有理多項式 (行列|配列) ^ 整数
```

0 の行列を返す。fd は fopen() が返すファイルディスクリプタ，size は読み込むデータ数，type は読み込むデータの型である。type には，C 言語のデータの型を指定する。以下の表に指定できる型を示す。ただし，符合あり (signed) の型を指定できないことがある。また，fread(fd, size, type, 1) はバイナリデータの数値形式 (Endian) を変更して行ベクトルを返す。

表 4.1: バイナリデータの型

type	データ型
"char"	文字 (8 bits)
"signed char"	符合あり文字 (8 bits)
"unsigned char"	符合なし文字 (8 bits)
"short"	整数 (16 bits)
"signed short"	符合あり整数 (16 bits)
"unsigned short"	符合なし整数 (16 bits)
"int"	整数 (16,32,64 bits)
"signed int"	符合あり整数 (16,32,64 bits)
"unsigned int"	符合なし整数 (16,32,64 bits)
"long"	整数 (32 bits)
"signed long"	符合あり整数 (32 bits)
"unsigned long"	符合なし整数 (32 bits)
"float"	単精度浮動小数点数 (32 bits)
"double"	倍精度浮動小数点数 (32 bits)

関数 fwrite(fd, mat, type) は行列 mat の成分を指定したファイルに指定したデータ型のバイナリで書き込む。データは行の順に書き込まれる。fd は fopen() が返すファイルディスクリプタ，mat は書き込む行列，type は書き込むデータの型である。type には，C 言語のデータの型を指定する。

行列の成分を int 型のバイナリでファイルに保存し，そのデータを別の行列に読み込む例を以下に示す。

C.12　関係演算 (Relational Operation)

関係演算子は >, >=, <, <= である，これらは全て同じ優先度を持つ。ちょうど 1 つ下の優先を持つものとしては等価演算子 ==, != があり，これらは同じ優先度を持つ。関係演算子は算術演算子よりも優先度が低い。

```
整数 1 < 整数 2    整数 1 は整数 2 より小さい
整数 1 > 整数 2    整数 1 は整数 2 より大きい
整数 1 <= 整数 2   整数 1 は整数 2 より小さい，又は等しい
整数 1 >= 整数 2   整数 1 は整数 2 より大きい，又は等しい
整数 1 == 整数 2   整数 1 は整数 2 と等しい
整数 1 != 整数 2   整数 1 は整数 2 と等しくない
実数 1 < 実数 2    実数 1 は実数 2 より小さい
実数 1 > 実数 2    実数 1 は実数 2 より大きい
実数 1 <= 実数 2   実数 1 は実数 2 より小さい，又は等しい
実数 1 >= 実数 2   実数 1 は実数 2 より大きい，又は等しい
実数 1 == 実数 2   実数 1 は実数 2 と等しい
実数 1 != 実数 2   実数 1 は実数 2 と等しくない
行列 < 0           maxeigval(行列) < 0
行列 > 0           mineigval(行列) > 0
行列 <= 0          maxeigval(行列) <= 0
行列 >= 0          mineigval(行列) >= 0
行列 == 0          行列は零行列である
行列 != 0          行列は零行列でない
0 > 行列           maxeigval(行列) < 0
0 < 行列           mineigval(行列) > 0
0 >= 行列          maxeigval(行列) <= 0
0 <= 行列          mineigval(行列) >= 0
0 == 行列          行列は零行列である
0 != 行列          行列は零行列でない
```

付 録D　リアルタイム処理 (RTMATX)

MATX にリアルタイム処理機能を付加した RTMATX(リアルタイム MATX) を使えば，PC9801 シリーズおよび IBM PC 互換機の MS-DOS 上で動作するリアルタイム処理プログラムを作成することができる。

リアルタイムの割り込みなどのための関数が提供されており，ユーザはリアルタイム処理関数，画面表示 / コマンド入力関数を記述するだけでリアルタイム処理を行なうことができる。

制御系の解析設計は一般に下図のように行なわれるが，従来の制御系設計 CAD がサポートしているのは，同定・モデリングからシミュレーションまでの範囲であり，制御系の実現 (実際のプラントを実際に求められた制御器を用いて制御すること) は，専用マシン上でアセンブラ，C 言語などでプログラムを作成し，実行してきた。これでは，制御系を実行する前に複雑な専用マシンの使い方やアセンブラなどを覚えなければならないため，プログラムの開発，修正に多大な時間がかかる例が多くあった。RTMATX は，MATX 言語でプログラムを記述できるだけでなく，シミュレーション用に作成した関数をほとんどそのままリアルタイム制御に用いることができるので，制御系の実現が非常に容易になる。

3.26.2　マクロ置換

次のマクロ定義

