＜著者略歴＞

古賀 雅伸（こが まさのぶ）

1988 年　東京工業大学工学部制御工学科卒業
1993 年　博士 (工学)
1993 年　東京工業大学工学部助手
現　在　東京工業大学大学院情報理工学研究科助手

MATX リファレンスマニュアル

**1994 年 9 月 26 日**　初　版　発　行
**1995 年 8 月 10 日**　2　版　発　行
**1996 年 4 月 15 日**　3　版　発　行
**1997 年 4 月 21 日**　4　版　発　行
**1998 年 4 月 12 日**　5　版　発　行

執筆者　古賀 雅伸
発行者　古賀 雅伸
印刷者　東京工業大学
製本者
発行所　東京工業大学
〒 152-8552
東京都目黒区大岡山 2-12-1
電話 (03)5734-2328

本書の内容を無断で複写生成することは，著作者の権利を侵害することがありますので，その場合にはあらかじめ著作者あて許諾をお求め下さい。

# 目 次

**第 1 章　はじめに　17**
1.1 プログラミング言語 MATX . . . . . 18
1.2 MATX の構造 . . . . . 20
1.3 プログラム作成手順 . . . . . 22
1.4 MATX の動作環境 . . . . . 24

**第 2 章　プログラムの作り方とその実行方法　27**
2.1 matx を用いたプログラムの実行 . . . . . 27
2.2 matx のオプション . . . . . 30
2.3 matc を用いた実行可能ファイルの作り方 . . . . . 30
2.4 matc のオプション . . . . . 32
2.5 コマンドライン履歴・編集 . . . . . 33
2.6 mated を用いた行列の入力・編集 . . . . . 34
2.7 matp を用いた行列の表示 . . . . . 35
2.8 C 言語インターフェース . . . . . 36

**第 3 章　コマンドと便利な機能　39**
3.1 画面に表示 . . . . . 39
3.2 キーボードからの入力と編集 . . . . . 39
3.3 入出力 . . . . . 39
3.4 登録関数の表示 (matx のみ) . . . . . 40
3.5 登録変数の表示 (matx のみ) . . . . . 40
3.6 変数の消去 . . . . . 41

行列の成分, 166
行列の配列, 175
局所変数, 37, 97, 103
虚数単位, 126

クイットファイル, 58
クラス変換, 47, 258
グラフィックス, 77

警告, 44
係数のシフト
　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
結合規則, 257

広域変数, 36
合計, 184, 185, 201, 202
高速フーリエ変換, 205
コマンドライン引数, 111
コマンドライン編集, 33
コマンドライン履歴, 33
コメント, 52
固有値, 182, 186
固有値問題, 197
固有ベクトル, 186
コンパイラ, 19
コンパイル, 278

差, 260
最小値, 182
最小特異値, 182
最大値, 182
最大特異値, 182
最大ノルム, 182
参照渡し, 110
サンプリング時間, 274, 280

シェル, 42
シェルスクリプト, 29
式行列, 47, 168
指数, 147, 167, 173
指数行列, 186
実業列, 182
実行ファイル, 30
実数, 117
自動きざみ, 191
シフト, 187
シミュレーション, 220, 273
終了, 46
シュール分解, 198
商, 263
状態フィードバック, 220, 223, 293
常微分方程式, 182

数式処理, 17
数値計算, 17
スカラ, 182
スカラ倍, 261
スタートアップファイル, 57

制御系設計 CAD, 273
正規乱数行列, 190



10.4 実数関数 . . . . . 120

**第 11 章 複素数** **125**
11.1 複素数の記述方法 . . . . . 125
11.2 複素数の比較 . . . . . 126
11.3 複素数関数 . . . . . 126

**第 12 章 文字列** **129**
12.1 文字列の成分の取り出し . . . . . 129
12.2 文字列の成分の代入 . . . . . 129
12.3 文字列の比較 . . . . . 129
12.4 文字列から指定文字の位置を調べる . . . . . 130
12.5 文字列の評価 (matx) . . . . . 130
12.6 文字列への変換 . . . . . 132
12.7 文字列関数 . . . . . 133

**第 13 章 多項式** **135**
13.1 多項式の記述方法 . . . . . 135
13.2 複素多項式 . . . . . 136
13.3 多項式の係数の参照 . . . . . 136
13.4 多項式の評価 . . . . . 137
13.5 多項式の微分と積分 . . . . . 138
13.6 多項式の係数のシフト . . . . . 138
13.7 多項式の比較 . . . . . 138
13.8 多項式関数 . . . . . 139

**第 14 章 有理多項式** **141**
14.1 有理多項式の記述方法 . . . . . 141
14.2 複素有理多項式 . . . . . 141
14.3 有理多項式の分子多項式と分母多項式 . . . . . 142
14.4 有理多項式の次数の増加 . . . . . 143
14.5 有理多項式の評価 . . . . . 143

行列の—, 187
配列の—, 203
rotateLeft()
行列の—, 187
配列の—, 203
rotateRight()
行列の—, 187
配列の—, 203
rotateUp()
行列の—, 187
配列の—, 203
round()
行列の—, 187
多項式の—, 139
複素数の—, 126
有理多項式の—, 145
round()
実数の—, 120
配列の—, 205
round2z()
行列の—, 187
実数の—, 120
整数の—, 119
多項式の—, 139
複素数の—, 126
有理多項式の—, 145
round2z()
配列の—, 205
Rows()
行列の—, 182
配列の—, 200
rtIsRunning(), 291
rtIsTesting(), 290, 291
rtIsTimeOut(), 282, 291
rtmatc, 277
rtmatcc, 277
rtmatccc, 277
RTMATX, 273
rtSetBreak(), 280, 291
rtSetClock(), 275, 280, 291
rtSetTask(), 280, 291
rtStart(), 280, 290, 291
rtStartRehearsal(), 290, 291
rtStop(), 280, 291
rtStop, 283
rtStopRehearsal(), 290, 291

save, 65
scanf(), 39
実数の—, 122
整数の—, 119
文字列の—, 133
schur(), 198
sensor(), 281, 284
sensor_init(), 283, 284
settimer(), 44
sgn()
実数の—, 120
配列の—, 205
複素数の—, 126
shiftDown()





D.2.1 プロテクトモード RTMATX(DJGPP 版) . . . . . 276
D.2.2 リアルモード RTMATX(Borland C 版) . . . . . 277
D.3 動作環境 . . . . . 279
D.4 プログラムの動作 . . . . . 279
D.4.1 メイン関数 . . . . . 280
D.4.2 リアルタイム処理関数 . . . . . 281
D.4.3 画面表示 / コマンド入力ループ関数 . . . . . 282
D.4.4 実験装置のインターフェース関数 . . . . . 283
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 . . 284
D.4.6 入出力カード (D/A, カウンタ) のインターフェース関数287
D.5 メモリ管理 . . . . . 290
D.6 リアルタイム処理用関数 . . . . . 291
D.7 シミュレーションからリアルタイム処理へ . . . . . 293
D.7.1 倒立振子の制御のシミュレーション . . . . . 296
D.7.2 倒立振子のリアルタイム制御 . . . . . 298

**付 録 E エラーメッセージ** **299**
E.1 プログラムの解釈，コンパイル時のエラーメッセージ . . . . 299
E.1.1 コンパイル時のメッセージの説明 . . . . . 301
E.2 実行時のエラーメッセージ . . . . . 307
E.2.1 matx の実行時のエラーメッセージ . . . . . 307
E.2.2 matx の実行時のメッセージの説明 . . . . . 308
E.2.3 クラスライブラリのエラーメッセージ . . . . . 310

**索 引** **311**

iscomplex()
　　行列の—, 182
　　多項式の—, 139
　　配列の—, 200
　　有理多項式の—, 145
isnan()
　　行列の—, 182
　　実数の—, 120
　　配列の—, 200
　　複素数の—, 126
isreal(), 212
　　行列の—, 182
　　多項式の—, 139
　　配列の—, 200
　　有理多項式の—, 145
isscalar(), 182

kbhit(), 119
kermit, 90
kernel(), 186

LANG, 45
length(), 100, 210
　　行列の—, 182
　　配列の—, 200
　　文字列の—, 133
libs.rsp, 278
link_eqs(), 297
LINPACK, 21
linspace()
　　配列の—, 203
list_last, 37
ListInit(), 36
Little Endian, 65
load, 28, 61, 65
log()
　　行列の—, 186
　　実数の—, 120
　　配列の—, 205
　　複素数の—, 126
log10()
　　実数の—, 120
　　配列の—, 205
　　複素数の—, 126
logspace(), 219
　　配列の—, 203
lower()
　　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
lu(), 198
LU 分解, 182, 198

m-file, 23
machine_endian(), 65
main(), 30, 36, 97, 103
Makefile, 276, 278
makelist(), 208
mat, 62
matc, 20, 30
mated, 20, 34, 71, 74



# 図 目 次

1.1　MATX の構造 . . . . . 20
1.2　MATX の内部処理 . . . . . 21
1.3　自動型変換規則 . . . . . 22
1.4　Matlab によるプログラム作成手順 . . . . . 23
1.5　MATX によるプログラム作成手順 . . . . . 23

5.1　行列エディタ (実行列) . . . . . 72
5.2　行列エディタ (複素行列) . . . . . 73
5.3　行列エディタ (実部行列) . . . . . 73
5.4　行列エディタ (虚部行列) . . . . . 74

17.1 制御対象とコントローラ . . . . . 225
17.2 複雑なシステムのシミュレーション . . . . . 230

B.1　MATX のメモリ管理 . . . . . 249
B.2　行列のメモリ管理 (1) . . . . . 251
B.3　行列のメモリ管理 (2) . . . . . 252
B.4　行列のメモリ管理 (3) . . . . . 252
B.5　行列のメモリ管理 (4) . . . . . 252
B.6　行列のメモリ管理 (5) . . . . . 253
B.7　行列のメモリ管理 (6) . . . . . 253
B.8　行列のメモリ管理 (7) . . . . . 253
B.9　行列のメモリ管理 (8) . . . . . 254

Borland C, 278
break, 115, 116

case, 115
ceil()
　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
char, 63
chdir, 69
clear
　画面のクリア, 43
　変数の消去, 41
clock(), 44
CoArray, 147
CoArray(), 49
Cols()
　行列の—, 182
　配列の—, 200
CoMatrix, 147
CoMatrix(), 49
Complex(), 48
conj()
　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
conjtrans()
　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
continue, 116
CoPolynomial, 135
CoPolynomial(), 48
CoRational, 141
CoRational(), 48
cos()
　実数の—, 120
　配列の—, 205
　複素数の—, 126
cosh()
　実数の—, 120
　配列の—, 205
　複素数の—, 126
cpp, 277
cumprod()
　行列の—, 184
　配列の—, 201
cumprod_col()
　行列の—, 185
　配列の—, 202



を使うか関数を呼び出さなければならない。一方，MATX はデータの型を「区別した」言語である。データ型には，整数，実数，複素数，文字列，多項式，有理多項式，行列，配列，リストがあり，行列，配列，多項式，有理多項式，リストは次数指定を必要としない。

MATX は数値解析だけでなく数式処理も行なえ，方程式の解を求めたり，多項式や有理多項式の導関数や不定積分を計算できる。MATX の文法の根幹となる制御の流れや関数やプログラム構造は，C 言語から受け継がれている。C 言語は，PDP-11 上で Dennis M.Ritchie によって開発されたプログラミングである [7, 20, 8, 22]。MATX はすべて C 言語で記述されており，特に，構文解析部は構文解析部生成言語として有名な **yacc** を用いて作成されている [18, 21]。

Matlab は，インタプリタであるため Fortran や C で作成されたプログラムに比べて，複雑な計算の繰り返しなどに時間がかかる。そのため，Matlab は特別なインターフェースをもっていて，別にコンパイルしてリンクされた Fortran や C のルーチン (mex-files) を Matlab 環境から普通の m-file のように呼び出すことができる。m-file によるアルゴリズムの実現の速度が十分でない時に，この機能は有効である。しかしながら，mex-file の作成は簡単な作業ではなく，Matlab と Fortran の間で関数のパラメータを交換するための特別な gateway function を書かなければならない。

MATX は対話型処理を行なうインタプリタだけでなく，コンパイラも提供している。インタプリタを matx，コンパイラを matc と呼ぶ。パッケージ全体を表す時は MATX と書いてインタプリタ (matx) と区別する。コンパイラ (matc) は，インタプリタ (matx) と同じプログラムを処理することができ，与えられたプログラムから移植性の良い C 言語のコードを生成する。プログラムを拡張するためには，アルゴリズムを **mm** ファイルの中に関数として実現する。この過程は，matx を使って対話的に途中経過をチェックしながら行なえる。関数が完成した後，matc を使ってコンパイルし，実行プログラムを作ることができる。C 言語のファイルを matc の出力コードにリンクすることによって，C 言語の関数を MATX の関数から簡単に呼び出すことができる。このおかげで，既存の膨大な C 言語のルーティンを利用するこ

### E.2.3　クラスライブラリのエラーメッセージ

クラスライブラリの実行時のメッセージは，次の書式で表示される。

> ”sourcecode”：message in func1()
> Error was found in func1() from func2() from func3()

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，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。

- エラーの発生に関係するソースコードを推定して，できるだけオリジナルに近いかたち (sourcecode) で表示する。したがって，必ずしも元のソースコードと一致するとは限らない。
- 行列演算のエラーに関して，関係する行列の名前とその行列の大きさを表示する。ただし，名前が分からない場合，行列の大きさだけを表示する。
- 多項式演算のエラーに関して，関係する多項式の名前とその多項式の次数を表示する。ただし，名前が分からない場合は，多項式の次数だけを表示する。
- 有理関数演算のエラーに関して，関係する有理関数の名前とその有理関数の分子・分母の次数を表示する。ただし，名前が分からない場合，有理関数の分子・分母の次数だけを表示する。
- リスト演算のエラーに関して，関係するリストの名前とそのリストの成分の数を表示する。ただし，名前が分からない場合，リストの成分の数だけを表示する。
- 関数名 func1 は，行列演算では Mat***() ，多項式演算では Poly***() ，有理関数演算では Rat***() ，リスト演算では List***() となる。
- 関数が呼び出された順番を同時に表示する。上の書式の場合，func3() から func2() が呼び出され，func2() から func1() が呼び出され，func1() の中でエラーが検出されたことが分かる。



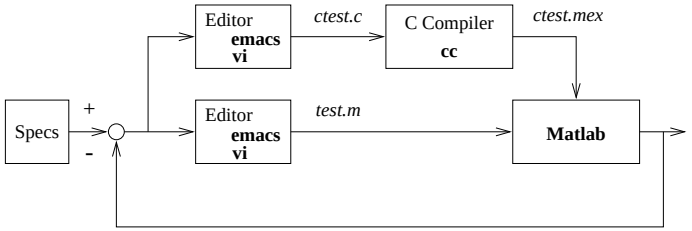



図 1.4: Matlab によるプログラム作成手順

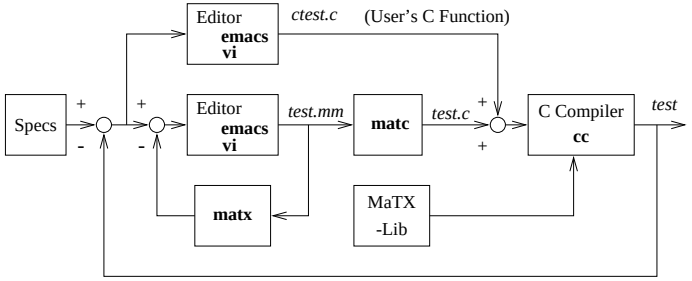



図 1.5: MATX によるプログラム作成手順

てアルゴリズムを m-file(test.m) の中に実現する。仕様を満たすまで m-file のテストと書き直しが繰り返される。C 言語や Fortran で書かれたファイル (ctest.c) をそれぞれの言語のコンパイラでコンパイルすれば (ctest.mex)，Matlab の関数から呼び出すことができる。

MATX でプログラムを作成するには，仕様にしたがってアルゴリズムをエディタ (vi, emacs) を使って mm ファイル (test.mm) の中に関数として実現する。インタープリタ (matx) を使えば，ファイル単位，関数単位，または行単位に会話的に mm ファイルをテストすることができる。動作確認の終った mm ファイルは，matc によって C 言語に変換される。C 言語のプログラム (ctest.c) を matc の出力コードにリンクすれば，MATX の関数から簡単に C 言語の関数を呼び出すことができる。全ての C 言語のファイルが完成し

**switch condition must be an integer**
　switch 文の条件に整数でないものが使用された。

**syntax error**
　文法的な間違いが検出されました。

**Too few arguments for ”print”**
　データを保存する print コマンドの引数の数が足りなかった。

**Too few arguments for ”save”**
　save コマンドの引数の数が足りなかった。

**Undefined Character ’chracter’**
　不適切な文字が使用されました。

**Undefined variable**
　宣言されていない変数が使われました。

**Used outside loop**
　break や continue がループの外で使われた。

**Variable may be used before set**
　初期化されていない変数が参照されました。

**Void function returns value**
　void 関数が値を返している。

**Void is the class of functions**
　void 型で変数が宣言された。

**Zero division**
　ゼロ割が検出された。



# 第2章　プログラムの作り方とその実行方法

プログラムをどのようにして作成して実行するかは，使うシステムに依存する。具体例として，UNIX オペレーティング・システムでのプログラムの作成方法と実行方法を示す。

## 2.1　matx を用いたプログラムの実行

インタプリタ matx を用いてプログラムを実行するには，少なくとも以下の 5 つの方法がある。

1. matx の引数にソースファイルを指定して実行すると，ソースファイルは指定された順番に解釈実行される。(この時，ソースファイルの拡張子 mm は省略できる。) すなわち，一番最後のファイルの最終行に実行したい関数名を “funcname();” のように追加すれば，その関数を実行することができる。