```
  #define     YES    1
```

によって，その名前を文字列によって置換するという簡単な代入が行なわれる。

普通，定義行の終りまでが置換テキストであるが，“\” を行末に付けることによって，長い定義を次の行に続けることができる。#define で定義された名前の通用範囲は，定義した点からソ- スファイルの最後までである。

また，引数つきマクロの定義も可能である。これを使えば，テキストの置換はマクロの呼び出しに依存することになる。例えば，ABCD というマクロを定義したとする。

```
#define ABCD(S) S(1,Matrix),S(2,Matrix),S(3,Matrix),S(4,Matrix)
```

この ABCD はインライン展開され

```
S = tf2ss((s+1)/(s^2+3*s+4));
bodeplot(ABCD(S))
```

は次の行に換わる。

```
S = tf2ss((s+1)/(s^2+3*s+4));
bodeplot(S(1,Matrix),S(2,Matrix),S(3,Matrix),S(4,Matrix));
```

3.26.3　条件つきコンパイル

次のコンパイラ (インタプリタ) の制御文


```
sample.exe: sample.o  file1.o file2.o
  rtmatc sample.o file1.o file2.o

sample.o: sample.mm
  rtmatc -c sample.mm

file1.o: file1.mm
  rtmatc -c file1.mm

file2.o: file2.mm
  rtmatc -c file2.mm
```

D.2.2　リアルモード RTMATX(Borland C 版)

Borland C を用いる RTMATX のリアルタイム処理プログラムの作成手順は次の 3 段階からなる。

1. rtmatc で .mm ファイルから .c ファイルを作成
2. rtmatccc で .c ファイルから .obj ファイルを作成
3. rtmatcc で .obj ファイルから .exe ファイルを作成

MS-DOS のメモリ制約のため，rtmatc は C プリプロセッサを内部から呼び出せないので，#define や#include などのプリプロセス機能を使うには，手順 1 の前に，cpp で .mmm(ソース) ファイルから .mm ファイルを作成しておく。matc は Borland C 付属の cpp が出力するコードを扱えないので，RTMATX と一緒に配布されている (util/bin ディレクトリにある)GNU の cpp を使う。また，MS-DOS のメモリ制約のため，config.sys から必要性の低いドライバを削除しないと，rtmatc や cpp が起動できないことがある。

リアルタイム処理プログラムを作成する手順例を次に示す。

```
a = (3,4);
```

と記述すると

```
a = (3.00000000E+00,  4.00000000E+00)
```

となる。そして，複素行列を記述するための簡単な方法は少なくとも 2 つあって，一方は，

```
A = ([[1 2][3 4]], [[5 6][7 8]]);
```

であり，もう一方は

```
A = [[(1,5), (2,6)][(3,7), (4,8)]];
```

である。両者は同じ結果を導く。また，

```
x = $;
A = 3*x^2 + 4*x + 5;
B = 6*x^2 + 7*x + 8;
C = (A, B);
```

のように記述すると

```
C = (3,6) s^2 + (4,7) s + (5,8)
```

となる。複素値の実部と虚部は関数 Re() と Im() によって取り出すことができる。

3.25　コメント

“/*” と “*/” の間にある文章と “//” から行末までの文章は，インタプリタ (matx) とコンパイラ (matc) の両者に無視される。

D.4.2　リアルタイム処理関数

リアルタイム処理関数 on_task() の例を次に示す。

```
Func void on_task()
{
  Real r;
  Matrix yy, y, xf, u;

  // 目標値の計算
  r = m1*sin(w1*stime*t) + m2*sin(w2*stime*t);
  t++;

  yy = sensor();          // センサーからの入力

  y = [[  r  - yy(1)]     // 観測出力 1
       [y(1) - yy(2)]]; // 観測出力 2

  xf = A * x + B * y;     // コントローラの状態を更新
  u =  C * x + D * y;     // 制御入力
  x = xf;                 // コントローラの状態を保存

  actuator(u);            // アクチュエータへの出力
}
```

コメントの説明を補足する。

- プラントの出力が目標値に追従することを目的としているため，センサの出力から目標値を引いて観測出力としている。
- 関数 sensor() は，センサの測定値を A/D 変換器などで取り込んだ値を物理量に変換して返す。
- 関数 actuator() はアクチュエータの発生するトルクなどを操作する。この関数内部では D/A 変換器などを操作している。

```
s = String(複素数)          複素数の文字列化
s = String(多項式)          多項式の文字列化

i1 = Integer(文字列)        整数の文字列表現の整数化
i1 = Integer(実数)          実数に一番近い整数

r1 = Real(文字列)           実数の文字列表現の実数化
r1 = Real(複素数)           複素数の実部

c1 = Complex(実数)          実数が実部で，虚部が零の複素数

P1 = Polynomial(実数)       定数項が実数の実多項式
CP1 = Polynomial(複素数) 定数項が複素数の複素多項式
P1 = Polynomial(実行列)   行列の成分を昇べきの順に係数に持つ多項式，

                            次数はベクトルの次数-1

CP1 = CoPolynomial(実多項式) 引数を実部とする複素多項式

R1 = Rational(実数 r)       分子が Polynomial(r1)，分母が 1
CR1 = Rational(複素数 c) 分子が Polynomial(c1)，分母が 1
R1 = Rational(実多項式)   分子が実多項式，分母が 1
R1 = Rational(実行列 M1) 分子が Polynomial(M1)，分母が 1

CR1 = CoRational(実有理関数) 引数を実部とする複素有理関数

M1 = Matrix(配列)           配列と同じ成分を持つ行列
M1 = Matrix(多項式)         多項式の係数を昇べきの順に並べた横ベクト
ル
M1 = Matrix(実数 r)         (1,1) 成分が r の 1x1 実行列
CM1 = Matrix(複素数 c)    (1,1) 成分が c の 1x1 複素行列
```



```
Func void actuator_init()
{
  void actuator_stop();

  apd = amd = fsp = fsm = fcp = fcm = Z(DA_CH, 1);

  // 変換係数の設定
  fsp(1) = 6.843;
  fcp(1) = 6.843/3;
  fsm(1) = 8.026;
  fcm(1) = 8.026/3;
  apd(1) = 8.355 ;
  amd(1) = 9.394 ;

  fsp(2) = 1.967 ;
  fcp(2) = 1.967/3;
  fsm(2) = 2.089 ;
  fcm(2) = 2.089/3;
  apd(2) = 1.050 ;
  amd(2) = 1.071 ;

  actuator_stop(); // アクチュエータ停止
}
```

アクチュエータを停止する関数の例を次に示す。

```
Func void actuator_stop()
{
  void da_zero();

  da_zero(); // D/A 出力を 0 にする
}
```

センサの出力を読み込む関数の例を示す。

3.13　時間の計測

関数 settimer() でインターバルタイマの値をリセットし，関数 gettimer() を呼び出すと，その間の時間 (秒数) を 1 ms 単位で計測できる。UNIX ではプロセスの CPU 時間が，Windows と DOS では実行時間が計測される。

```
settimer();
for (i = 1; i <= 100; i++) {
  A = rand(100);
}
print gettimer();
```

3.14　現在の日付と時間

関数 clock() は現在の日付と時間を含む 6 成分からなる行ベクトル返す。

```
[year month day hour minute seconds]
```

```
date_time = clock();
```

最初の 5 個の成分は整数であり，最後の seconds は正確な実数である。

3.15　警告

警告メッセージを表示するには，warning() 関数を使用する。メッセージが標準エラー出力に出力される。長さが 0 の文字列を引数にして warning() を呼ぶと，警告は無視される。この関数には printf() と同じ書式を指定できる。

D/A 変換 (CONTEC) の出力関数の例を次に示す。

```
Func void da_conv(da_port, da_data)
  Index da_port;
  Matrix da_data;
{
  Integer i, data_i;
  Real data;

  for (i = 1; i <= length(da_port); i++) {
    data = da_data(i);
    if (data >= 10.0) {
      data = 9.9951;
    } else if (data <= -10.0) {
      data = -10.0;
    }

    data_i = Integer(4096.0 * (data + 10.0) / 20.0);
    if (rtIsTesting() == 0) {
      Outport(PORTNO_da, data_i*0x10 + da_port(i)-1);
    }
  }
}
```

D/A 変換器へ 0[v] に設定する関数の例を次に示す。