   [例] 最後のソースファイル sub3.mm の最終行に “main();” を追加して実行。

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

2. matx の引数にオプション-e に続けて MATX のプログラムを書くと，

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

**Can't be transformed to a matrix**
　行列に変換できないものを行列に変換しようとした。

**Can't be transformed to a complex matrix**
　複素行列に変換できないものを複素行列に変換しようとした。

**Can't be transformed to a polynomial matrix**
　多項式行列に変換できないものを多項式行列に変換しようとした。

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

**Can't be transformed to an array**
　配列に変換できないものを配列に変換しようとした。

**Can't be transformed to a complex array**
　複素配列に変換できないものを複素配列に変換しようとした。

**Can't be transformed to a polynomial array**
　複素配列に変換できないものを複素配列に変換しようとした。

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

**Can't be transformed to an index**
　指数に変換できないものを指数に変換しようとした。

**Can't be transformed to a list**
　リストに変換できないものをリストに変換しようとした。

**Can't compare these variables**
　定義されていない型による比較が行われた。



表 2.1: matx のオプション

| オプション | 機 能 |
|---|---|
| - | 標準入力 (キーボード) から入力する |
| -- | 標準入力から入力 (プロンプトなし) |
| -nocpp | C プリプロセッサによる処理を行なわない |
| -nolog | ログファイル MaTX.log を作らない |
| -nofep | コマンドライン編集機能を使わない |
| -nohist | 履歴機能を使わない |
| -e 'stmt' | stmt を実行する |
| -Darg | arg を C プリプロセッサに渡す |
| -Npdddd | プログラムコードのサイズを dddd ステップとする |

し，インタプリタ (matx) を使ってプログラムを実行するとき，プログラムのどこからでも実行を始めることができ，main() をプログラムの中に作る必要はない。

1. 拡張子 mm の MaTX ソースファイル mmtest.mm をエディタで作成する。
2. ソースファイルをコマンド "matc mmtest" でコンパイルし，実行可能型ファイル mmtest(Windows の場合，mmtest.exe) 作成する。
3. 作成した実行可能型ファイル mmtest (Windows の場合，mmtest.exe) の名前をコマンドラインで入力する。

ソースファイルをコマンドラインで指定するとき，拡張子 mm を省略できる。オプション -nomm を指定すると，mm 以外の拡張子をもつファイルをコンパイルできる。実行可能型ファイルの名前は，引数として与えられた先頭のファイルのファイル名から拡張子を除いたものとなる。実行可能型ファイルの名前は "-o prog" のように指定できる。matc 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 "matc (1)" が表示される。このモードは，プログラムのデバッグに便利である。プログラムの実行

### D.7.2　倒立振子のリアルタイム制御

この章の前で説明したように，リアルタイム処理のプログラムを作成するには，3 つの関数を記述するだけで良い。メイン関数と画面表示関数はは例題で示した通りで良い。リアルタイム処理関数はシミュレーションで用いた link_eqs() を次のように変更するだけで良い。

```
Func void on_task()
{
  Matrix Y, Xh;

  Y  = sensor();             // センサから入力
  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
  actuator(U);               // アクチュエータへ出力
  Xo = Ah*Xo + Bh*Y + Jh*U; // オブザーバの更新
}
```



```
% mated A.mat B.mat C.mat
```

## 2.7　matp を用いた行列の表示

[1]プログラム matp を使えば，MAT データフォー マットでファイルに保存した行列を表示することができる．

```
% matp A.mat B.mat C.mat
```

[1]付録参照

$M$ [kg] : 台車・プーリ・モータ系の等価質量
$r$ [m] : 台車の変位
$u$ [V] : パワーアンプへの入力電圧
$a$ [N/V] : $u$ から台車に働く力までのゲイン
$F$ [N/(m/s)] : 台車・ベルト・プーリ・モータ系の等価摩擦係数
$H$ [N] : 振子から台車に働く水平力

である。振子についての運動方程式は

$$\begin{aligned} J\ddot{\theta} &= Vl\sin\theta - Hl\cos\theta - c\dot{\theta} \\ & m\tfrac{d^2}{dt^2}(r + l\sin\theta) = H \\ & m\tfrac{d^2}{dt^2}(l\cos\theta) = V - mg \end{aligned}$$

である。ただし，

$J$ [kgm$^2$] : 振子の重心回りの慣性モーメント
$\theta$ [rad] : 振子の鉛直線下からの角度
$V$ [N] : 台車から受ける力の垂直方向成分
$l$ [m] : 軸から振子の重心までの距離
$H$ [N] : 台車から受ける力の水平方向成分
$c$ [kgm$^2$/s] : 軸の摩擦係数
$m$ [kg] : 振子の質量
$g$ [m/s$^2$] : 重力加速度

である。状態変数として，台車の変位 $r$，振子の角度$\theta$，台車の速度$\dot{r}$，振子の角速度$\dot{\theta}$をとり，状態ベクトルを

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} r \\ \theta \\ \dot{r} \\ \dot{\theta} \end{bmatrix}$$



# 第3章　コマンドと便利な機能

## 3.1　画面に表示

値を表示するには，クラスの区別なく print コマンドを使用する。コンマで区切ることにより，複数の値を同時に指定できる。

```
print 1, 1.0, (1,2), "Hello", [[1 2][3 4]]
```

## 3.2　キーボードからの入力と編集

値を入力・編集するには，クラスの区別なく read コマンドを使用する。コンマで区切ることにより，複数の値を同時に指定できる。行列を入力・編集する時は，行列エディタが起動される。

```
read a, b, A, B
```

## 3.3　入出力

C 言語と同様にデータを画面に書式を指定して出力 (printf()) したりデータをキーボードから読み込む (scanf()) ことができる。 入出力の例を次に示す。

```
Func void da_zero()
{
  Integer i;
  Index da_port;
  Matrix da_data;

  void da_conv();

  da_port = Z(DA_CH, 1);
  da_data = Z(DA_CH, 1);
  for (i = 1; i <= DA_CH; i++) {
    da_port(i) = i;
    da_data(i) = 0.0;
  }
  da_conv(da_port, da_data);
}
```

## D.5　メモリ管理

リアルタイム処理を実現にはメモリ管理に工夫が必要である。なぜなら，リアルタイム処理は高速に行なわれなければならないだけでなく，割り込み処理中はメモリ確保するためのシステムコールを呼び出せないからである。

MATX のメモリ管理機構は，この問題を解決するのに適している。その方法は，リアルタイム処理を始める前に一度だけ，リアルタイム処理関数を呼び出すことによって，リアルタイム処理中に必要なメモリをメモリ管理リストに登録するのである。

具体的には，関数 rtStart() が実行されると，サンプリング時間毎の割り込み処理が始まる前に 1 回だけ on_task() が実行される。この 1 回目の実行中かどうかは関数 rtIsTesting() によって調べることができる。このテスト中はアクチュエータから出力を出さないようにしなければならない。

関数 on_task() に条件分岐あると，1 回のテスト実行だけでは on_task() の実行に必要なメモリを正確に調べることができない。このような場合，全ての条件に関するテスト実行を



表 3.1: メニューのキーバインディング

| 機能 | キー |
|---|---|
| 下移動 | ^N または j または ↓ |
| 上移動 | ^P または k または ↑ |
| i 番のメニュー | 数字 i (0 ~ 9) |
| 10 以上の番号を選択 | +を 1 回で 10 |
| | +を 2 回で 20 |
| 決定 | リターン |

```
system("xplot -silent A.mat");
 ! ls *.c
 ! pwd
```

## 3.11　画面のクリア (matx のみ)

画面をクリアするには，clear コマンド[3]を使用する。

```
clear;
```

## 3.12　ベル

関数 bell() を呼ぶと，ベルが鳴る。

```
bell();
```

[3] clear コマンドは変数の消去にも使用する。

```
Func Matrix sensor()
{
  Integer i;
  Matrix cnt_data;

  Integer count_get(); // カウンタ読み込み関数

  cnt_data = Z(CNT_CH,1);
  for (i = 1; i <= CNT_CH; i++){
    cnt_data(i) = c_k(i) * count_get(i);
  }

  return cnt_data;
}
```

アクチュエータへ出力する関数の例を示す。

```
Func void actuator(data)
  Matrix data;
{
  Integer i;
  Index da_port;
  Matrix v;

  void da_conv(); // D/A 出力関数

  v = Z(DA_CH, 1);
  for (i = 1; i <= DA_CH; i++) {
    if (data(i) >= 0.0) {
      v(i) = data(i) / apd(i);
    } else {
      v(i) = data(i) / amd(i);
    }
  }

  da_port = [1 2];
  da_conv(da_port, v);
}
```



## 3.20　クラス変換

型の異なるデータの演算においては，自動型変換が行なわれるが，いくつかの演算は，明示的な型変換を必要とする。例えば，

```
A = ([1 2 3] * [4 5 6]') * [[1 2][3 4]];
```

で，([1 2 3] * [4 5 6]') は $1\times1$ の実行列であるので，次の行列との乗算はエラーとなる。$1\times1$ の実行列から (1,1) 成分を取り出すには式をブラケット [ と ] で囲み変数行列と同様に扱えるようにする 15.5.3。すなわち

```
A = [[1 2 3] * [4 5 6]'](1) * [[1 2][3 4]];
```

のように変更する。

```
A = [[1 2][3 4]];  B = [[5 6][7 8]];
C = A .* B;
D = Array(A) * Array(B);
```

の結果 C と D は一致する。

```
A = [[1 2][3 4]];
for (i = 1; i <= 10; i++) {
  print A * i >> "A-" + String(i) + ".mat";
}
```

で，関数 String() は，整数を文字列に変換する。
クラス変換関数を用いてクラスを変換することができる。クラス名とそのクラスへの変換関数の名前は一致している。以下にクラス変換関数の一覧を示す。

| | |
|---|---|
| s = String(整数) | 整数の文字列化 |
| s = String(実数) | 実数の文字列化 |

### D.4.3　画面表示 / コマンド 入力ループ関数

画面表示 / コマンド 入力ループ関数 off_task_loop() の例を次に示す。

```
Func void off_task_loop()
{
  Integer end_flag;

  end_flag = 0;
  do {
    if (rtIsTimeOut()) {  // タイムアウトのチェック
      warning("Time Out\n");
      break;
    }

    gotoxy(5, 11);          // 観測出力の表示
    printf("y1 = %f[m] y2 = %f[rad]", y(1), y(2));

    if (kbhit()) {          // コマンド入力
      switch (getch()) {
        case 0x1b:          // ESC キーが押されると
          end_flag = 1;     // ループが終了する
          break;
        default:
          break;
      }
    }
  } while (end_flag == 0);
}
```

- 関数 rtIsTimeOut() はサンプリング時間内にリアルタイム処理が終了したかどうかを調べる。処理が終了しなかった場合は，警告を出力して関数を終了する。
- ESC キーが押されるとループが終了し，関数も終了する。
- 関数 gotoxy() はプロンプトを移動する。
- 関数 kbhit() はキーボードが押されたかどうか調べる。
- 関数 getch() は押されたキーの値を読み込む
- gotoxy() , kbhit() , getch() は C コンパイラ付属の関数である。



```
Matrix A, B, C;

A = B = C = Z(2,3);    // 式
```

は正しいが，

```
Matrix A, B, C;
Polynomial D;
List LL;

LL = {1, (1,2), [1 2]};   // 文
{A, B, C} = {A, B} = LL;  // 文
```

は正しくない。

## 3.23　式の値を保存する変数 ans

式の評価結果が変数に代入されない場合，その値は変数 ans に保存される。以下に例を示す。

```
MaTX (1) 1 + 2 + 3
ans = 6
MaTX (2) ans*ans
ans = 36
MaTX (3) sqrt(ans)
ans = 6
```

## 3.24　複素値

MaTX の全ての演算や関数の記述において，複素数値を実値と区別なく使うことができる。複素値は，一対の実値をコンマで区切って丸括弧（ と ）で囲むことによって簡単に記述することができる。たとえば，

```
A:> cpp sample.mmm > sample.mm
A:> rtmatc sample.mm
A:> rtmatccc sample.c
A:> rtmatcc sample
```

コンパイル，リンクを行なうコマンド rtmatccc や rtmatcc はバッチファイルであり，Borland C を使う場合を次に示す。

```
A:> type rtmatccc.bat

bcc @\rtmatx\bin\matx.rsp %1 %2 %3 %4 %5 %6 %7 %8 %9

A:> type rtmatcc.bat

tlink /c /s c0h %1 %2 %3 %4 %5,%1,%1,@\rtmatx\bin\libs.rsp
```

また，matx.rsp や libs.rsp にはコンパイルスイッチとリンクスイッチが記述されている。

```
A:> type matx.rsp

-c -mh -w-pro -w-par* -DMSDOS -DBRC20 -I. -I\rtmatx\include -DLST

A:> type libs.rsp

c:\rtmatx\lib\libmatx +
c:\borlandc\lib\emu +
c:\borlandc\lib\mathh +
c:\borlandc\lib\ch
```

以下のような Makefile を作成しておくと便利である。



```
  Func void afo()
  {
     Matrix A,B,C;

     A = [[1 2][3 4]];
     B = [[5 6][7 8]];
     C = A + B;
     print A,B,C;
  }
#ifdef __MATX__
  main();
#endif
```

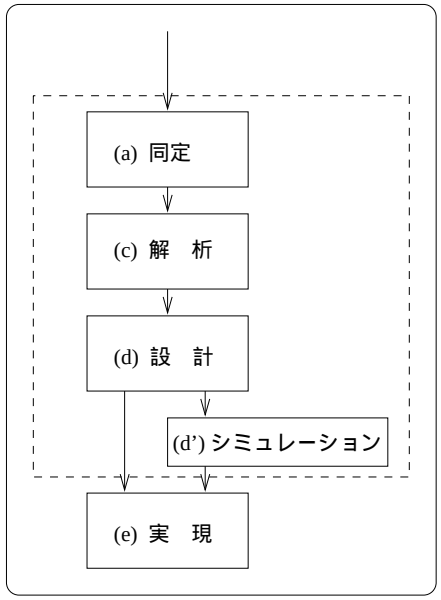
### 3.26.1　ファイルの読み込み

ファイルの読み込み機能は，マクロ定義#defineの集合や変数の宣言などの取り扱いを容易にするためのものである。次の形のソース行

```
  #include "filename.mm"
```

は，ファイル filename.mm の内容と置き換えられる。filename.mm に引用符がついているときには，そのファイルの探索は，元のソースファイルのあるディレクトリから始められる。次のように filename.mm が<と>で囲まれている場合には，コンパイラ (インタプリタ) の規則に従ってそのファイルの探索が行なわれる。

```
  #include <filename.mm>
```




制御系の解析設計手順

## D.1　リアルタイム処理

ディジタル制御を行なうには，ある一定の時間 (サンプリング時間) 毎に

1. センサ (ポテンショメータなど) によるプラント出力の取り込み
2. 制御則の計算
3. アクチュエータ (モータなど) によるプラント入力の操作

の 3 つのステップを繰り返さなければならない。また，この他にも目標値や制御停止命令などを処理するので，多重処理のできるプログラムが必要である。



(c) デフォルト mm ファイルディレクトリ (コンパイル時に決定される) の MaTXOUT.mm

## 3.29　バグ

- プログラム中に転置を意味するシングルクオート “’” があると，その行ではプリプロセッサ (cpp) が置換作業を行わないことがある。

| | |
|---|---|
| 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 |
| 配列 1 < 配列 2 | 各成分の比較 (0\|1) の配列 |
| 配列 1 > 配列 2 | 各成分の比較 (0\|1) の配列 |
| 配列 1 <= 配列 2 | 各成分の比較 (0\|1) の配列 |
| 配列 1 >= 配列 2 | 各成分の比較 (0\|1) の配列 |
| 配列 1 == 配列 2 | 各成分の比較 (0\|1) の配列 |
| 配列 1 != 配列 2 | 各成分の比較 (0\|1) の配列 |

| | |
|---|---|
| 配列 <（整数\|実数） | 各成分の比較 (0\|1) の配列 |
| 配列 >（整数\|実数） | 各成分の比較 (0\|1) の配列 |
| 配列 <=（整数\|実数） | 各成分の比較 (0\|1) の配列 |
| 配列 >=（整数\|実数） | 各成分の比較 (0\|1) の配列 |
| 配列 ==（整数\|実数） | 各成分の比較 (0\|1) の配列 |
| 配列 !=（整数\|実数） | 各成分の比較 (0\|1) の配列 |
| (整数\|実数) < 配列 | 各成分の比較 (0\|1) の配列 |
| (整数\|実数) > 配列 | 各成分の比較 (0\|1) の配列 |
| (整数\|実数) <= 配列 | 各成分の比較 (0\|1) の配列 |
| (整数\|実数) >= 配列 | 各成分の比較 (0\|1) の配列 |
| (整数\|実数) == 配列 | 各成分の比較 (0\|1) の配列 |
| (整数\|実数) != 配列 | 各成分の比較 (0\|1) の配列 |
| 文字列 1 == 文字列 2 | 文字列 1 は文字列 2 と等しい |
| 文字列 1 != 文字列 2 | 文字列 1 は文字列 2 と一致しない |
| 多項式 1 == 多項式 2 | 多項式 1 は多項式 2 と等しい |
| 多項式 1 != 多項式 2 | 多項式 1 は多項式 2 と等しくない |
| 有理多項式 1 == 有理多項式 2 | 有理多項式 1 は有理多項式 2 と等しい |
| 有理多項式 1 != 有理多項式 2 | 有理多項式 1 は有理多項式 2 と等しくない |



### 4.2.2 一般データのファイル入出力

データをファイルへ書き込むには，print コマンドを用いる。出力先を記号->によってファイルにリダイレクトする。matx の場合，拡張子を省略すると，拡張子は mx となる。データは，MX データフォーマットでファイルに書き込まれる。

```
print a, b, c, d -> "filename.mx";
```

データをファイルからの読み込むには，read コマンドを用いる。入力先を記号<-によってファイルにリダイレクトする。matx の場合，拡張子を省略すると，拡張子は mx となる。matx の場合，ファイル名だけを指定すると，ファイルに保存されている全てのデータが読み込まれる。ただし，保存したデータと同じ変数に読み込まれる。

```
read a, b, c, d <- "filename.mx";
```

この方法はデータをバイナリ形式でファイルに保存するので，データの精度が保存される。次に示すように入力先を変更する記号<-は式の中に現われる可能性がある。

```
print 0<-a;
```

この式は-a 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。

### 4.2.3 バイナリデータのファイル入出力

関数 fread(fd, size, type) は指定したファイルからバイナリデータを読み込み，それを成分とする行ベクトルを返す。読み込みに失敗すると，0 x

```
実配列              = 実配列 ^ 実数
複素配列            = 実配列 ^ 複素数
複素配列            = 複素配列 ^ 実数
複素配列            = 複素配列 ^ 複素数

実配列              = 実配列 ^ 実配列
複素配列            = 複素配列 ^ 実配列
実多項式配列        = 実多項式配列 ^ 実配列
複素多項式配列      = 複素多項式配列 ^ 実配列
実有理多項式配列    = 実有理多項式配列 ^ 実配列
複素有理多項式配列 = 複素有理多項式配列 ^ 実配列

実行列              = 実行列 .^ 整数
複素行列            = 複素行列 .^ 整数
実多項式行列        = 実多項式行列 .^ 整数
複素多項式行列      = 複素多項式行列 .^ 整数
実有理多項式行列    = 実有理多項式行列 .^ 整数
複素有理多項式行列 = 複素有理多項式行列 .^ 整数

実行列              = 実行列 .^ 実数
複素行列            = 複素行列 .^ 実数
複素行列            = 複素行列 .^ 複素数

実行列              = 実行列 .^ 実行列
複素行列            = 複素行列 .^ 実行列
実多項式行列        = 実多項式行列 .^ 実行列
複素多項式行列      = 複素多項式行列 .^ 実行列
実有理多項式行列    = 実有理多項式行列 .^ 実行列
複素有理多項式行列 = 複素有理多項式行列 .^ 実行列

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

## C.8　転置 (Transpose)

スカラー ( 整数 , 実数 , 多項式 , 有理多項式 ) は転置によって変わらない。



## 4.4　プロセスの終端の検出

fopen() によってオープンしたファイルからデータを読み込むとき , ファイルの終端を検出したいことがある。C 言語と同様に関数 feof() によって , それを検出できる。関数 feof() に調べたいファイルのディスクリプタを指定すると , ファイルの終端に達していれば , 1 を , 終端に達していなければ 0 を , ファイルがオープンされていなければ −1 を返す。

## 4.5　プロセス間通信

ファイルと同様にプロセスをオープン popen() し ,データをプロセスに送ったり (fprintf()) したり , データをプロセスから受け取ったり (fscanf()) できる。プロセスを popen() でオープンしたあとは , プロセスディスクリプタによってプロセスを指定する。これは , ファイルディスクリプタと同様に扱える。通信が不要になったプロセスは pclose() によって終了する。プロセスディスクリプタを用いたプロセス間通信の例を次に示す。

```
if ((pid = popen("bc")) < 0) {
    error("Can't open %s", "bc");
}
fprintf(pid, "3 + 4\n");
{a} = fscanf(pid, "%d");
print a;
pclose(pid);
```

次に , gnuplot とのプロセス間通信の例を示す。

```
実多項式 (行列|配列)   = 実多項式 * 実 (行列|配列)
実多項式 (行列|配列)   = 実 (行列|配列) * 実多項式
複素多項式 (行列|配列) = 実多項式 * 複素 (行列|配列)
複素多項式 (行列|配列) = 複素 (行列|配列) * 実多項式
複素多項式 (行列|配列) = 複素多項式 * 実 (行列|配列)
複素多項式 (行列|配列) = 実 (行列|配列) * 複素多項式
複素多項式 (行列|配列) = 複素多項式 * 複素 (行列|配列)
複素多項式 (行列|配列) = 複素 (行列|配列) * 複素多項式

実有理多項式 (行列|配列)   = 実有理多項式 * 実 (行列|配列)
実有理多項式 (行列|配列)   = 実 (行列|配列) * 実有理多項式
複素有理多項式 (行列|配列) = 実有理多項式 * 複素 (行列|配列)
複素有理多項式 (行列|配列) = 複素 (行列|配列) * 実有理多項式
複素有理多項式 (行列|配列) = 複素有理多項式 * 実 (行列|配列)
複素有理多項式 (行列|配列) = 実 (行列|配列) * 複素有理多項式
複素有理多項式 (行列|配列) = 複素有理多項式 * 複素 (行列|配列)
複素有理多項式 (行列|配列) = 複素 (行列|配列) * 複素有理多項式

実多項式 (行列|配列)   = (整数|実数) * 実多項式 (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) = (整数|実数) * 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) * (整数|実数)
複素多項式 (行列|配列) = 複素数 * 実多項式 (行列|配列)
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) = 複素数 * 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) * 複素数

実有理多項式 (行列|配列)   = (整数|実数) * 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) * (整数|実数)
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) = 複素数 * 実有理多項式 (行列|配列)
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) = 複素有理多項式 (行列|配列) * 複素数

文字列 = 整数 * 文字列
文字列 = 文字列 * 整数
```



# 第5章　行列エディタ (Matrix Editor)

行列の大きさが大きくなればなるほど，行列の入力・編集作業は面倒になる。入力中にちょっとした間違いを犯しても，もう一度最初からやり直さなければならない。そういった面倒な作業からユーザを解放するために行列エディタ (Matrix Editor) が開発された。ユーザがアルファベットや特殊文字などの不適当な文字をタイプしても，行列エディタが常にタイプされた文字を判断しているので，問題は生じない。行列エディタの特徴を以下に示す。

1. 任意の大きさの行列を入力・編集でき，大きさを任意に変更できる。
2. 大きな行列を編集する時，画面が上下左右にスクロールする。
3. ウィンドウシステムやグラフィック端末を必要とせず，curses ライブラリがサポートするキャラクタ端末，エスケープシーケンスをサポートする端末で使用可能。
4. 実行列だけでなく，複素行列も入力・編集可能。ただし，数式行列の編集機能は未開発。
5. 単位行列の入力，零行列へのリセット，複素行列への変換，転置行列への変換が容易。
6. 行削除，列削除，ペースト，領域の複写が可能。
7. ファイルへの書き込みと，ファイルからの読み込みができる。
8. Emacs 風キー，vi 風キー，矢印キーによるカーソルの移動が可能。

クラス変換される。演算結果はクラスレベルの高い方の値のクラスと一致する。(ただし，スカラー倍と累乗を除く。) 配列の演算は，各成分毎に演算が行なわれ，演算結果は，各成分の演算結果を成分とする配列になる。

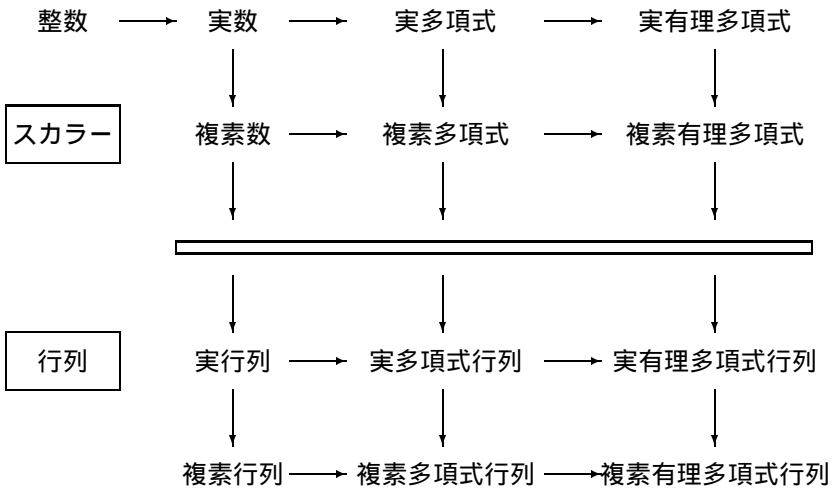





表 5.1: mated のキーバインディング

| 機能 | Emacs | Vi | 特殊キー |
|---|---|---|---|
| 右移動 | ^F | l | → |
| 左移動 | ^B | h | ← |
| 上移動 | ^P | k | ↑ |
| 下移動 | ^N | j | ↓ |
| 1 行の始め | ^A | ^ | |
| 1 行の終り | ^E | $ | |
| 指定した行と列へジャンプ | | g | |
| 最終行最終列 | | G | |
| 1 行削除 | ^K-r | d-r | |
| 1 列削除 | ^K-c | d-c | |
| 書き直し | ^L | | |
| ペースト | ^Y | p | |
| マーク | | m | |
| 領域の獲得 | | i | |
| 領域のペースト | | o | |
| 単位行列 | | E | |
| 零行列 | | Z | |
| 全成分が 1 の行列 | | O | |
| 全成分が等しい行列 | | F | |
| 実部行列 | | R | |
| 虚部行列 | | I | |
| 実行列を複素行列に変換 | | C | |
| 転置行列 (共役転置行列) | | T | |
| ファイルからの読み込み | | r | |
| ファイルへの書き込み | | w | |
| 終了 | | q | ESC |

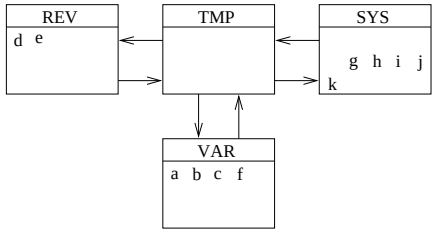



図 B.9: 行列のメモリ管理 (8)

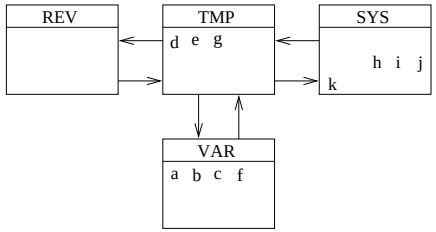



図 B.10: 行列のメモリ管理 (9)

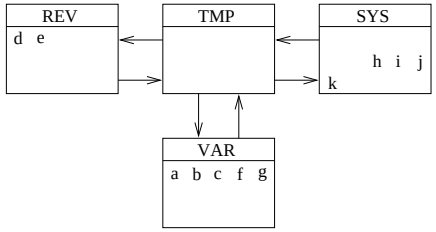



図 B.11: 行列のメモリ管理 (10)

たときのメモリの状態は図 B.12 のようになる。VAR 状態にあったメモリが全て TMP 状態となる。 関数の戻り値が変数に代入されると TMP 状態にある全てのメモリが REV 状態に移り，メモリの状態は図 B.13 のようになる。次に，この関数が呼び出されるときは，全てのメモリは REV 状態から確保され



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

この結果，gnuplot が起動され，データが自動的にスケールされウィンドウ上に表示され，X 軸と Y 軸が描かれる。

データが表示された後で，関数を使って，ラベル，タイトル，を変更したり，Postscript 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，gplot(X,Y)(mgplot(win,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];
Ys = sin(t);
Yc = cos(t);
gplot(t, Ys);
mgplot(2, t, Yc);
```

によって，時刻 0.0 から 4*PI までの正弦波と余弦波を描くことができる。

### 6.1.2　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 gplot()(mgplot()) に gplot(X,Y)(mgplot(win,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，gplot(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，gplot(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。

2. TMP 状態 (malloc() によって OS からメモリを確保した状態)

3. VAR 状態 (変数に代入された状態)

4. REV 状態 (メモリ管理リストに登録された状態)

4 つの状態を遷移する。状態遷移に関係する関数について説明する。

- MaDef() は malloc() を使って OS からメモリを確保する。状態は SYS から TMP に移る。この状態の全てのメモリは ,後述する MatTmpUndef() が呼び出されると , REV 状態に移る。
- MatDestroy() は free() を使って , 全ての状態のメモリを解放して OS に戻す。すなわち , 状態は SYS に移る。
- MatAssign() によって TMP 状態のメモリは VAR 状態に移る。これは変数に代入されたことを意味し , VAR 状態のメモリは , MatTmpUndef() が呼び出されても , 影響されない。MatAssign() の後で MatTmpUndef() が呼び出され , 全ての TMP 状態のメモリが REV 状態に移る。
- 数 MatFree() によって VAR 状態のメモリが TMP 状態に移動する。これは変数の値が必要でなくなったことを意味する。
- MatUndef() と MatInstall() によって TMP 状態のメモリは REV 状態に移り , メモリ管理リストに登録される。
- MatRequest() により REV 状態のメモリは TMP 状態に移る。一般に , 関数 MatRequest() は関数 MatDef() より高速である。

行列クラスライブラリにおいてメモリが必要になった場合 , まず , メモリ管理リストを調べて適当な大きさの行列が登録されていれば , そのメモリを用い , 適当な大きさの行列が登録されていなければ , OS からメモリを確保する。このメモリ管理機構によって , 関数 malloc() と free() が呼ばれる回数を減らし , 内部処理の高速化を行なっている。特に , この方法はシミュレーションのように同じプログラムが繰り返し呼ばれる場合に有効である。



表 6.4: グラフィック関数 4(gnuplot)

| 関数名/引数 | 機能 |
|---|---|
| greplot_semilogx(X, Y)<br>greplot_semilogx(X, Y, titles)<br>greplot_semilogx(X, Y, titles, cmds)<br>Array X, Y; List titles; List cmds; | セミログ x–y プロット<br>(x 軸 ログ) |
| greplot_semilogy(X, Y)<br>greplot_semilogy(X, Y, titles)<br>greplot_semilogy(X, Y, titles, cmds)<br>Array X, Y; List titles; List cmds; | セミログ x–y プロット<br>(y 軸 ログ) |
| greplot_loglog(X, Y)<br>greplot_loglog(X, Y, titles)<br>greplot_loglog(X, Y, titles, cmds)<br>Array X, Y; List titles; List cmds; | ログログ x–y プロット |
| mgreplot_semilogx(win, X, Y)<br>mgreplot_semilogx(win, X, Y, titles)<br>mgreplot_semilogx(win, X, Y, titles, cmds)<br>Integer win; Array X, Y;<br>List titles; List cmds; | セミログ x–y プロット<br>(x 軸 ログ) |
| mgreplot_semilogy(win, X, Y)<br>mgreplot_semilogy(win, X, Y, titles)<br>mgreplot_semilogy(win, X, Y, titles, cmds)<br>Integer win; Array X, Y;<br>List titles; List cmds; | セミログ x–y プロット<br>(y 軸 ログ) |
| mgreplot_loglog(win, X, Y)<br>mgreplot_loglog(win, X, Y, titles)<br>mgreplot_loglog(win, X, Y, titles, cmds)<br>Integer win; Array X, Y;<br>List titles; List cmds; | ログログ x–y プロット |

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))。

複素有理多項式行列　rows×cols 個の複素有理多項式。
(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))。

リスト　length 個の成分のクラスを表す 4 バイト長整数とデータ
の結合。

配列　実配列　rows×cols 個の精度に応じた実
数 (4 バイトまたは , 8 バイト)。

複素配列　2×(rows×cols) 個の精度に応じ
た実数 (4 バイトまたは , 8 バイ
ト)。

実多項式配列　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))。

複素多項式配列　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))。



表 6.7: グラフィック関数 7(gnuplot)

| 関数名 | 引数 | 機能 |
|---|---|---|
| gplot_title(text) | String text | グラフタイトル |
| gplot_xlabel(text) | String text | X 軸ラベル |
| gplot_ylabel(text) | String text | Y 軸ラベル |
| gplot_grid() | | グリッド (トグル) |
| gplot_grid(onoff) | Integer onoff | グリッド線 |
| gplot_key() | | 線のタイトル (トグル) |
| gplot_key(onoff) | Integer onoff | 線のタイトル |
| gplot_replot() | | ウィンドウの再表示 |
| gplot_cmd(cmd) | String cmd | 任意のコマンド |
| gplot_text(text,x,y) | String text | 任意のテキスト |
| | Integer x, y | x, y 座標 |
| gplot_options(options) | String options | gnuplot のオプション |
| mgplot_title(win, text) | Integer win | グラフタイトル |
| | String text | |
| mgplot_xlabel(win, text) | Integer win | X 軸ラベル |
| | String text | |
| mgplot_ylabel(win, text) | Integer win | Y 軸ラベル |
| | String text | |
| mgplot_grid(win) | Integer win | グリッド線 (トグル) |
| mgplot_grid(win, onoff) | Integer onoff | グリッド線 |
| mgplot_key(win) | Integer win | 線のタイトル (トグル) |
| mgplot_key(win, onoff) | Integer onoff | 線のタイトル |
| mgplot_replot(win) | Integer win | ウィンドウの再表示 |
| mgplot_cmd(win, cmd) | String cmd | 任意のコマンド |
| mgplot_text(win,text,x,y) | String text | 任意のコマンド |
| | Integer x,y | x, y 座標 |
| mgplot_options(win,options) | String options | gnuplot のオプション |

| データ型 | C |
|---|---|
| Matrix | 0 |
| Integer | 1 |
| Real Number | 2 |
| Complex Number | 3 |
| String | 4 |
| Polynomial | 5 |
| Rational | 6 |
| List | 7 |
| Array | 8 |
| Index | 9 |

2. 2 番目の整数は，データの型によって表すものが異なる。

| データ型 | 名前 | 内容 |
|---|---|---|
| Matrix | rows | 行列の行の数 |
| Integer | dummy | |
| Real Number | dummy | |
| Complex Number | dummy | |
| String | stringlen | 文字列長 |
| Polynomial | degree | 次数 |
| Rational | ndegree | 分子の次数 |
| List | length | 成分の数 |
| Array | rows | 配列の行の数 |
| Index | rows | 1 |