```
printf("%d %lf %s", 10, 3.1415, "Hello")

{a, b, c} = scanf("%d %lf %s")
print a, b, c
```

3.4　登録関数の表示 (matx のみ)

登録されている組み込み関数とユーザ定義関数を表示するには what コマンドを使用する。

```
what
```

また，which コマンドを使えば，どのファイルで定義されたユーザ定義関数が登録されているか分かる。変数を指定しないで which コマンドを使うと，登録されている全ての関数の情報が表示される。

```
which
which bode, ss2tfm
```

which コマンドで表示されるファイルは，require 修飾子[1]付きの関数宣言で変更できる。

3.5　登録変数の表示 (matx のみ)

登録されている変数を表示するには who コマンドを使用する。

```
who
```

[1] 8.1.1 節参照。


```
    Integer value;              を出力する。

void Outportb(portid, value); portid で指定したハードウェア
    Integer portid;             ポートに value が指定する 1 バイト
    Integer value;              を出力する。
```

D.7　シミュレーションからリアルタイム処理へ

ここでは，図 D.1 に示す倒立振子のオブザーバ+状態フィードバックによる制御について，シミュレーションおよびリアルタイム制御のプログラムの例を示す。

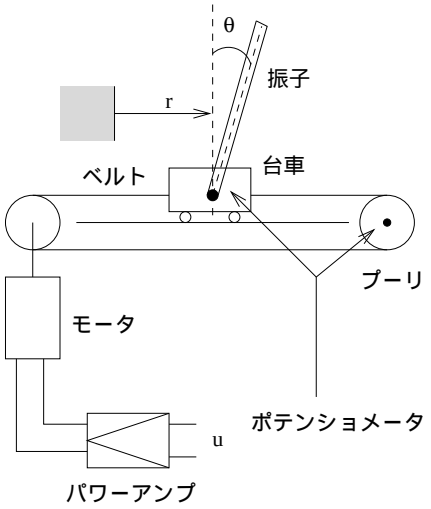


図 D.1: 倒立振子制御系の実験装置

台車についての運動方程式は

$$M\ddot{r} = au - F\dot{r} - H$$

である。ただし

2.8　C 言語インターフェース

C 言語で書かれたプログラムを matc が生成したコードにリンクすることは，とても簡単である。C 言語でプログラミングする時に守らなければならない規則を以下に示す。ライブラリ関数の詳しい説明は，別冊のライブラリ・リファレンス・マニュアルを参照すること。

1. ファイルの先頭で，使用するクラスのヘッダーファイルを include する。関数 main() の最初で，行列クラス，多項式クラス，有理関数クラス，リストクラスの初期化関数 MatInit(), PolyInit(), RatInit(), ListInit() を呼び出す。その後，広域変数の行列，多項式，有理関数，リストを初期化する。

```
#include <matrix.h>
#include <polynomial.h>
#include <rational.h>
#include <list.h>

Matrix a, b;     // 広域行列変数
Polnomial c, d; // 広域多項式変数
Rational e, f;  // 広域有理関数変数
List g, h;       // 広域リスト変数

void main(argc, argv)
  int  argc;
  char *argv[];
{
  // 各クラスの初期化
  MatInit(); PolyInit();
  RatInit(), ListInit();

  // 広域変数の初期化
  a = MAT_DEF,  b = MAT_DEF;
  c = POLY_DEF, d = POLY_DEF;
  e = RAT_DEF,  f = RAT_DEF;
  g = LIST_DEF, h = LIST_DEF;
  ......................
}
```


次に，状態方程式を記述する関数 diff_eqs() を示す。

```
Func void diff_eqs(DX, t, X, U)
  Matrix DX, X, U;
  Real t;
{
  Real x1, x2, x3, x4, u, c2, s2, dm;

  x1 = X(1); // 台車の位置
  x2 = X(2); // 振子の角度
  x3 = X(3); // 台車の速度 (推定値)
  x4 = X(4); // 振子の角速度 (推定値)
  u = U(1);  // 入力

  c2 = cos(x2); s2 = sin(x2);
  dm = (1 + alpha*s2^2);

  DX = Z(4,1);  // 状態方程式
  DX(1) = X(3);
  DX(2) = X(4);
  DX(3) = (a32*s2*c2 + a33*x3 + a34*c2*x4
           + a35*s2*x4^2 + b3*u)/dm;
  DX(4) = (a42*s2 + a43*c2*x3 + a44*x4
           + a45*s2*c2*x4^2 + b4*c2*u)/dm;
}
```

そして，コントローラを記述する関数 link_eqs() を示す。

```
Func void link_eqs(U, t, X)
 Matrix U, X;
 Real t;
{
 Matrix Y, Xh;

 Y  = C * X;                // 出力方程式
 Xh = Ch*Xo + Dh*Y;         // オブザーバの推定
 U  = - F*Xh;               // 状態フィードバック
 Xo = Ah*Xo + Bh*Y + Jh*U; // オブザーバの更新
}
```

中にエラーが発見されると，エラーメッセージが画面に表示され，実行が中止される。詳しいエラー情報がログファイル MaTX.log に出力される。
[例] 4 つのソースファイルから実行可能型ファイル main を作成する。

```
% emacs main.mm sub1.mm sub2.mm sub3.mm
% matc main.mm sub1.mm sub2.mm sub3.mm
% main
```

2.4　matc のオプション

matc のオプションを次の表に示す。

表 2.2: matc のオプション

オプション	機 能
-mm	各ファイルに対し C コンパイルを行わず
	c ファイルを生成する
-c	各ファイルに対しリンクを行わず
	o ファイルを生成する
-v	バージョンと C コンパイラ呼び出し情報を表示する
-o prog	実行可能ファイルの名前を prog とする
-static	スタティックリンクを行う*
-nocpp	C プリプロセッサによる処理を行なわない
-nomm	mm 以外の拡張子をもつファイルをコンパイルする
-Darg	arg を C プリプロセッサに渡す
-Ncdddd	内部バッファの大きさを dddd バイトとする
-Npdddd	プログラムコードのサイズを dddd ステップとする

*　: このオプションは SunOS, Solaris でのみ有効。

E.1.1　コンパイル時のメッセージの説明

Attempt to evaluate non-variable
関数 eval() に不適切な引数が渡された。

Arguments must be an integer or an index
ブロック行列の指定に整数，指数，以外のものが使われた。

Assignment to non-variable
定数への代入が行われた。

Builtin function is redefined
組み込み関数が再定義された。

Can't be transformed to a string
文字列へ変換できないものを文字列に変換しようとした。

Can't be transformed to an integer
整数に変換できないものを整数に変換しようとした。

Can't be transformed to a real number
実数に変換できないものを実数に変換しようとした。

Can't be transformed to a complex number
複素数に変換できないものを複素数に変換しようとした。

Can't be transformed to a polynomial
多項式に変換できないものを多項式に変換使用とした。

Can't be transformed to a complex polynomial
複素多項式に変換できないものを複素多項式に変換使用とした。

Can't be transformed to a rational
有理関数に変換できないものを有理関数に変換しようとした。

そのプログラムを実行できる。ファイルの最終行に実行したい関数名を書く代りに，“-e 'funcname()'” のようにコマンドラインに書ける。なお，オプション -e はコマンドラインの任意の場所で使える。

[例] 最後のソースファイル sub3.mm の最終行に “main();” を追加する代りに，コマンドラインに “-e 'main()'” と書く。

```
% matx main.mm sub1.mm sub2.mm sub3.mm -e 'main()'
```

3. matx は，引数に指定された全ソースファイルを解釈実行すると，すぐに終了するが，引数の最後に “-” があると，全ソースファイルを解釈した後，インタラクティブ・モードになり，プロンプトを表示する。つまり，インタラクティブ・モードになった後で実行したい関数名を “funcname();” と入力すれば，その関数を実行することができる。

   [例] 引数の最後の “-” を付けて実行し，インタラクティブ・モードで起動した後，“main();” と入力する。