3. 3 番目の整数は，データの型によって表すものが異なる。



表 6.10: グラフィック関数 1(xplot)

| 関数名 | 引数 | 機能 |
|---|---|---|
| xplot(Y)<br>xplot(X, Y) | Array X, Y | 線形 x–y プロット |

```
% setenv DISPLAY ‘hostname‘:0.0
```

いったん xplot が起動されると，マウスを使ってポップアップメニューでラベル，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。また，次の関数を使ってもラベル，タイトル，グリッドを描くことができる。

表 6.11: グラフィック関数 2(xplot)

| 関数名 | 引数 | 機能 |
|---|---|---|
| xplot_title(text) | String text | グラフタイトル |
| xplot_xlabel(text) | String text | X 軸ラベル |
| xplot_ylabel(text) | String text | Y 軸ラベル |
| xplot_grid() | | グリッド線 |

### 6.2.1　基本的なプロット

もし Y が横ベクトルなら xplot(Y) によって Y の成分とその成分の指数を対応させた線形プロットが作成される。たとえば，ある実数の集合 {0., .48, .84, 1., .91, .6, .14} をプロットするには，その実数をベクトルの中に入れて xplot() 関数を次のように実行する。

```
# For fvwm-1
Lenience

# For fvwm-2
# Style "*" Lenience
```

インタプリタ (matx) の実行中，関数の外側では，どんなデータでもそのクラスに関係なく任意の変数に代入することができる。すなわち，左辺の変数のクラスと右辺のデータのクラスが一致してなくても，代入は成功する。また，次の節で説明する暗黙の型変換によって，異なる型のデータの代入が可能となる場合がある。これ以外の全ての場所では，代入において，左辺の変数のクラスと右辺のデータのクラスは一致していなければならない。

## 7.3　静的変数

静的 (static) 変数は外部変数や自動変数に続く第 3 の記憶クラスである。MATX には C 言語と異なり局所 static 変数のみ存在し，大域 static 変数は存在しない。なぜなら，インタプリタにとって大域 static 変数を通常の大域変数と区別することに意味がないからである。

static 変数は自動変数と異なり，関数が起動される毎に初期化されることなく，ずっと存在し続ける。これは，static 変数が関数内で私的な永久記憶をもつをことを意味する。

static 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。

### 17.5.4　複雑なシステムのシミュレーション

ここでは，もっと複雑なシステムのシミュレーションを行なう方法を紹介する。システムが複雑になると，微分方程式を記述する関数 diff_eqs() や各システムの接続関係を記述する関数 link_eqs() の先頭において，拡大された状態 X から各システムの状態を取り出す作業が複雑になる。この作業を簡単にする関数 VecCont() と VecChop() の使い方を下図に示すシステムのシミュレーションを通して説明する。

図に示す各ブロック $P$,$D$,$C$,$F$ はプラント，プラントの不確定要素，前置コントローラ，フィードバックコントローラを表し，各ブロックは線形状態方程式と線形出力方程式で記述され，その係数は $(Ap, Bp, Cp)$, $(Ad, Bd, Cd)$, $(Ac, Bc, Cc)$, $(Af, Bf, Cf)$ であるとする。

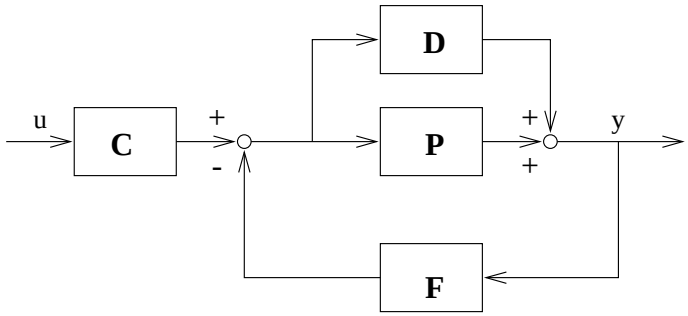



図 17.2: 複雑なシステムのシミュレーション



# 第 8 章　関数

関数には，行なうべき計算の過程を指示する文，および計算で使われる値を格納する変数が含まれる。MATX の関数は，C 言語の関数と似ている。普通は，関数には自分の好きな名前を付けてもよいが，main は特別な名前である。コンパイラ (matc) を使って実行可能型ファイルを作成した時，プログラムの実行は関数 main() の先頭から始まる。これは，プログラムにはどこかに関数 main() がなければならないことを意味する。ただし，インタプリタ (matx) を使ってプログラムを実行する時は，プログラムのどこからでも実行を始めることができ，関数 main() をプログラムの中に作る必要はない。

main() の型は，Integer あるいは void である。関数 main() の型が Integer の時，他の関数と同様にその呼び出し側に対して値を返すことができる。この場合，呼び出し側とは，そのプログラムが実行される環境である。通常，戻り値が 0 であれば正常終了，0 でない値は異常な，あるいはエラー の終了状況を表す。

MATX の関数は，再帰的に使用できる。つまり，関数は自分自身を直接または間接に呼び出してよい。そして，関数の局所変数は，その関数が呼び出される毎に新しく作られる。関数の定義は，入れ子 (ネスト) にできない。また，関数はいくつかのファイルに分けて定義でき，コンパイルをファイル毎に行なえる。

## 8.1　関数の宣言

関数を宣言することによって，その関数を呼び出すことが可能となる。関数名は，英文字または下線 “_” で始まる任意長の英数字文字列であり，大文

```
Matrix Ap, bp, cp;      // 制御対象のパラメータ
Matrxi Ac, bc, cc, dc; // 制御器のパラメータ

Func void init_params()
{
  // 制御対象のパラメータ
  Ap = [[0  1]
        [2  1]];
  bp = [0 1]';
  cp = [-1, 1];

  // 制御器のパラメータ
  Ac = [5];
  bc = [1];
  cc = [54];
  dc = [9];
}

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

  void diff_eqs(), link_eqs(), init_params();

  init_params();

  x0 = [1 0 0]';
  {TC, XC, UC} =
     Ode45Auto(0.0, 20.0, x0, diff_eqs, link_eqs);

  print [[TC][XC][UC]] >> "cont.mat";
}
```

List 7-1:　動的フィードバック制御のシミュレーション (大域変数使用)



## 8.4　可変長引数関数

### 8.4.1　リストを利用した可変長引数関数

リストを利用して，可変長の引数をもつ関数を定義することができる。可変長の引数を持つ関数の例として，引数として渡された行列の和を返す関数を以下に示す。ただし，引数の行列の大きさは，全て等しいと仮定している。

```
Func Matrix sum_mat(args)
  List args;
{
  Integer i, argn;
  Matrix ss;

  argn = length(args);
  ss = args(1, Matrix);
  for (i = 2; i <= argn; i++) {
    ss = ss + args(i, Matrix);
  }
  return ss;
}
```

可変長の引数を持つ関数を呼び出す時は，渡したい引数から 1 つのリストを作り，それを引数として渡す。関数 sum_mat() の呼び出しは，次のようになる。

```
a = [[1 0][0 1]]; b = [[1 2][3 4]];
c = [[4 3][2 1]]; d = [[0 1][1 0]];
print sum_mat({a, b, c, d});
```

```
Func void main()
{
  Real dt;
  Matrix x0;
  Array TC, XC, UC, TH, XH, UH;
  void diff_eqs(), link_eqs();

  dt = 0.01;        // サンプリング時間
  x0 = [1 1 1]';    // 初期状態
  {TC,XC,UC} = Ode45Auto(0.0, 5.0, x0, diff_eqs, link_eqs);
  x0 = [1 1 1]';    // 初期状態
  {TH,XH,UH} =
     Ode45HybridAuto(0.0, 5.0, dt, x0, diff_eqs, link_eqs);

  print [[TC][XC][UC]] >> "cont.mat";
  print [[TH][XH][UH]] >> "hybd.mat";
}

Func void diff_eqs(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 void link_eqs(u, t, x)
  Real t;
  Matrix u, x;
{
  Matrix f;

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

List 5:　状態フィードバック制御のシミュレーション (1)



## 8.8　関数の参照

MATX では，関数それ自身は変数ではないが，関数の参照を関数の引数として渡すことができる。関数の参照は，関数の名前と同じように使える。ただし，C 言語の関数のポインタに対応するような記述方法はない。

```
Func void main()
{
  Matrix a;
  void foo(), bar();

  a = [[1 2][3 4]];
  foo(bar, a);
}

Func void foo(afo, a)
  void afo();
  Matrix a;
{
  afo(a);
}

Func void bar(a)
  Matrix a;
{
  print a;
}
```

## 8.9　コマンドラインの引数

コマンドラインの引数は，関数 `main()` にリストの形で引数として渡される。リストの全ての成分は文字列であり，1 番目の成分はコマンド名である。整数や実数をコマンドラインで渡すには，リストの成分を取り出してクラス変換関数で変換して利用する。例えば，

## 17.4 線形システムのボードとナイキスト周波数応答

与えられた線形システムの伝達関数のボード線図とナイキスト線図を描くためのデータを計算する。以下は，リストの説明である。

- 1 行，19 行，33 行: 関数の始まり。
- 3 行，5 行，23 行，24 行，37 行，38 行: 局所変数の宣言。
- 6 行: 関数の宣言。
- 8 行: 多項式変数の定義。
- 9 行: 3 次実有理関数を入力。
- 11 行: 対数空間ベクトル。
- 12 行，13 行: 複数の戻り値を返す関数 `Bode()` と `Nyquist()` を呼ぶ。
- 20 行，21 行，34 行，35 行: 関数の引数の宣言。
- 26 行，40 行: 虚数単位の定義。
- 27 行，41 行: 有理関数配列の評価。(配列の各成分毎に評価する)
- 28 行，29 行: 配列演算。
- 30 行，42 行: 2 つの成分を持つリストを返す。



## 9.3 Do-while ループ

do-while ループでは，ループ本体を通る各パスを終ったあとにテストが行なわれる。ループ本体はつねに，少なくとも 1 回は実行される。その構文は

```
do {
    statements;
} while (expr) ;
```

ここでは，まず `statements` が実行された後，`expr` が計算される。それが真ならば，`statements` は再び実行される。このループが終了するのは，`expr` が偽になったときである。

## 9.4 Switch 文

`switch` 文は，ある式が複数の値の 1 つと一致しているかどうかテストし，それに応じて分岐する特別の多分岐判断機構である。

```
switch (exp) {
 case int_1: statements;
 case int_2: statements;
 ....................
 case int_n: statements;
 default: statements;
}
```

`switch` 文は，括弧内の整数式を計算し，その値を全ての `case` と比較する。各 `case` には，整数定数の式で名札をつけなければならない。ある `case` が式の値と一致すれば，実行はその `case` で始められる。`default` のラベルのついた `case` は，他のいずれの `case` も満足されない時に実行される。

`break` 文スイッチから直接抜け出る手段である。`case` は単に名札として働くから，1 つの `case` プログラムが終了したのち，抜け出すための明示的な動作がなければ，実行は次の `case` へ移る。

## 17.2　離散系リカッティ方程式

離散時間の線形システムのリカッティ方程式の解を繰り返しにより求めるプログラムの例をに示す。実際の問題には一般化固有値問題に帰着する方法が有効である。解行列 P と一時変数行列 PP は，行列 A と同じ大きさの単位行列で初期化される。解行列 P と 1 回前の解行列 PP の差のフロベニウス・ノルムが eps より小さくなるまで離散時間系のリカッティ方程式の計算が繰り返される。以下は，リストの説明である。

- 1 行と 16 行: 関数の始まり。
- 3 行，19 行，20 行: 局所変数の宣言。
- 6 行，8 行，10 行，23 行: 定数行列の入力。
- 9 行，26 行，27 行: 行列の計算。
- 12 行: 関数 DRiccati() を呼ぶ。
- 13 行: 解行列 P を表示する。
- 17 行: 関数の引数の宣言。
- 26 行: 行列 P - PP のフロベニウスノルムの大きさを調べる。
- 28 行: 離散時間系のリカッティ方程式の計算
- 29 行: 行列 P の対称性を回復。



## 10.3　整数関数

整数関数の一覧を以下に示す。

```
Integer i, i2;
Real tol;

Integer bit_and(i1, i2)     ビット毎の AND (i1 & i2)
Integer bit_comp(i1)        1 の補数 (~i1)
Integer bit_or(i1, i2)      ビット毎の OR (i1 | i2)
Integer bit_xor(i1, i2)     ビット毎の排他的 OR (i1 ^ i2)
Integer bit_lshift(i1, i2) i2 ビット左シフト (i1 << i2)
Integer bit_rshift(i1, i2) i2 ビット右シフト (i1 >> i2)
Integer rem(i1, i2)         割算の余り (i1 % i2)
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))
Integer max(i1, i2)         最大値
Integer min(i1, i2)         最小値
Integer sgn(i1)             符合関数 (i1 > 0, 1;
                                      i1 = 0, 0;
                                      i1 < 0, -1)
Integer pow(i1, i2)         i1^i2
Integer fact(i1)            i1 の階乗
Integer trans(i1)           i1'
Integer conj(i1)            i1 の共役
Integer conjtrans(i1)       i1#
Integer inv(i1)             i1~ (いつもゼロ)
Integer round2z(i1)         ゼロへの丸め (tol = EPS)
Integer round2z(i1, tol)    ゼロへの丸め

Integer c, fd;
```

## 16.5　リスト関数

　リスト関数の一覧を以下に示す。関数 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る。

```
Integer length(List L1);                  リストの成分の数
List    appendlist(List L1, Object A); L1 に成分 A を付加する。
List    fscanf(Integer fd, String format, ...);  ファイル入力
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
```

| | |
|---|---|
| Array acos(A1); | 各成分の acos() の結果からなる配列 |
| Array cosh(A1); | 各成分の cosh() の結果からなる配列 |
| Array acosh(A1); | 各成分の cosh() の結果からなる配列 |
| Array tan(A1); | 各成分の tan() の結果からなる配列 |
| Array atan(A1); | 各成分の atan() の結果からなる配列 |
| Array tanh(A1); | 各成分の tanh() の結果からなる配列 |
| Array atanh(A1); | 各成分の atanh() の結果からなる配列 |
| Array ceil(A1); | 各成分の ceil() の結果からなる配列 |
| Array floor(A1); | 各成分の floor() の結果からなる配列 |
| Array fix(A1); | 各成分の fix() の結果からなる配列 |
| Array round(A1); | 各成分の round() の結果からなる配列 |
| Array round2z(A1) | ゼロへの丸め (tol = EPS) |
| Array round2z(A1, tol) | 絶対値が tol より小なら 0 |
| Array pow(A1, i1); | 各成分の pow() の結果からなる配列 ( A1^i1 ) |
| Array trans(A1); | A1 の転置 ( A1' ) |
| Array conjtrans(A1); | A1 の共役転置 ( A1# ) |



| | |
|---|---|
| Complex sin(c1); | 正弦関数 |
| Complex sinh(c1); | 双曲線正弦関数 |
| Complex asin(c1); | 逆正弦関数 |
| Complex asinh(c1); | 逆双曲線正弦関数 |
| Complex cos(c1); | 余弦関数 |
| Complex cosh(c1); | 双曲線余弦関数 |
| Complex acos(c1); | 逆余弦関数 |
| Complex acosh(c1); | 逆双曲線余弦関数 |
| Complex tan(c1); | 正接関数 |
| Complex tanh(c1); | 双曲線正接関数 |
| Complex atan(c1); | 逆正接関数 |
| Complex atanh(c1); | 逆双曲線正接関数 |
| Complex log(c1); | 自然対数 |
| Complex log10(c1); | 常用対数 |
| Complex exp(c1); | 指数関数 |
| Complex conj(c1); | 共役複素数 |
| Complex sqrt(c1); | 平方根 |
| Complex sgn(c1); | 符合関数 (c1/abs(c1)) |
| Complex pow(c1, c2) | c1^c2 |
| Complex trans(c1) | c1' |
| Complex conj(c1) | c1 の共役 |
| Complex conjtrans(c1) | c1# |
| Complex inv(c1) | c1~ |
| Complex ceil(c1) | 最高値の整数への丸め関数 |
| Complex floor(c1) | 最低値の整数への丸め関数 |
| Complex fix(c1) | 0 方向の整数への丸め関数 |
| Complex round(c1) | 最も近い整数への丸め関数 |
| Complex round2z(c1) | ゼロへの丸め (tol = EPS) |
| Complex round2z(c1, tol) | 絶対値が tol より小なら 0 |

```
Complex prod(CoArray A1);     全ての成分の積
Polynomial prod(PoArray A1); 全ての成分の積
Rational prod(RaArray A1);    全ての成分の積

Real mean(Array A1);          全ての成分の平均値
Complex mean(CoArray A1);     全ての成分の平均値
Polynomial mean(PoArray A1); 全ての成分の平均値
Ratioanal mean(RaArray A1);  全ての成分の平均値

Real std(Array A1);           全ての成分の標準偏差
Complex std(CoArray A1);      全ての成分の標準偏差
Polynomial std(PoArray A1);  全ての成分の標準偏差
Rational std(RaArray A1);     全ての成分の標準偏差

Array cumsum(Array A1);       全ての成分の累積和
Array cumprod(Array A1);      全ての成分の累積積


Array A1;

Array sum_col(A1);       (縦・横) ベクトルの成分の和
                         列毎の和からなる横ベクトル
Array sum_row(A1);       (縦・横) ベクトルの成分の和
                         行毎の和からなる縦ベクトル
Array prod_col(A1);      (縦・横) ベクトルの成分の積
                         列毎の積からなる横ベクトル
Array prod_row(A1);      (縦・横) ベクトルの成分の積
                         行毎の積からなる縦ベクトル
Array mean_col(A1);      (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
Array mean_row(A1);      (縦・横) ベクトルの成分の平均値
```



において，eval("a + b") は，実数 a と b の和の結果を返すのに対して，eval("A + B") は，行列 A と B の和の結果を返す。

また，次の例に示すように eval() を関数内で用いると，プログラムのデバッグを効率的に行なうことができる。