```
% matx main.mm sub1.mm sub2.mm sub3.mm -
MaTX (1) main();
```

4. matx を引数なしで実行すると，インタラクティブ・モードで起動されプロンプトを表示するので，load 命令で mm ファイルを読み込んで，関数を実行することができる。(この時，ファイルの拡張子を省略すると mm となる。)

   [例] あらかじめ，ソースファイル sub3.mm の最終行に “main();” を追加しておく。matx を引数なしで実行し，インタラクティブ・モードで起動して，load 命令でソースファイルを読み込む。このとき，sub3.mm は，最後に読み込む。

Not suitable arguments class in pause()
関数 pause() に整数以外のものが渡された。

Not suitable declaration
関数の宣言が適切でない。

Number cann't be read from a file
行列以外のデータを”<<”でファイルから読み込もうとした。

Number cann't be written to a file
行列以外のデータを”>>”でファイルに保存しようとした。

Number must be zero
行列の正定性，負定性を調べるとき，ゼロ以外の値が使われた。

Only number can be used
定義されていない否定演算が行われた。

Redeclaration of function
関数の再宣言が前と同一でない。

Redeclaration of function (Argument)
可変長引数の関数の再宣言が前と同一でない。

Right-hand side must be a list
部分リストへの代入にリスト以外のものが使われた。

Right-hand side must be a string
部分文字列への代入に文字列以外のものが使われた。

Salars and matrices are used in a matrix
行列とスカラが同時に，行列の定義内で使われた。

String and values are used in a matrix
文字列とそれ以外のものが同時に，行列の定義内で使われた。

た後は，C コンパイラによってコンパイルし，実行ライブラリ (MaTX-Lib) とリンクして実行プログラム (test) を作成する。最後に，実行ファイルを実行して全ての仕様が満たされているかチェックする。

1.4　MATX の動作環境

MATX は以下のコンピュータとオペレーティングシステムで動作する。

Function returns no value
　関数が値を返していない。

Incorrect number of arguments
　関数の引数の数が定義と異なる。

Name too long
　変数名前の長さが長すぎる。

No help file is available
　ヘルプファイルが見つからない。

switch too deep
　switch 文の入れ子が深すぎる。

Switch label too much
　switch 文のラベルが多すぎる。

Syntax error was found in string evaluation
　関数 eval() による文字列の評価において文法エラーが検出された。

とができ，計算速度が上がり，メモリ効率を良く使用できる。

1.2　MATX の構造

MATX の構造を図 1.1 に示す。現在，MATX は 3 つユーザ・インターフェース・ツール (インタプリタ matx，コンパイラ matc，行列エディタ mated) を持っている。これらのツールは，クラスライブラリ **MaTX-Lib** を使って作成されている。MaTX-Lib は，複素数クラス，多項式クラス，有理多項式クラス，行列クラス，リストクラスの C 言語のクラス・ライブラリの集合である。

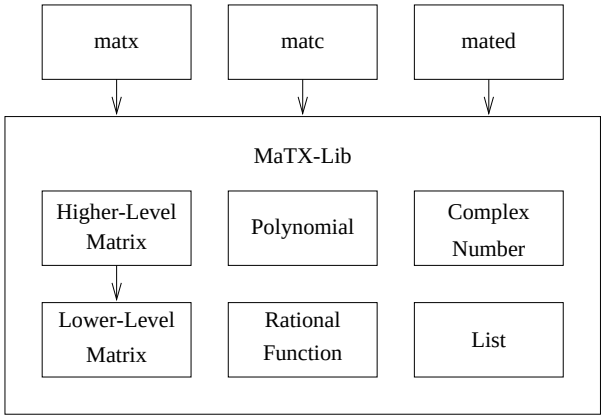


図 1.1: MATX の構造

特に，行列クラス・ライブラリは，高レベル・ライブラリと低レベル・ライブラリから構成される。低レベル・ライブラリは，約 300 個の関数から構成される。データの高速アクセスを目的としているので，データのチェック等を行なわないため，使用には注意が必要である。データの演算とメモリ効率を高めるための最適化もこのライブラリで行なわれている。高レベル・ライブラリは，低レベル・ライブラリの上に作られ，約 200 個の関数から構成される。高レベル・ライブラリの目的は，データの抽象化を行なうユーザ・イ

実数の—, 120
配列の—, 205
複素数の—, 126
access(), 68
acos()
実数の—, 120
配列の—, 205
複素数の—, 126
acosh()
実数の—, 120
配列の—, 205
複素数の—, 126
actuator(), 281, 284
actuator_init(), 283, 284
actuator_stop(), 283, 284
all(), 177
配列の—, 200
all_col(), 177
配列の—, 200
all_row(), 177
配列の—, 200
ans, 51
any(), 177
配列の—, 200
any_col(), 177