```
Func void afo()
{
  String cmd;

  ...................

  for (;;) {
    read cmd;
    if (cmd == "q") {
      break;
    }
    eval(cmd);
  }

  ...................
}
```

関数内で eval() を使うとき，引数として文字列"who"と"whos"を用いると，その関数の局所変数と大域変数の両方を参照することができる。

```
Func void afo()
{
  ...................

  eval("who");
  eval("whos");

  ...................
}
```

```
                    val: 一般固有値 (対角行列)
                    vec: 一般固有ベクトル

Matrix M1;

List svd(M1);       特異値分解
                    {U, D, V} = svd(A);
                    U * D * V# = A;
                    U: 左特異ベクトル
                    V: 右特異ベクトル
                    D: 特異値 (A と同じ大きさの行列)
List lu(M1);        LU 分解
                    {L, U, P} = lu(A);
                    L * U = P * A;
                    L: 下三角行列
                    U: 上三角行列
                    P: 置換行列
List hess(M1);      Hessenberg 分解
                    {Q, H} = hess(A);
                    Q# * A * Q = H;
                    Q: 直交行列 or ユニタリ行列
                    H: ヘッセンベルグ行列
List schur(M1);     Schur 分解
                    {U, T} = schur(A);
                    U# * A * U = T;
                    U: 直交行列 or ユニタリ行列
                    T: 擬似上三角行列 or 上三角行列,

Matrix M1, M2;
```



# 第13章　多項式

実数と複素数を係数とする 1 変数の多項式の数式処理を行なうことができる。今のところ，多項式の係数は，昇べきの順に横ベクトルとして保存されている。

[ 宣言 ]

```
Polynomial   a, b, c;    // 実多項式
CoPolynomial a2, b2, c2; // 複素多項式
```

[ 例 ]

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

## 13.1　多項式の記述方法

多項式は，多項式変数をドル記号 $ を用いて定義し，その変数を用いて記述する。多項式の係数が整数の時，自動的に実数に変換される。1 次の多項式の変数名には，任意の名前を使うことができるが，出力される多項式の変数名は，常に s である。クラス変換関数 Polynomial() と Matrix() を使って横ベクトルと多項式を相互に変換できる。多項式の係数は，横ベクトルの成分と昇べきの順に一致する。

```
s = $;                      // 1 次の係数が 1 の 1 次多項式
P1 = 2*s^2 + 3*s + 4;       // 2 次の実多項式
P2 = Polynomial([4 3 2]); // P2 = 2*s^2 + 3*s + 4
CP1 = (1,2)*s + (3,4);      // 1 次の複素多項式
```

```
List Ode45(Real t0, Real t1, Matrix x, void dx());
                              : h=4.0E-2, dtsav=1.0E-3
List Ode45(Real t0, Real t1, Matrix x, void dx(),
             void input()); : h=4.0E-2, dtsav=1.0E-3
List Ode45(Real t0, Real t1, Matrix x, void dx(),
             void input(), Real h); : dtsav=1.0E-3
List Ode45(Real t0, Real t1, Matrix x, void dx(),
             void input(), Real h, Real dtsav);
```

RKF45 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。
{T, X, U} = Ode45();
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 Ode45Auto(Real t0, Real t1, Matrix x, void dx());
               : eps=1.0E-6, dtsav=1.0E-3,
                 dtmin=EPS, dtmax=abs(t1-t0)/5
List Ode45Auto(Real t0, Real t1, Matrix x, void dx(),
               void input());
               : eps=1.0E-6, dtsav=1.0E-3,
                 dtmin=EPS, dtmax=abs(t1-t0)/5
List Ode45Auto(Real t0, Real t1, Matrix x, void dx(),
               void input(), Real eps);
              : dtsav=1.0E-3, dtmin=EPS, dtmax=abs(t1-t0)/5
List Ode45Auto(Real t0, Real t1, Matrix x, void dx(),
               void input(), Real eps, Real dtsav);
              : dtmin=EPS, dtmax=abs(t1-t0)/5
List Ode45Auto(Real t0, Real t1, Matrix x, void dx(),
               void input(), Real eps,
```



| 表 記 | 意 味 |
|---|---|
| P1 == P2 | 多項式 P1 と P2 は等しい |
| P1 != P2 | 多項式 P1 と P2 は等しくない |

## 13.8　多項式関数

多項式関数の一覧を以下に示す。

```
Polynomial P1;
Integer i1;
Real tol;
```

```
Matrix  roots(P1);          多項式の根
Integer degree(P1);         多項式の次数
Integer isreal(P1);         実多項式の有無
                            (真 = 1, 偽 = 0)
Integer iscomplex(P1);      複素多項式の有無
                            (真 = 1, 偽 = 0)

Polynomial floor(P1)        係数の最低値の整数への丸め
Polynomial ceil(P1)         係数の最高値の整数への丸め
Polynomial fix(P1)          係数の 0 方向の整数への丸め
Polynomial round(P1)        係数の最も近い整数への丸め
Polynomial round2z(P1)      係数の 0 への丸め (tol = EPS)
Polynomial round2z(P1, tol) 係数の絶対値が tol より小なら 0
Polynomial pow(P1, i1)      P1^i1
Polynomial trans(P1)        P1'
Polynomial conj(P1)         P1 の共役
Polynomial conjtrans(P1)    P1#
Rational   inv(P1)          P1~
```

計算する

```
Matrix M1;
Integer m, n;

Matrix  rand(M1);          M1 と同じ大きさの一様乱数行列
Integer rand("seed"):      一様乱数の種
Integer rand("seed", m):   一様乱数の種を設定
Matrix  randn(M1);         M1 と同じ大きさの正規乱数行列

Matrix  rngkut4(Real t, Matrix x, void dx())
                : dt = 1.0E-7, u = Z(0);
Matrix  rngkut4(Real t, Matrix x, void dx(), Matrix u)
                : dt = 1.0E-7
Matrix  rngkut4(Real t, Matrix x, void dx(), Matrix u,
                Real dt)
```

4 次のルンゲ- クッタの公式にしたがって常微分方程式の解を求める。

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



## 14.4　有理多項式の次数の増加

有理多項式の演算は次の規則に従って行なわれ，分子多項式と分母多項式の共通項のキャンセルは行なわれないので，次数の増加に注意する必要がある。

$$\frac{a}{b}+\frac{c}{c}=\frac{a\cdot c+b\cdot c}{b\cdot c}\quad,\quad \frac{a}{b}-\frac{c}{c}=\frac{a\cdot c-b\cdot c}{b\cdot c}$$

$$\frac{a}{b}*\frac{c}{c}=\frac{a\cdot c}{b\cdot c}\quad,\quad \frac{a}{b}/\frac{c}{c}=\frac{a\cdot c}{b\cdot c}$$

## 14.5　有理多項式の評価

関数 eval() を用いて，有理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 = $;
R1 = (s + 2)/(2*s + 3);
r1 = eval(R1, 2.0);          // 2.0 を代入
R2 = eval(R1, s + 1);        // s+1 を代入
R2 = eval(R1, (s+1)/(s+2)); // (s+1)/(s+2) を代入
A2 = eval(R1, [1:4]);        // (1,2,3,4) を代入
M2 = eval(R1, M);            // (M+2*I(M))/(s*M+3*I(M)
```

有理多項式の評価に伴うクラス変換の規則を以下に示す。

```
                          行毎の標準偏差からなる縦ベクトル
Matrix cumsum_col(M1);    列毎の累積和
Matrix cumsum_row(M1);    行毎の累積和
Matrix cumprod_col(M1);   列毎の累積積
Matrix cumprod_row(M1);   行毎の累積積
```

```
Matrix M1, M2;

Matrix eigval(M1);        固有値
Matrix eigval(M1, M2);    一般固有値
Matrix eigvec(M1);        固有ベクトル
Matrix eigvec(M1, M2);    一般固有ベクトル
```

eigval() は固有値を実部の降順に返し，eigvec() はその固有値に対応する順で固有ベクトルを返す。

```
Matrix M1;

Matrix singval(M1);         特異値
Matrix singleftvec(M1);     左の特異ベクトル
Matrix singrightvec(M1);    右の特異ベクトル
```

singval() は特異値を降順に返し，singleftvec() と singrightvec() はその特異値に対応する順で特異ベクトルを返す。

```
Matrix M1;
Real tol;

Matrix  vectorprod(M1);       ベクトル積，外積
Matrix  exp(M1);              指数行列
Matrix  sqrt(M1);             平方根行列
Matrix  log(M1);              対数行列
```



# 第15章　行列と配列と指数

行列と配列のデータ構造は全く一致し，複数のデータを 1 次元的または 2 次元的に保存するという性質は変わらない。行列の演算では代数的な行列演算が行なわれるのに対し，配列の演算では配列の各成分に対して演算が行なわれ，演算結果は各成分の演算結果を成分とする配列となる。計算の途中で一時的に行列を配列として扱う配列演算子 .*, ./, .\, .^ やスカラと行列や配列の演算においてスカラをその値で満たされた行列として扱う演算子 .+と .-がある。行列と配列はクラス変換関数 Matrix() と Array() によって相互変換できる。便利な添字記法を使えば，行・列・成分・部分行列を簡単に参照できる。

　指数は整数を 1 次元的に並べたものであり，部分行列の取り出しや代入といった操作に使われる。クラス変換関数 Index() を使って行列や配列から指数を作れる。

[ 宣言 ]

```
  Matrix A, B, C;       // 実行列
  CoMatrix CA, CB, CC; // 複素行列
  PoMatrix PA, PB, PC; // 多項式行列
  RaMatrix RA, RB, RC; // 有理多項式行列

  Array D, E, F;        // 実配列
  CoArray CD, CE, CF;  // 複素配列
  PoArray PD, PE, PF;  // 多項式配列
  RaArray RD, RE, RF;  // 有理多項式配列

  Index II, JJ;         // 指数
```

[ 例 ]

```
  [[1 2][3 4]], [(1,2), (3,4)]                 // 行列
  Array([[1 2][3 4]]), Array([(1,2), (3,4)])  // 配列
  Index([1 2 3 4]), Index([10 8 6 4 2])       // 指数
```

## 15.8　行列関数

行列関数の一覧を以下に示す。固有値の計算はEISPACKのアルゴリズムを使っている [16],[15],[4]。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解を求める関数 OdeAuto() は，文献 [14] のアルゴリズムを参照している。また，RKF45 の公式にしたがって，指定されたきざみ幅で，常微分方程式の解を求める関数 Ode45() は，文献 [3] のアルゴリズムを参照している。LU 分解，QR 分解，QZ 分解は，文献 [5] を参考にした。

```
Matrix M1;
Real tol;

Integer  rank(M1);        ランク
Integer  rank(M1, tol);   ランク
                          tol: 許容誤差
Integer  Rows(M1);        行の数
Integer  Cols(M1);        列の数
Integer  length(M1);      行と列の大きい方の数
Integer  isreal(M1);      実行列の有無（真 = 1，偽 = 0）
Integer  iscomplex(M1);   複素行列の有無（真 = 1，偽 = 0）
Integer  isscalar(M1);    スカラーの有無（真 = 1，偽 = 0）
Array    finite(M1);      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(M1);       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
Matrix   sgn(M1);         各成分の sgn() を結果からなる行列
```



### 15.2.1　零行列

ピース行列[1]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零行列を簡単に作ることができる。

```
M1 = Z(3);          3x3 の実零行列
M1 = Z(3, c);       3x3 の複素零行列,          c = 複素数
M1 = Z(3, P);       3x3 の実多項式零行列,      P = 実多項式
M1 = Z(3, CP);      3x3 の複素多項式零行列,    CP = 複素多項式
M1 = Z(3, R);       3x3 の実有理多項式零行列,   R = 実有理多項式
M1 = Z(3, CR);      3x3 の複素有理多項式零行列, CR = 複素有理多項式
M1 = Z(3, 4);       3x4 の実零行列,
M1 = Z(3, 4, c);    3x4 の複素零行列,          c = 複素数
M1 = Z(3, 4, P);    3x4 の実多項式零行列,      P = 実多項式
M1 = Z(3, 4, CP);   3x4 の複素多項式零行列,    CP = 複素多項式
M1 = Z(3, 4, R);    3x4 の実有理多項式零行列,   R = 実有理多項式
M1 = Z(3, 4, CR);   3x4 の複素有理多項式零行列, CR = 複素有理多項式
M2 = Z(M1);         M1 と同じ大きさで同じクラスの零行列
M2 = Z(3, M1);      M1 を横 3 個並べた大きさで同じクラスの零行列
M2 = Z(2, 3, M1);   M1 を縦 2 個に横に 3 個並べた大きさで
                    M1 と同じクラスの零行列)
```

### 15.2.2　単位行列

行列が非正方である時，対角成分のみ 1 で他の成分は 0 である。多項式行列の場合，対角成分のみ定数項が 1 である多項式になる。有理多項式行列の場合，対角成分のみ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単位行列を簡単に作ることができる。

```
M1 = I(3);          3x3 の実単位行列
M1 = I(3, c);       3x3 の複素単位行列,          c = 複素数
M1 = I(3, P);       3x3 の実多項式単位行列,      P = 実多項式
M1 = I(3, CP);      3x3 の複素多項式単位行列,    CP = 複素多項式
M1 = I(3, R);       3x3 の実有理多項式単位行列,   R = 実有理多項式
M1 = I(3, CR);      3x3 の複素有理多項式単位行列, CR = 複素有理多項
式
M1 = I(3, 4);       3x4 の実行列,
M1 = I(3, 4, c);    3x4 の複素行列,              c = 複素数
```

[1] 行列の配列参照

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



```
s = $;

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

PM1 = diag(s+1, s+2, s+3, s+4);
```

多項式行列を記述する便利な方法には，少なくとも 2 つある。1 つは，

```
s = $;

A = [[1 2][3 4]]*s^2 + [[5 6][7 8]]*s
    + [[9 10][11 12]];
```

であり，もう 1 つは

```
s = $;

A = [[  s^2+5*s+9 , 2*s^2+6*s+10]
     [3*s^2+7*s+11, 4*s^2+8*s+12]];
```

で，どちらも同じ多項式行列を作る。前者は，行列を多項式の係数として用い，後者は，4 つの多項式を陽に行列の成分として使っている。

複素多項式行列の記述方法には，複素多項式を成分として記述する方法と一対の実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 * を指定して複素多項式行列を記述することもできる。

```
CPM1 = (PM1,PM2); // CPM1 = PM1 + PM2 i,
                  // PM1 と PM2 は，実多項式行列
CPM1 = (PM1,*);   // CPM1 = PM1 + Z(PM1) i,
                  // PM1 は，実多項式行列
CPM1 = (*,PM1);   // CPM1 = Z(PM1) + PM1 i,
                  // PM1 は，実多項式行列
```

は，行列 A の列の順番を逆にした行列である。

　指数とピース行列を同時に使えば，行列の配列から任意のブロック成分からなる部分行列を参照することができる。

　例えば，A が n x n 行列で，AA が n x n*n の行列の配列で，v が n 次元指数とすると，AA(v, A) は，

```
[AA(v(1), A), AA(v(2), A), AA(v(3), A), ..., AA(v(n), A)]
```

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，

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

によって，行列の配列 AA の 4 列，6 列，11 列に行列 BB の 2-4 列が代入される。もっと一般的に，もし v と w が指数なら，

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

は，v と w の成分を行と列としてもつ AA のブロック成分からなる部分行列を意味する。したがって，

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

は，行列の配列 AA の列の順番を逆にした行列である。



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

### 15.3.3　多項式行列の微分と積分

　多項式行列の微分と積分は，関数 derivative() と integral() を用いる。第 2 引数に求める導関数あるいは不定積分の階数を指定するが，階数を省略すると 1 階の導関数または不定積分を求める。

```
CPM2 = integral(CPM1);     // CPM1 の 1 階不定積分
CPM3 = integral(CPM1,2);   // CPM1 の 2 階不定積分
CMP4 = derivative(CMP2);   // CPM2 の 1 階導関数
CMP5 = derivative(CMP3,2); // CPM3 の 2 階導関数
```

### 15.3.4　多項式行列の各要素の係数のシフト

　多項式行列の各要素の係数のを高次方向と低次方向にシフトするには，関数
higher() と lower() ををそれぞれ用いる。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。

```
A = Array([-1, 1, -2, 2]);
idx = find(A < 0);  // idx = [1 3]
A(idx) = -A(idx);   // A = [1 1 2 2]
```

### 15.5.5　成分の代入

左辺の行列の大きさが指定された指数より小さいとき，自動的に拡大され，成分の代入が行われる。ただし，自動拡大機能を繰り返し用いると処理時間が長くなり，メモリの使用効率も悪くなるので，あらかじめ必要な最大の大きさの行列を定義してから，成分の代入を行う方が良い。左辺の行列の指定された指数の大きさと右辺の行列の大きさは一致しなければならない。複素行列には実数と複素数を代入できるが，実行列には実数しか代入できない。有理多項式行列には多項式と有理多項式を代入できるが，多項式行列には多項式しか代入できない。



の結果，

$$
\begin{aligned}
\mathtt{B} &= \begin{bmatrix} \mathtt{eval((s+1)/(s+2),\ A)} & \mathtt{eval((s+3)/(s+4),\ A)} \\ \mathtt{eval((s+5)/(s+6),\ A)} & \mathtt{eval((s+7)/(s+8),\ A)} \end{bmatrix} \\
&= \begin{bmatrix} 0.50000 & 0.16667 & 0.76471 & 0.05882 \\ 0.25000 & 0.75000 & 0.08824 & 0.85294 \\ 0.84375 & 0.03125 & 0.88235 & 0.01961 \\ 0.04688 & 0.89063 & 0.02941 & 0.91176 \end{bmatrix}
\end{aligned}
$$

となるが，配列を用いると

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

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

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

の結果，

$$
\begin{aligned}
\mathtt{B} &= \begin{bmatrix} \mathtt{eval(RM1,\ 1)} & \mathtt{eval(RM1,\ 2)} \\ \mathtt{eval(RM1,\ 3)} & \mathtt{eval(RM1,\ 4)} \end{bmatrix} \\
&= \begin{bmatrix} 0.66667 & 0.75000 & 0.80000 & 0.83333 \\ 0.80000 & 0.83333 & 0.85714 & 0.87500 \\ 0.85714 & 0.87500 & 0.88889 & 0.90000 \\ 0.88889 & 0.90000 & 0.90909 & 0.91667 \end{bmatrix}
\end{aligned}
$$

となる。有理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。