配列の—, 200
any_row(), 177
配列の—, 200
appendlist(), 209, 210
arg()
実数の—, 120
配列の—, 205
複素数の—, 126
Array(), 47, 49, 147
ASCII コード, 132
asin()
実数の—, 120
配列の—, 205
複素数の—, 126
asinh()
実数の—, 120
配列の—, 205
複素数の—, 126
atan()
実数の—, 120
配列の—, 205
複素数の—, 126
atan2()
実数の—, 120
atanh()
実数の—, 120
配列の—, 205
複素数の—, 126

bell, 43
Big Endian, 65
bit_and(), 119
bit_comp(), 119
bit_lshift(), 119
bit_or(), 119
bit_rshift(), 119
bit_xor(), 119

B.10 行列のメモリ管理 (9) 254
B.11 行列のメモリ管理 (10) 254
B.12 行列のメモリ管理 (11) 255
B.13 行列のメモリ管理 (12) 255

D.1 倒立振子制御系の実験装置 293

gplot_clear(), 85
gplot_figcode(), 88
gplot_grid(), 86
gplot_hold(), 88
gplot_key(), 86
gplot_loglog(), 81
gplot_options(), 86
gplot_psout(), 88
gplot_quit(), 85
gplot_reset(), 85
gplot_semilogx(), 81
gplot_semilogy(), 81
gplot_text(), 86
gplot_title(), 86
gplot_xlabel(), 86
gplot_ylabel(), 86
greplot(), 80
greplot_loglog(), 84
greplot_semilogx(), 84
greplot_semilogy(), 84

help, 45
hess(), 198
higher()
    多項式行列の—, 159
    多項式の—, 138
    有理多項式行列の—, 165
    有理多項式の—, 145
HOME, 57, 58
hterm, 90

I(), 151
if, 113
ifft(), 205
Im(), 52
    複素行列, 154
    複素数の—, 125
    複素多項式行列の—, 156
    複素多項式の—, 136
    複素有理多項式行列の—, 161
    有理多項式の—, 142
Index(), 49, 147
Inf, 117
infnorm(), 182
Inport(), 291
Inportb(), 291
int, 63
int2str(), 132
int2str(), 119, 133
Integer(), 48
integral()
    多項式行列の—, 159
    多項式の—, 138
inv()
    行列の—, 187
    実数の—, 120
    整数の—, 119
    多項式の—, 139
    配列の—, 205
    複素数の—, 126
    有理多項式の—, 145

実数の—, 122
整数の—, 119
文字列の—, 133
prod(), 184, 185
配列の—, 201
prod_col()
行列の—, 185
配列の—, 202
prod_row()
行列の—, 185
配列の—, 202
pseudoinv(), 186
putenv(), 42

qr(), 198
qr_p(), 198
QR 分解, 182, 198
QR 分解 (置換付), 198
quit, 46
qz(), 198
QZ 分解, 182, 198

RaArray, 147
RaArray(), 49
RaMatrix, 147
RaMatrix(), 49
rand()
行列の—, 190
実数の—, 120
配列の—, 203
randn()
行列の—, 190
実数の—, 120
配列の—, 203
rank(), 182
rat_last, 37
RatInit(), 36
Rational(), 48
Re(), 52, 212
複素行列, 154
複素数の—, 125
複素多項式行列の—, 156
複素多項式の—, 136
複素有理多項式行列の—, 161
有理多項式の—, 142
read, 39, 62, 63
Real(), 48, 120
rem()
実数の—, 120
整数の—, 119
配列の—, 203
require, 40, 104, 109
reshape(), 172
行列の—, 187
return, 105, 110
RKF45, 182, 193
rkf45(), 193
rngkut4(), 190, 223
roots(), 139
rotateDown()

14.6 有理多項式の微分 144
14.7 有理多項式の係数のシフト 145
14.8 有理多項式の比較 145
14.9 有理多項式関数 145

第 15 章 行列と配列と指数 **147**
15.1 行列の記述方法 148
15.1.1 行列成分の変換規則 148
15.1.2 行ベクトル 149
15.1.3 列ベクトル 150
15.1.4 行列 150
15.2 いろいろな行列 150
15.2.1 零行列 151
15.2.2 単位行列 151
15.2.3 1 で満たされた行列 152
15.2.4 等差数列配列 152
15.2.5 対角行列とブロック対角行列 153
15.2.6 複素行列 153
15.2.7 行列を成分とする行列 154
15.3 多項式行列 154
15.3.1 多項式行列の記述方法 154
15.3.2 多項式行列の評価 156
15.3.3 多項式行列の微分と積分 159
15.3.4 多項式行列の各要素の係数のシフト 159
15.4 有理多項式行列 160
15.4.1 有理多項式行列の記述方法 160
15.4.2 有理多項式行列の評価 162
15.4.3 有理多項式行列の微分 165
15.4.4 有理多項式行列の各要素の係数のシフト 165
15.4.5 有理多項式配列の分子配列と分母配列 166
15.5 行列成分の操作 166

X ウィンドウ, 93

yacc, 19

Z(), 151
zeros(), 145

アクチュエータ, 274, 281
アセンブラ, 273
値渡し, 110
暗黙の型変換, 100

一様乱数行列, 190
一般固有値, 186
一般固有値問題, 197
一般固有ベクトル, 186
インクリメント, 268
インターバルタイマ, 275
インタプリタ, 19
インタラクティブ・モード, 28

ウィンドウ・マネ- ジャ- , 93

エディタ, 23
エラー停止, 45
エラーメッセージ, 299
演算, 257
エンディアン, 63

大きさの自動拡大, 170
遅れ, 197
オブザーバ, 293

カーネル, 186
階乗, 119, 120
外積, 186
カウント・クロック, 275
片対数グラフ, 81, 93
型変換関数, 132
可変長引数, 107
画面クリア, 43
カレントウィンドウ, 78
環境変数, 53, 57, 58, 61, 77, 90
関係演算, 269
関数, 103
関数の参照, 111
関数の引数, 109
関数のポインタ, 111

キーバインディング, 33, 74
擬似逆行列, 186
逆, 263
逆行列, 187
逆高速フーリエ変換, 205
逆フーリエ変換, 205
行数, 182
行ベクトル, 149
共役行列, 187
共役転置, 267
共役転置行列, 187
行列, 147
行列エディタ, 18, 34, 39, 71
行列関数, 182
行列式, 182

3.7 変数の存在を調べる (matx のみ) . . . 41
3.8 環境変数の設定と取得 . . . 42
3.9 メニューを表示する . . . 42
3.10 OS のコマンドを実行する . . . 42
3.11 画面のクリア (matx のみ) . . . 43
3.12 ベル . . . 43
3.13 時間の計測 . . . 44
3.14 現在の日付と時間 . . . 44
3.15 警告 . . . 44
3.16 エラー停止 . . . 45
3.17 ヘルプ . . . 45
3.18 停止 . . . 46
3.19 終了 . . . 46
3.20 クラス変換 . . . 47
3.21 複数行に渡る式 (コマンドラインのみ) . . . 50
3.22 代入式 . . . 50
3.23 式の値を保存する変数 ans . . . 51
3.24 複素値 . . . 51
3.25 コメント . . . 52
3.26 プリプロセッサ . . . 53
3.26.1 ファイルの読み込み . . . 55
3.26.2 マクロ置換 . . . 56
3.26.3 条件つきコンパイル . . . 56
3.27 スタートアップファイル . . . 57
3.28 クイットファイル (matx のみ) . . . 58
3.29 バグ . . . 59

第 4 章 ファイル **61**
4.1 mm ファイル の読み込み (matx のみ) . . . 61
4.2 データのファイル入出力 . . . 62
4.2.1 行列データのファイル入出力 . . . 62

右特異ベクトル, 186

無駄時間, 197

メニュー, 42
メモリ管理, 290
メモリ管理リスト, 290

目標値, 274
文字の位置, 130
文字列, 129
モデリング, 273

ユークリッド ノルム, 182
優先度, 257
有理多項式, 141
有理多項式行列, 160

ランク, 182

リアルタイム, 273
リアルタイム MATX, 273
リアルタイム制御, 293
リアルモード, 276
リカッティ, 212, 214
リスト, 106, 207
リストの合成, 209
リストの取り出し, 209
リダイレクト
->, 63
<-, 63
<<, 62
>>, 62

両対数グラフ, 81
リンク, 278

累乗, 187, 265
累積積, 184, 185, 201, 202
累積和, 184, 185, 201, 202
ループ, 114
ルンゲ・クッタ, 182, 190

零行列, 151
零空間, 186
列数, 182
列ベクトル, 150

ローテート, 187
論理演算, 271

和, 259
割り込み, 273
割り込みキー, 280