```
CRM2 = higher(CRM1);   // CRM1 の高次方向への 1 回シフト
CRM3 = higher(CRM1,2); // CRM1 の高次方向への 2 回シフト
CRM4 = lower(CRM2);    // CRM2 の低次方向への 1 回シフト
CRM5 = lower(CRM3,2);  // CRM3 の低次方向への 2 回シフト
```

### 15.4.5　有理多項式配列の分子配列と分母配列

有理多項式配列の各要素の分子多項式からなる分子配列と分母多項式からなる分母配列は，関数 Nu() と De() を用い取り出す。

```
s = $;
A = [[1 2]     B = [[5 6]
     [3 4]];        [7 8]];

C = (ONE(2)*s + A) ./ (ONE(2)*s + B);

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

N = Nu(Array(C)); // N = [[s+1  s+2]
                  //      [s+3  s+4]]

D = De(Array(C)); // D = [[s+5  s+6]
                  //      [s+7  s+8]]
```

## 15.5　行列成分の操作

行列の名前の次に行番号と列番号をコンマ “,” で区切って丸括弧（ と ）で囲んで行列の成分を参照する。1 行目の成分を参照する時，行番号を省略できる。ブロック成分の参照には，参照する行の区間と列の区間をコロン : を使って指定する。

指数を使えば，ある行列から任意の成分を取り出し，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できる。

ピース行列[2]を使えば，1 つの大きな行列を別の小さな行列の 2 次元配列と考え，効率良くブロック成分を参照できる。ただし，ピース行列を使用する場合，配列の次数は 0 から始まる。指数とピース行列を同時に利用すれば，ブロック成分からなる行列の配列を操作できる。



### 15.5.1　成分の取り出し

| | |
|---|---|
| M1(1,1) | (1,1) 成分 |
| M1(5) | 成分を (1,1) → (1,Cols(M1)) → (2,1) → (2,Cols(M1)) → ... → (Rows(M1),Cols(M1)) の順番に並べた時，5 番目の成分 |
| M1(3:5) | 3-5 番目成分からなるベクトル |
| M1(1,:) | 第 1 行ベクトル |
| M1(:,1) | 第 1 列ベクトル |
| M1(1:2,2:3) | 1-2 行，2-3 列のブロック成分 |
| M1(2:3,4) | 2-3 行，4 列成分よりなる列ベクトル |
| M1(2:3,:) | 2-3 行成分よりなるブロック成分 |
| M1(2:3,2:) | 2-3 行，2 列以降のブロック成分 |
| M1(4,2:3) | 4 行，2-3 列成分よりなる行ベクトル |
| M1(:,2:3) | 2-3 列成分よりなるブロック成分 |
| M1(2:,2:3) | 2 行以降，2-3 列のブロック成分 |

### 15.5.2　ブロック成分の取り出し

M2 を M1 のピース行列とする。

[2] 行列の配列参照

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

RM2 = [[(s+3)/(s+2)  (s+5)/(s+4)]
       [(s+7)/(s+6)  (s+1)/(s+8)];

CRM1 = (RM1, RM2);

CRM2 = [[((1,1)*s+(1,3))/(s+2) ((1,1)*s+(3,5))/(s+4)]
        [((1,1)*s+(5,7))/(s+6) ((1,1)*s+(7,1))/(s+8)]];
```

### 15.4.2　有理多項式行列の評価

関数 eval() を用いて，有理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，有理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)]
       [(s+5)/(s+6)  (s+7)/(s+8)]];

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

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



```
M1(1,1) = 1.0;       (1,1) 成分に代入する
M1(2) = 1.0;         (1,2) 成分に代入する
                     (1,i) 成分の指定する時，1 を省略できる
CM1(1,1) = (1, 2);   (1,1) 成分に代入する
M(3:5) = M2;         3-5 番目成分に代入する
M(1,:) = M2;         1 行に代入する
M(:,2) = M2;         2 列に代入する
M(3:) = M2;          3 番目以降に代入する
M1(1:2,1:2) = M2;    1-2 行，1-2 列に代入する
M1(2:,3:) = M2;      (2,3) が左上に一致するように代入する
M1(2,3:) = M2;       (2,3) が左上に一致するように代入する
M1(2:,3) = M2;       (2,3) が左上に一致するように代入する
M1(1,2:3) = M2;      1 行，2-3 列に代入する
M1(:,2:3) = M2;      2-3 列に代入する
M1(2:3,1) = M2;      2-3 行，1 列に代入する
M1(2:3,:) = M2;      2-3 行に代入する
```

### 15.5.6　ブロック成分の代入

左辺の行列の大きさが指定された指数より小さいとき，自動的に拡大され，成分の代入が行われる。ただし，自動拡大機能を繰り返し用いると処理時間が長くなり，メモリの使用効率も悪くなるので，あらかじめ必要な最大の大きさの行列を定義してから，成分の代入を行う方が良い。左辺の行列の指定された指数の大きさと右辺の行列の大きさは一致しなければならない。複素行列には実数と複素数を代入できるが，実行列には実数しか代入できない。有理多項式行列には多項式と有理多項式を代入できるが，多項式行列には多項式しか代入できない。

M2 を M1 のピース行列とする。

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

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



```
M1(IDX2)          M1 の IDX2 成分からなる部分行列
M1(1,IDX2)        M1 の 1 行 IDX2 列からなる部分行列
M1(IDX1,IDX2)     M1 の IDX1 行 IDX2 列からなる部分行列
M1(IDX1,1)        M1 の IDX1 行 1 列からなる部分行列
M1(IDX1,1:2)      M1 の IDX1 行 1~2 列からなる部分行列
M1(1:2,IDX2)      M1 の 1~2 行 IDX2 列からなる部分行列
MM(IDX1,IDX2,M)   MM のピース行列 M による (IDX1+1) 行
                  (IDX2+1) 列からなる部分行列
MM(0,IDX2,M)      MM のピース行列 M による 1 行
                  (IDX2+1) 列からなる部分行列
MM(IDX1,0,M)      MM のピース行列 M による (IDX1+1) 行
                  1 列からなる部分行列

M1(IDX2)          = M2;   M1 の IDX2 成分に M2 を代入
M1(1,IDX2)        = M2;   M1 の 1 行 IDX2 列に M2 を代入
M1(IDX1,IDX2)     = M2;   M1 の IDX1 行 IDX2 列に M2 を代入
M1(IDX1,1)        = M2;   M1 の IDX1 行 1 列に M2 を代入
M1(IDX1,1:2)      = M2;   M1 の IDX1 行 1~2 列に M2 を代入
M1(1:2,IDX2)      = M2;   M1 の 1~2 行 IDX2 列に M2 を代入
MM(IDX1,IDX2,M)   = M2;   MM の (IDX1+1) 行 (IDX2+1) 列に
                          ピース行列 M を使って M2 を代入
MM(0,IDX2,M)      = M2;   MM の 1 行 (IDX2+1) 列に
                          ピース行列 M を使って M2 を代入
MM(IDX1,0,M)      = M2;   MM の (IDX1+1) 行 1 列に
                          ピース行列 M を使って M2 を代入
```

### 15.5.10　行列の配列 (ピース行列の利用)

行列は，実数，複素数，多項式，有理多項式の 1 次元または 2 次元の配列として考えることができる。同じ大きさの行列を成分とする配列は，ピース行列を使って記述する。大きさの異なる行列を成分とする配列を記述するた

```
CM3 = (M1,M2); // CM3 = M1 + M2*i;     (M1,M2:実行列)
CM3 = (M1,*);  // CM3 = M1 + Z(M1)*i; (M1:実行列)
CM3 = (*,M1);  // CM3 = Z(M1)*i + M1; (M1:実行列)
```

複素行列の実部と虚部は，関数 Re() と Im() を用いてとり出すことができる。

```
M1 = Re((M1,M2));  // 実部行列 = Re(複素行列)
M2 = Im((M1,M2));  // 虚部行列 = Im(複素行列)
```

### 15.2.7　行列を成分とする行列

行列の式を行列の成分として用いることができる。ただし，それぞれの式の計算結果の行列の大きさは，行列の成分として適当でなければならない。

```
M1 = [[1 2][3 4]];  M2 = [5 6]';  M3 = [7 8];
M4 = [9];  M5 = [[1 0][0 1]];  M6 = [1];

M7 = [[ M1   , -M2*M6\M2#]
      [-M5   , -M1#      ]];
```

## 15.3　多項式行列

### 15.3.1　多項式行列の記述方法

多項式行列は，多項式を成分とする行列である。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。



| 表 記 | 意 味 |
|---|---|
| 行列 1 .< 行列 2 | 各成分の比較結果 (0\|1) の配列 |
| 行列 1 .> 行列 2 | 各成分の比較結果 (0\|1) の配列 |
| 行列 1 .<= 行列 2 | 各成分の比較結果 (0\|1) の配列 |
| 行列 1 .>= 行列 2 | 各成分の比較結果 (0\|1) の配列 |
| 行列 1 .== 行列 2 | 各成分の比較結果 (0\|1) の配列 |
| 行列 1 .!= 行列 2 | 各成分の比較結果 (0\|1) の配列 |
| 行列 1 .< 配列 1 | 各成分の比較結果 (0\|1) の配列 |
| 行列 1 .> 配列 1 | 各成分の比較結果 (0\|1) の配列 |
| 行列 1 .<= 配列 1 | 各成分の比較結果 (0\|1) の配列 |
| 行列 1 .>= 配列 1 | 各成分の比較結果 (0\|1) の配列 |
| 行列 1 .== 配列 1 | 各成分の比較結果 (0\|1) の配列 |
| 行列 1 .!= 配列 1 | 各成分の比較結果 (0\|1) の配列 |
| 配列 1 .< 行列 1 | 各成分の比較結果 (0\|1) の配列 |
| 配列 1 .> 行列 1 | 各成分の比較結果 (0\|1) の配列 |
| 配列 1 .<= 行列 1 | 各成分の比較結果 (0\|1) の配列 |
| 配列 1 .>= 行列 1 | 各成分の比較結果 (0\|1) の配列 |
| 配列 1 .== 行列 1 | 各成分の比較結果 (0\|1) の配列 |
| 配列 1 .!= 行列 1 | 各成分の比較結果 (0\|1) の配列 |

```
M1 = [];               // 0 x 0 の行列
M1 = [1 2 3];          // スペースで区切る
M1 = [1 2 , -3];       // -単項演算子をカンマで区切る
M1 = [1 2 (-3)];       // -単項演算子を括弧で囲む
CM1 = [(1,2), (3,4)]; // 複素行ベクトル
```

### 15.1.3　列ベクトル

転置を意味する” ’ ”を用いて列ベクトルを記述する。

```
M1 = [1 2 3]';          // スペースで区切る
M1 = [1 2 , -3]';       // -単項演算子をカンマで区切る
M1 = [1 2 (-3) ]';      // -単項演算子を括弧で囲む
CM1 = [(1,2), (3,4)]'; // 複素列ベクトル
```

### 15.1.4　行列

行列は，行ベクトルを並べて記述する。

```
  M1 = [[1 2 3][4,-5 6][7 (-8), -9]];

  M1 = [[1  2     3]    CM1 = [[(1,2) (3,4)]
        [4 ,-5    6]           [(5,6) (7,8)]];
        [7 (-8), -9]];
```

## 15.2　いろいろな行列

いろいろな種類の行列を簡単に記述するための方法がたくさん用意されている。



```
Marix M1, V1;          V1 はベクトル
Integer p;

Real max(M1);          最大成分
Real min(M1);          最小成分
List maximum(M1);      行列の最大成分とその位置
                       {r1, i, j} = maximum(M1);
                       r1 == M1(i,j)
List maximum(V1);      ベクトルの最大成分とその位置
                       {r1, i} = maximum(V1);
                       r1 == V1(i)
List minimum(M1);      行列の最小成分とその位置
                       {r1, i, j} = minimum(M1);
                       r1 == M1(i,j)
List minimum(V1);      ベクトルの最小値とその位置
                       {r1, i} = minimum(V1);
                       r1 == V1(i)
Real maxsing(M1);      最大特異値
Real minsing(M1);      最小特異値
Real det(M1);          実行列の行列式
Real trace(M1);        実行列のトレース
Real frobnorm(M1);     (実/複素) 行列のフロベニウスノルム
                       (ユークリッド ノルム)
Real infnorm(M1);      (実/複素) 行列の最大ノルム
Real norm(M1, p);
                       (実/複素) 行列の ノルム
                       M1 が行列の時，
                       p = 1: 1-ノルム
                       p = 2: 最大特異値
                       M1 がベクトルの時，p-ノルム
```

```
                          (真 = 1, 偽 = 0)

Rational floor(R1)        係数の最低値の整数への丸め
Rational ceil(R1)         係数の最高値の整数への丸め
Rational fix(R1)          係数の 0 方向の整数への丸め
Rational round(R1)        係数の最も近い整数への丸め
Rational round2z(R1)      係数の 0 への丸め (tol = EPS)
Rational round2z(R1, tol) 係数の絶対値が tol より小なら 0
Rational pow(R1, i1)      R1^i1
Rational trans(R1)        R1'
Rational conj(R1)         R1 の共役
Rational conjtrans(R1)    R1#
Rational inv(R1)          R1~
```



```
Matrix  kernel(M1);           零空間 (カーネル)
Matrix  pseudoinv(M1);        擬似逆行列
Matrix  pseudoinv(M1, tol);   擬似逆行列
                              tol: 許容誤差
Matrix  abs(M1);              各成分の abs()



Matrix M1;
Real tol;
Integer i1;

Matrix conj(M1);          共役行列
Matrix trans(M1);         転置行列      M1'
Matrix conjtrans(M1);     共役転置行列  M1#
Matrix inv(M1);           逆行列        M1~
Matrix pow(M1, i1);       累乗        M1^i1

Matrix ceil(M1);          各成分の ceil() の結果
Matrix floor(M1);         各成分の floor() の結果
Matrix fix(M1);           各成分の fix() の結果
Matrix round(M1);         各成分の round() の結果
Matrix round2z(M1);       各成分の round2z() の結果
Matrix round2z(M1, tol);  各成分の round2z() の結果

Matrix M1, M2;
Integer k, m, n;
Index idx;

Matrix  vec2diag(M1);         縦ベクトルから対角行列
Matrix  vec2diag(M1, k);
```

```
CR1 = (R1,R2); // CR1 = R1 + R2*i; (R1, R2: 実有理多項式)
CR1 = (R1,*);  // CR1 = R1 + 0*i;  (R1: 実有理多項式)
CR1 = (*,R1);  // CR1 = 0  + R1*i; (R1: 実有理多項式)
```

複素有理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。

```
R1 = Re((R1,R2));  // 実部有理多項式 = Re(複素有理多項式)
R2 = Im((R1,R2));  // 虚部有理多項式 = Im(複素有理多項式)
```

有理多項式の実部と虚部は以下の方法で計算される。

$$\mathrm{Re}\left(\frac{\mathrm{a}+\mathrm{b}i}{\mathrm{c}+\mathrm{d}i}\right)=\frac{\mathrm{ac}+\mathrm{bd}}{\mathrm{c}^2+\mathrm{d}^2},\quad \mathrm{Im}\left(\frac{\mathrm{a}+\mathrm{b}i}{\mathrm{c}+\mathrm{d}i}\right)=\frac{\mathrm{bc}-\mathrm{ad}}{\mathrm{c}^2+\mathrm{d}^2}$$

ただし，a, b, c, d は実多項式である。

$$\mathrm{Re}\left(\frac{\mathrm{a}+\mathrm{b}i}{\mathrm{c}+\mathrm{d}i}\right)=\frac{(\mathrm{a}+\mathrm{b}i)(\mathrm{c}-\mathrm{d}i)}{(\mathrm{c}+\mathrm{d}i)(\mathrm{c}-\mathrm{d}i)}=\frac{\mathrm{ac}+\mathrm{bd}}{\mathrm{c}^2+\mathrm{d}^2}+\frac{\mathrm{bc}-\mathrm{ad}}{\mathrm{c}^2+\mathrm{d}^2}i$$

## 14.3　有理多項式の分子多項式と分母多項式

有理多項式の分子多項式と分母多項式は，関数 Nu() と De() を用いて取り出すことができる。

```
P1 = Nu(P1 / P2);  // 分子多項式 = Nu(有理多項式)
P2 = De(P1 / P2);  // 分母多項式 = De(有理多項式)
```



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

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

T: 時間の系列

## 13.5　多項式の微分と積分

多項式の微分と積分は，関数 derivative() と integral() を用いる。第 2 引数に求める導関数 (derivative) あるいは不定積分 (indefinite integral) の階数を指定するが，階数を省略すると 1 階の導関数または不定積分を求める。

```
s = $;
P1 = 3*s^2 + 4*s + 5;
P2 = integral(P1);      // 2 次の実多項式の 1 階不定積分
P2 = integral(P1,2);    // 2 次の実多項式の 2 階不定積分
P3 = derivative(P2);    // 2 次の実多項式の 1 階導関数
P3 = derivative(P2,2)   // 2 次の実多項式の 2 階導関数
```

## 13.6　多項式の係数のシフト

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 = $;
P1 = 3*s^2 + 4*s + 5;
P2 = higher(P1);     // 係数を高次方向に 1 回シフト
P2 = higher(P1,2);   // 係数を高次方向に 2 回シフト
P3 = lower(P2);      // 係数を低次方向に 1 回シフト
P3 = lower(P2,2);    // 係数を低次方向に 2 回シフト
```

## 13.7　多項式の比較

関係演算子 ==と!= を使って，つの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。



```
                Real dtsav, Real dtmin);
               : dtmax=abs(t1-t0)/5
List Ode45Auto(Real t0, Real t1, 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。
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} = Ode45Auto();
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 Ode45Hybrid(Real t0, Real t1, Real dt, Matrix x,
        void dx())                : h=4.0E-2, dtsav=1.0E-3
List Ode45Hybrid(Real t0, Real t1, Real dt, Matrix x,
        void dx(), void input()) : h=4.0E-2, dtsav=1.0E-3
List Ode45Hybrid(Real t0, Real t1, Real dt, Matrix x,
        void dx(), void input(), Real h) : dtsav=1.0E-3
List Ode45Hybrid(Real t0, Real t1, Real dt, Matrix x, void dx(),
        void input(), Real h, Real dtsav)
```

RKF45 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} = Ode45Hybrid();
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 qr(M1);
```
QR 分解
```
{Q, R} = qr(A);
Q * R = A;
```
Q: 直交行列 or ユニタリ行列
R: 上三角行列

```
List qr_p(M1);
```
QR 分解 (置換付)
```
{Q, R, P} = qr_p(A);
Q * R = A * P;
```
Q: 直交行列 or ユニタリ行列
R: 上三角行列
P: 置換行列

```
List qz(M1, M2);
```
QZ 分解
```
{AA, BB, Q, Z, VEC} = qz(M1, M2)
Q# * M1 * Z = AA;
Q# * M2 * Z = BB;
val = diag2vec(AA) ./ diag2vec(BB);
M1 * VEC = M2 * VEC * vec2diag(val);
```
AA: 上三角行列
BB: 上三角行列
Q: 直交行列 or ユニタリ行列
Z: 直交行列 or ユニタリ行列
VEC: 一般固有ベクトル
val: 一般固有値

| 表 記 | 意 味 |
|---|---|
| 文字列 1 == 文字列 2 | 文字列 1 と文字列 2 は等しい |
| 文字列 1 != 文字列 2 | 文字列 1 と文字列 2 は等しくない |

## 12.4　文字列から指定文字の位置を調べる

ある文字列中に指定した 1 文字が存在するか，存在するなら何文字目であるかを関数 strchr() にって調べられる。関数 strchr() の第一引数は対象となる文字列，第二引数は調べたい 1 文字を指定する。もし，対象文字列に指定した文字が存在しない場合，関数 strchr() は 0 を返す。例えば，

```
str = "Hello world";
id1 = strchr(str, "w");
id2 = strchr(str, "A");
```

を実行すると，id1 = 7，id2 = 0 の結果を得る。

## 12.5　文字列の評価 (matx)

関数 eval() を用いて，MATX の任意のコ- ドからなる文字列を評価することができ，関数の戻り値は評価結果と一致する。すなわち，関数の戻り値の型は，評価する文字列に依存する。式以外のコ- ドを評価したときは，常に整数 0 を返す。例えば，

```
Real a, b, c;
Matrix A, B, C;

a = 1.0; b = 2.0;
A = [[1 2][3 4]]; B = [[5 6][7 8]];

c = eval("a + b");
C = eval("A + B");
```



```
                                行毎の平均値からなる縦ベクトル
Array std_col(A1);              (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
Array std_row(A1);              (縦・横) ベクトルの成分の標準偏差
                                行毎の標準偏差からなる縦ベクトル
Array cumsum_col(A1);           列毎の累積和からなる横ベクトル
Array cumsum_row(A1);           行毎の累積和からなる縦ベクトル
Array cumprod_col(A1);          列毎の累積積からなる横ベクトル
Array cumprod_row(A1);          行毎の累積積からなる縦ベクトル


Real d1, d2;
Integer split;

Array logspace(d1, d2)
                    対数空間ベクトル (split=50)
Array logspace(d1, d2, split)
                    対数空間ベクトル
Array linspace(d1, d2)
                    線形空間ベクトル (split=100)
Array linspace(d1, d2, split)
                    線形空間ベクトル


Array A1, A2;
Integer i1,k;
Real r1;

Array vec2diag(A1);             縦ベクトルから対角配列
Array vec2diag(A1, k);
                                縦ベクトルから対角配列
                                (k だけ主対角より上)
```

虚数単位 (Imaginary Unit) は以下の 2 つの方法で定義することができる。

```
i = (0, 1);
i = sqrt(-1);
```

## 11.2　複素数の比較

関係演算子 ==, != を使って，2 つの複素数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 |
|---|---|
| c1 == c2 | 複素数 c1 と複素数 c2 は等しい |
| c1 != c2 | 複素数 c1 と複素数 c2 は等しくない |

## 11.3　複素数関数

複素数関数の一覧を以下に示す。

```
Complex c1;
Real tol;

Integer finite(c1);          有限関数
                             もし c1 が有限値なら 1
                             無限大あるいは NaN なら 0
Integer isnan(c1);           もし c1 が NaN なら 1
                             それ以外なら 0
Real    abs(c1);             絶対値
Real    arg(c1);             偏角
Comlex  max(c1, c2);         最大値
Comlex  min(c1, c2);         最小値
```



# 第16章　リスト

リストは，任意のクラスの値を成分として持つことができる 1 次元配列である。リストを利用して，複数個の戻り値を同時に返す関数を定義することができる。

[ 宣言 ]

```
List a, b, c;
```

[ 例 ]

```
{4, (3,4), [1 2]}, {"Hello", s+1, (s+1)/(s+2)}
```

## 16.1　リストの記述方法

リストを記述するには，成分となる値をコンマ “,” で区切って書き並べ，ブレース { と } で囲む。

```
x = {4, (3,4), [1 2]};
```

により，成分が整数，複素数，行列であるリスト x に代入される。リストの成分には任意の式を書くことができる。

```
x = {sqrt(3.0), ((1.0+2.0)*4.0,:), [1 2] + [3 4]};
```

により，成分が実数，複素数，行列であるリストが x に代入される。関数

```
Real ceil(r1);                  最高値の整数への丸め関数
Real fact(r1);                  階乗
Real floor(r1);                 最低値の整数への丸め関数
Real fix(r1);                   0 方向の整数への丸め関数
Real round(r1);                 最も近い整数への丸め関数
Real round2z(r1)                ゼロへの丸め (tol = EPS)
Real round2z(r1, tol)           絶対値が tol より小なら 0
Real sgn(r1);                   符合関数 (r1 > 0.0, 1.0;
                                          r1 = 0.0, 0.0;
                                          r1 < 0.0, -1.0)
Integer finite(r1);             有限関数
                                もし r1 が有限値なら 1
                                無限大あるいは NaN なら 0
Integer isnan(r1);              もし r1 が NaN なら 1
                                それ以外なら 0

Real pow(r1, r2)                r1^r2
Real trans(r1)                  r1'
Real conj(r1)                   r1 の共役
Real conjtrans(r1)              r1#
Real inv(r1)                    r1~


Integer fd;
String file, format;

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
```



# 第17章　例題

この章では，いくつかの例題プログラムを紹介する。連続時間線形システムのリカッティ方程式と離散時間線形システムのリカッティ方程式の解を求めるプログラム，線形システムの伝達関数と伝達関数行列を求めるプログラム，線形システムのボードとナイキスト周波数応答を計算するプログラム，連続時間システムの状態フィードバック制御及び動的フィードバック制御のシミュレーションを行なうプログラムを紹介する。

```
   010(8),  0100(64),    0xA(10),
 0x10(16),  0x1c(28),  0xff(255)
```

## 10.2　整数と実数の比較

関係演算子 >, >=, <, <=, ==, != を使って，2 つの整数 (実数) 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 |
|---|---|
| i1 > i2 | 整数 i1 は整数 i2 より大きい |
| i1 >= i2 | 整数 i1 は整数 i2 以上である |
| i1 < i2 | 整数 i1 は整数 i2 より小さい |
| i1 <= i2 | 整数 i1 は整数 i2 以下である |
| i1 == i2 | 整数 i1 と整数 i2 は等しい |
| i1 != i2 | 整数 i1 と整数 i2 は等しくない |
| r1 > r2 | 実数 r1 は実数 r2 より大きい |
| r1 >= r2 | 実数 r1 は実数 r2 以上である |
| r1 < r2 | 実数 r1 は実数 r2 より小さい |
| r1 <= r2 | 実数 r1 は実数 r2 以下である |
| r1 == r2 | 実数 r1 と実数 r2 は等しい |
| r1 != r2 | 実数 r1 と実数 r2 は等しくない |

次の式は-a 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;
```



```
   Func void main()
   {
     Matrix A, B, C, Q, R, P, PP;
     Matrix DRiccati();
05:
     A = [[0 1][-3, 4]];       // システム行列
     B = [0 1]';
     C = [-2, 1];
     Q = C'*C;
10:  R = I(1);

     P = DRiccati(A, B, Q, R); // DRiccati() を呼ぶ
     print P;                  // P を表示
   }
15:
   Func Matrix DRiccati(A, B, Q, R)
     Matrix A, B, Q, R;
   {
     Matrix P, PP;
20:  Real eps;

     P = PP = I(A);          // 解 P の初期値
     eps = 1.0E-7;

25:  // 解が収束するまで繰り返す
     while (frobnorm(P - PP) > eps) {
         PP = P;
         P = Q + A'*P*A - A'*P*B*(R + B'*P*B)~ * B'*P*A;
         P = (P' + P)/2.0;   // P の対称性を回復
30:  }

     return P;               // 解 P を返す
   }
```

List 2:　離散時間リカッティ方程式の解

## 9.2　For ループ, While ループ

```
while (expr) {
    statements;
}
```

では，まず expr が計算される。expr が 0 でないとき，statement が実行され，expr が再度評価される。このサイクルは expr が 0 になるまで続けられ，その時点で実行は statement の次に移る。for 文

```
for (expr1; expr2; expr3) {
    statements;
}
```

は以下に等しい。

```
expr1;
while (expr2) {
    statements;
    expr3;
}
```

文法的には for の三つの要素は式である。一般的には式 1 と式 3 は代入または関数呼び出しであり，式 2 は関係式である。三つのどの部分のどれを省略してもよいが，セミコロンは省力してはならない。テスト部 (式 2) が省略されたら，それは永久に真とみなされる。したがって

```
for (;;) {
    statements;
}
```

は “無限” ループであり，脱出は，他の手段 (例えば break や return) によることになる。



```
Func void main()
{
  Polynomial s;
  Rational G;
  Array w, ga, ph, re, im;
  List Bode(...), Nyquist(...);

  s = $;                          // 多項式変数
  G = (s^2 + 3*s + 4)/(s^3 + 5*s^2 + 6*s + 7);
                                  // 伝達関数
  w = logspace(0.01, 100.0);      // 周波数の範囲
  {ga, ph} = Bode(G, w);          // ゲインと位相
  {re, im} = Nyquist(G, w);       // 実部と虚部

  print [[w][re][im]] >> "nyquist.mat"; // データ保存
  print [[w][ga][ph]] >> "bode.mat";    // データ保存
}

Func List Bode(G, w)
  Rational G;
  Array w;
{
  Complex j;
  Array data, ga, ph;

  j = (0,1);                      // 虚数単位
  data = eval(G, j*w);            // G(j*w)
  ga = 20.0 * log10(abs(data)); // ゲイン [dB]
  ph = arg(data) * 180.0 / PI;  // 位相  [degree]
  return {ga, ph};
}

Func List Nyquist(G, w)
  Rational G;
  Array w;
{
  Complex j;
  Array data;

  j = (0,1);                      // 虚数単位
  data = eval(G, j*w);            // G(j*w)
  return {Re(data), Im(data)};  // 実部と虚部
}
```

List 4:　ボード線図とナイキスト線図

実数，文字列，複素数は，値渡しされ，行列，有理多項式，有理関数，リストは，参照渡しされる。

```
Func void main()
{
    Matrix a;
    void foo();

    a = [1];
    print a;        // a = [1]
    foo(a);
    print a;        // a = [2]
}

Func void foo(b)
    Matrix b;
{
    print b;        // b = [1]
    b = [2];
    print b;        // b = [2]
}
```

## 8.7　return 文

return 文は呼ばれた関数の値を，呼んだプログラムに返すための機構である。void 型以外の関数の場合，return の後ろには任意の式を書いてよい。呼ぶ側の関数は，望むなら戻り値を無視することもできる。ただし，void 型の関数の場合，return の後ろに式を書いてはならず，void 型以外の関数の場合，return の後ろの式を省略できない。

```
return;        // void 関数
return expr;   // void 以外の関数
```



### 17.5.2　状態フィードバック制御のシミュレーション (2)

前と同じ連続システムに対して，4 次のルンゲ- クッタの公式を直接使ってハイブリッド制御 (きざみを小さくすれば，連続時間制御と考えることもできる) のシミュレーションを行なう。

```
Func void main()
{
  Integer i, ii;
  Real dt, t, t1, t2;
  Matrix x, u, controller();
  Array X, U;
  void plant();

  t1 = 0.0;       // 初期時刻
  t2 = 5.0;       // 終端時刻
  dt = 0.01;      // きざみ
  ii = Integer((t2 - t1) / dt);

  x = [1 1 1]';  // 初期状態
  u = [0];

  X = Z(1, ii, x);
  U = Z(1, ii, u);

  t = t1;
  X(0, 0, x) = x;

  for (i = 1; i < ii; i++) {
    u = controller(t, x);
    x = rngkut4(t, x, plant, u, dt);

    U(0, i-1, u) = u;
    X(0, i, x) = x;
    t = t + dt;
  }
  u = controller(t, x);
  U(0, ii-1, u) = u;

  print [[X][U]] >> "cont.mat";
}
```

```
Func ClassName FunctionName(arg1, arg2, arg3, ..., argN)
ClassName1 arg1, arg2;
ClassName2 arg3;
...............
ClassNameM argN;
{
    変数の定義

    複数の文（関数の本体）

    return;       // void 関数
    return value; // void 以外の関数
}
```

## 8.3　複数の値を返す関数

リストを利用して，複数個の戻り値を同時に返す関数を定義することができる。そのような関数は，次の形式で記述する。

```
Integer a, d;
Complex b, f;
Matrix c, e;
List func();
{a, b, c} = func(d, e, f);
```

“a, b, c” は，関数 func() からの複数個の戻り値を表していて，“d, e, f” は，関数 func() の引数を意味する。関数 func() は，複数個の戻り値から構成されるリストを返す。左辺のリストの成分の数が関数の返すリストの成分の数より少ない時，左辺のリストの成分の数だけ関数の返すリストの成分が使用され，それ以上の成分は捨てられる。



```
Func void diff_eqs(DX, t, X, UY)
  Real t;
  Matrix X, DX, UY;
{
  Matrix dxp, dxc, xp, xc, up, uc;

  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]];
}

Func void link_eqs(UY, t, X)
  Real t;
  Matrix UY, X;
{
  Matrix xp, up, yp, xc, yc, uc;

  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 7-2:　動的フィードバック制御のシミュレーション (大域変数使用)

```
Matrix Ap, Bp, Cp; // 制御器のパラメータ
Matrix Ad, Bd, Cd; // 不確定要素のパラメータ
Matrix Ac, Bc, Cc; // 制御器１のパラメータ
Matrix Af, Bf, Cf; // 制御器２のパラメータ
List state_idx;    // 状態の指数のリスト
List input_idx;    // 入力の指数のリスト

Func void main()
{
  Matrix xp0, xd0, xc0, xf0, x0;
  Matrix up0, ud0, uc0, uf0, u0;
  Array TC, XC, UC;

  void diff_eqs(), link_eqs();

  read Ap, Bp, Cp <- "plant";
  read Ad, Bd, Cd <- "uncertainty";
  read Ac, Bc, Cc <- "controller-1";
  read Af, Bf, Cf <- "controller-2";
  read xp0, xd0, xc0, xf0 <- "init-states";
  read up0, ud0, uc0, uf0 <- "init-inputs";

  {x0, state_idx} = VecCont({xp0, xd0, xc0, xf0});
  {u0, input_idx} = VecCont({up0, ud0, uc0, uf0});
  {TC, XC, UC} =
     Ode45Auto(0.0, 20.0, x0, diff_eqs, link_eqs);

  print [[TC][XC][UC]] >> "cont.mat";
}
```

であり，大文字と小文字は区別される。組み込み関数の名前やコマンド名等の予約語は，変数名として使用することができない。今のところ，変数は宣言の中で初期化できない。

```
Integer    i1, i2, i3;   // 整数
Real       r1, r2, r3;   // 実数
Complex    c1, c2, c3;   // 複素数
String     S1, S2, S3;   // 文字列
Polynomial P1, P2, P3;   // 有理多項式
Rational   R1, R2, R3;   // 有理関数
Matrix     M1, M2, M3;   // 行列
Array      A1, A2, A3;   // 配列
Index      I1, I2, I3;   // 指数
List       L1, L2, L3;   // リスト
```

## 7.2　通用範囲に関する規則

MATX プログラムを構成する関数と大域変数は，全てを同時にコンパイル必要はない，プログラムのソース・テキストを数個のファイルに保存しておき，以前にコンパイルしたルーチンをライブラリからロードしてもよい。

変数の名前の通用範囲 (scope) は，その名前が定義されるプログラムの部分である。関数の始めに宣言される局所変数では，通用範囲はその名前が宣言される関数であり，同じ名前の局所変数が別の関数にあってもこれとは関係がない。同じことは関数の引数についてもいえる。それらは実質的には局所変数だからである。

関数の通用範囲は，コンパイルされるべきファイルで宣言された点からそのファイルの終りまで続く。一方，大域変数が定義される前に参照されたり，`main()` 関数の定義されているファイルと別のソース・ファイルで参照されたりすると，`extern` 宣言が必要となる。

大域変数の宣言とその定義の区別は重要である。宣言は変数の性質を指示するものだが，定義は記憶への割り付けも引き起こす。ただし，インタプリタ (matx) では，宣言によって記憶への割り付けも行なわれる。



# 関連図書


[1] K. J. Astrom. Computer aided modeling, analysis and design of control systems – a perspective. *Control Systems Magazine*, pages 4–16, 1983.

[2] J.J. Dongarra, C.B. Moler, J.R. Bunch, and G.W. Stewart. *LINPACK — Users' Guide*. SIAM, Philadelphia, 1979.

[3] George E. Forsythe, Michael A. Malcolm, and Cleve B. Moler. *Computer Methods for Mathematical Computations*. Prentice-Hall, 1977.

[4] B.S. Garbow, J.M. Boyle, J.J. Dongarra, and C.B. Moler. *Matrix Eigensystem Routines – EISPACK Guide Extension*. Springer-Verlag, 1977.

[5] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 1989.

[6] J.E.Potter. Matrix quadratic solution. *SIAM Journal on Appl. Math.*, 14:496–501, 1966.

[7] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice-Hall, 1978.

[8] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language(ANSI C)*. Prentice-Hall, second edition, 1988.

けが表示されるようにできる。また，メインウィンドウやデ- タ管理ウィンドウなどは，アイコン・マネ- ジャによって自由に操作することができる。

```
################################################
#       ICON MANAGER
################################################

SortIconManager # keep them in alphabetic order

IconManagers
{
   "xplot"   "=280x900+2+2" 2
}

IconManagerShow
{
   "xplot"
}

IconifyByUnmapping
{
  "xplot"
}

StartIconified
{
        "XPlot"
        "2D_Info"
        "SemiLog_Info"
        "d1plot-data-1"
        "d1plot-data-2"
        "d1plot-data-3"
        "d2plot-data-1"
        "d2plot-data-2"
        "d2plot-data-3"
        "splot-data-1"
        "splot-data-2"
        "splot-data-3"
}
################################################
```

X ウィンドウ・システムのウィンドウ・マネ- ジャ- として fvwm を使う場合，システムによってはキーボード入力が出来ないことがある。その時は，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 .fwmrc に加える。



# 付 録A　データフォーマット

## A.1　MATデータフォーマット

1. ファイルの第 1 行目は#で始まり，行数 (整数)，列数 (整数) が続き，複素行列のデータの場合には次に大文字の C が続く。
2. データはスペース，タブ，改行のいずれかで区切られる。
3. #で始まる行はコメントと見なされる。
4. 行末にある必要以上のデータは無視される。

実行列

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

をファイルに保存すると

```
# 3 4
  1.0000   5.0000   9.0000
  2.0000   6.0000   10.000
  3.0000   7.0000   11.000
  4.0000   8.0000   12.000
```

となり，複素行列

$$\begin{bmatrix} 1+2i & 3+4i & 5+6i \\ 7+8i & 9+10i & 11+12i \end{bmatrix}$$

### 6.1.9 X ウィンドウを必要としないグラフィック表示

gplot() や mgplot() で呼び出される gnuplot には Tektronix 4010 や hterm のグラフィックモード のドライバが組み込まれているので，Tektronix 4010 や hterm のグラフィックモード をサポート する端末ソフト (kermit, vt98 など) を使えば，X ウィンドウ以外の端末でもグラフィックを表示することができる。 gplot() や mgplot() の出力モード は，変数 mgplot_term によって設定することができる。

```
mgplot_term = "";           // X Window System
mgplot_term = "x11";        // X Window System
mgplot_term = "kc_tek40xx"; // Tektronix 4010 (Kermit)
mgplot_term = "hterm";      // Hterm graphics (Hterm)
```

mgplot_term を “kc_tek40xx” に設定して，グラフィックを表示したとき，gnuplot の終了後に画面が乱れることがある。そのときは，関数 gcls() によって画面をクリアする。

```
gcls();
gcls;       // 引数が 0 個の関数は，括弧を省略できる
```

## 6.2 xplot

xplot[2]を使ってデータをプロットするには以下の関数を使う。

グラフィック関数 xplot() 使うと，自動的に xplot が起動されて，データが 1D キャンバス，2D キャンバス，あるいはセミログキャンバスにプロットされる。したがって，グラフィック関数 xplot() は X-Window System 上でしか使うことができない。

また，グラフィック関数を使うには，MATX を起動する前に，環境変数 DISPLAY にユーザが使っている端末の名前を設定しておく必要がある。

[2]MATX の FTP サーバから入手可能。ただし，対応する OS が限られる。



| データ型 | 名前 | 内容 |
|---|---|---|
| Matrix | cols | 行列の列の数 |
| Integer | dummy | |
| Real Number | dummy | |
| Complex Number | dummy | |
| String | dummy | |
| Polynomial | dummy | |
| Rational | ddegree | 分母の次数 |
| List | dummy | |
| Array | cols | 配列の列の数 |
| Index | cols | 指数の数 |

4. 4 番目の整数は，データの型によって表すものが異なる。

| データ型 | 名前 | 内容 |
|---|---|---|
| Matrix | class | 0(実行列) |
| または | class | 1(複素行列) |
| Array | class | 2(実多項式行列) |
| または | class | 3(複素多項式行列) |
| Index | class | 4(実有理多項式行列) |
| | class | 5(複素有理多項式行列) |
| Integer | dummy | |
| Real Number | dummy | |
| Complex Number | dummy | |
| String | dummy | |
| Polynomial | imagf | 0(実多項式) |
| | imagf | 1(複素多項式) |
| Rational | imagf | 0(実有理多項式) |
| | imagf | 1(複素有理多項式) |
| List | dummy | |

### 6.1.6 ラベル，タイトル，グリッド，テキスト，他のコマンド

次の関数を使ってラベル，タイトル，グリッド，テキストを描くことができる。

関数 gplot_grid() や関数 mgplot_grid() の引数 onoff に 1 を設定するとグリッド線が表示され，0 を指定するとグリッド線が表示されない。また，引数 onoff を指定しないと，オンオフがトグルとなる。

関数 gplot_key() や関数 mgplot_key() の引数 onoff に 1 を設定すると線のタイトル (キー) が表示され，0 を指定すると線のタイトル (キー) が表示されない。また，引数 onoff を指定しないと，オンオフがトグルとなる。

関数 gplot_cmd() や mgplot_cmg() を使って任意の (gnuplot の) コマンドを gnuplot に送ることができる。複数のコマンドを一度に送るには，コマンドを改行 “\n” で区切ればよい。関数 gplot_replot() や mgplot_replot() を呼ぶと，ウィンドウが再表示される。関数 gplot_options() や mgplot_options() によって，gnuplot の起動時のオプションを指定できる。例えば，mgplot_options(1, "-geometry 400x300") のように指定する。



| | |
|---|---|
| 実有理多項式配列 | rows×cols 個の実有理多項式。(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))。 |
| 複素有理多項式配列 | rows×cols 個の複素有理多項式。(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))。 |
| 指数 | cols 個の精度に応じた実数 (4 バイトまたは，8 バイト)。 |

表 6.3: グラフィック関数 3(gnuplot)

| 関数名/引数 | 機能 |
|---|---|
| gplot_semilogx(X, Y)<br>gplot_semilogx(X, Y, titles)<br>gplot_semilogx(X, Y, titles, cmd)<br>Array X, Y; List titles; List cmds; | セミログ x–y プロット<br>(x 軸 ログ) |
| gplot_semilogy(X, Y)<br>gplot_semilogy(X, Y, titles)<br>gplot_semilogy(X, Y, titles, cmd)<br>Array X, Y; List titles; List cmds; | セミログ x–y プロット<br>(y 軸 ログ) |
| gplot_loglog(X, Y)<br>gplot_loglog(X, Y, titles)<br>gplot_loglog(X, Y, titles, cmd)<br>Array X, Y; List titles; List cmds; | ログログ x–y プロット |
| mgplot_semilogx(win, X, Y)<br>mgplot_semilogx(win, X, Y, titles)<br>mgplot_semilogx(win, X, Y, titles, cmds)<br>Integer win; Array X, Y;<br>List titles; List cmds; | セミログ x–y プロット<br>(x 軸 ログ) |
| mgplot_semilogy(win, X, Y)<br>mgplot_semilogy(win, X, Y, titles)<br>mgplot_semilogy(win, X, Y, titles, cmds)<br>Integer win; Array X, Y;<br>List titles; List cmds; | セミログ x–y プロット<br>(y 軸 ログ) |
| mgplot_loglog(win, X, Y)<br>mgplot_loglog(win, X, Y, titles)<br>mgplot_loglog(win, X, Y, titles, cmds)<br>Integer win; Array X, Y;<br>List titles; List cmds; | ログログ x–y プロット |



## B.2 メモリの状態遷移

メモリの状態遷移の様子を次の例題を使って示す。

```
Func Matrix afo()
{
  Matrix A, B, C, D, E;

  A = [[1 2][3 4]];       // (1)
  B = [[5 6][7 8]];       // (2)
  C = A + B;              // (3)
  D = (A + B) * (A - B); // (4)
  E = (C + D) * (C - D); // (5)
  return E;               // (6)
}
```

プログラムが実行される前のメモリの状態は図 B.2 のようになっている。ただし，a, b, ..., k は $2 \times 2$ の大きさの行列のメモリを表している。関数が実

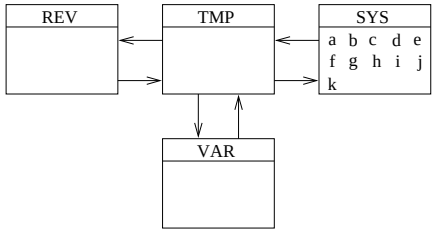



図 B.2: 行列のメモリ管理 (1)

行されて，式 (1) の右辺が評価されたときのメモリの状態は図 B.3 のようになっている。式 (1) の右辺が左辺に代入されたときのメモリ状態は図 B.4 のようになる。同様に式 (2) の代入の後，メモリ状態は図 B.5 のようになる。式 (3) の右辺が評価されたときのメモリの状態は図 B.6 のようになっている。式 (3) の右辺が左辺に代入されたときのメモリ状態は図 B.7 のようになる。式 (4) の右辺が評価されたときのメモリの状態は図 B.8 のようになっている。d, e, f には，それぞれ $(A+B)$, $(A-B)$, $(A+B)*(A-B)$ の計

```
% setenv DISPLAY hostname:0.0
```

また，環境変数 MATXGNUPLOT に使用したい gnuplot を指定できる。

```
(UNIX の場合の例)
% setenv MATXGNUPLOT /usr/local/MaTX/bin/gnuplot

(Windows の場合の例)
% set MATXGNUPLOT=c:\app\MaTX\bin\gnuplot
```

関数 gplot() と関数 mgplot() の違いは，gplot() では 1 個のウィンドウ (カレントウィンドウ) にしかデータをプロットできないのに対して，mgplot() を使えば，複数の任意のウィンドウにデータをプロットできることである。つまり，データをプロットしたいウィンドウの番号を第 1 引数に指定する。

関数 mgplot() によって最後にデータをプロットされたウィンドウは，カレントウィンドウとなるので，そのウィンドウにデータをプロットするには，ウィンドウ番号を省略して関数 gplot() を使えばよい。

ただし，関数 gplot() や関数 mgplot() を使うと，ウィンドウ上の線がクリアされて，新しいデータのみがプロットされる。古いデータに新しいデータを重ねてプロットするには，後で述べる関数 greplot() や関数 mgreplot() を使う。

### 6.1.1　基本的なプロット

もし Y が横ベクトルなら，gplot(Y)(mgplot(win, Y)) によって Y の成分とその成分の指数を対応させた線形プロットが作成される。たとえば，ある実数の集合 $\{0., .48, .84, 1., .91, .6, .14\}$ をプロットするには，その実数をベクトルの中に入れて gplot()(mgplot()) 関数を次のように実行する。



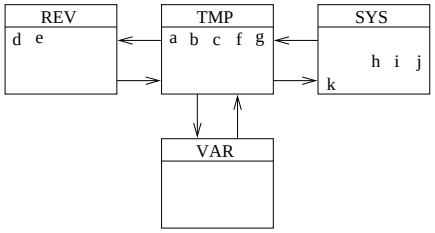



図 B.12: 行列のメモリ管理 (11)

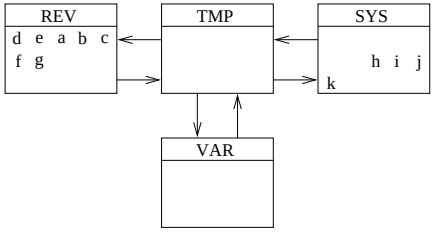



図 B.13: 行列のメモリ管理 (12)

ることになる。

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

            (  3)         (  4)         (  5)         (  6)
( 12)   3.227756E-01  8.225142E-01  1.384097E-02  9.237073E-01
( 13)   1.713277E-01  1.292042E-01  1.589902E-01  2.927582E-01
( 14)   7.294220E-02  4.309787E-01  7.498257E-01  1.519544E-01
( 15)   9.988188E-02  7.323241E-01  5.288297E-01  5.194745E-01
( 16)   8.113680E-01  8.804363E-01  1.127707E-03  1.070992E-01
( 17)   9.994860E-01  8.759036E-03  6.378548E-02  9.676939E-01
( 18)   5.361216E-01  9.766774E-01  2.334755E-01  4.185591E-01
( 19)   7.648637E-01  4.797411E-01  5.563773E-01  1.208822E-02
( 20)   2.827639E-01  1.078266E-01  2.987334E-01  5.287705E-01
( 21)   1.882765E-01  9.792556E-01  5.031973E-01  1.600592E-01
( 22)   2.117229E-01  9.438229E-01  6.220121E-01  2.136757E-01
( 23)   1.647209E-01  7.800235E-01  7.095231E-01  1.848203E-01
( 24)   8.080054E-01  7.973468E-01  9.864512E-01  6.998894E-01
( 25)   4.115652E-01  5.074672E-01  8.462741E-01  8.248976E-01
( 26)   6.695332E-02  4.392214E-01  8.449764E-01  3.010135E-01
```

図 5.4: 行列エディタ (虚部行列)

## 5.3　キーバインディング

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。領域のコピーを行なうには，まず，コピーしたい領域の左上に移動してマークを付ける。次に，領域の右下に移動して領域の獲得を行ない，最後にコピー先の左上に移動してペーストする。

## 5.4　行列エディタ (mated)

行列エディタ (mated) は，シェルのコマンドラインから起動することができ，MAT フォー マットの行列ファイルを入力・編集することができる．

```
% mated A.mat B.mat C.mat
```



## C.1　和 (Addition)

| | |
|---|---|
| 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)<br>+ 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)<br>+ 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)<br>+ 複素有理多項式 (行列\|配列) |
| (行列\|配列\|指数) | = (行列\|配列\|指数)<br>.+ (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)<br>.+ (行列\|配列\|指数) |
| 文字列 | = 文字列 + 文字列 |

## C.5 逆 (Inverse)

```
実数           = 実数 ~ = 1.0 / 実数
複素数         = 複素数 ~ = 1.0 / 複素数
実行列         = 実行列 ~
複素行列       = 複素行列 ~
実配列         = 実配列 ~
複素配列       = 複素配列 ~
有理多項式配列 = 多項式配列 ~
有理多項式配列 = 有理多項式配列 ~
実有理多項式   = 実多項式 ~
複素有理多項式 = 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 ~
文字列         = 文字列 ~
```

```
実行列         = 実行列 .~
複素行列       = 複素行列 .~
有理多項式行列 = 多項式行列 .~
有理多項式行列 = 有理多項式行列 .~
実配列         = 実配列 .~
複素配列       = 複素配列 .~
有理多項式配列 = 多項式配列 .~
有理多項式配列 = 有理多項式配列 .~
```

## C.6 商 (Division)

整数の割算では小数部は切捨てられる。行列と行列の商 “X = A \ B” は，“A * X = B” の解で，“X = A / B” は，“X * B = A” の解である。A が正方行列の時，LU 分解を用いて解が求められ，A が正方行列でない時，最小二乗解が求められる。

```
OP3 = OP1 / OP2 ----> (OP3 = OP1 * OP2~)
OP3 = OP1 \ OP2 ----> (OP3 = OP1~ * OP2)
```

```
save M1, r1, P1, [[1 2][3 4]], "Hello", "file.mm";
save "file"; // 全変数は"file.mm"に保存される。(matx のみ)
save;        // 全変数は"MaTX.mm"に保存される。(matx のみ)
```

インタプリタ (matx) の場合，save コマンドでファイルに保存したデータを load コマンドで読み込むことができる。ただし，データは広域変数として読み込まれるので，関数の内側で load コマンドを実行してはならない。拡張子を省略すると，拡張子は mm となる。

```
load "filename1.mm", "filename2.mm";
```

## 4.3　ファイル入出力

C 言語と同様にファイルをオープン fopen() し，データをファイルに出力 (fprintf()) したり，データをファイルから読み込む (fscanf()) こともできる。ファイルを fopen() でオープンしたあとは，ファイルディスクリプタによってファイルを指定する。不要になったファイルは fclose() によって閉じる。 ファイルディスクリプタを用いたファイル入出力の例を次に示す。

```
  if ((id1 = fopen("afo", "w")) < 0) {
      error("Can't open %s", "afo");
  }
  fprintf(id1, "%d %lf %s", 10, 3.1415, "Hello");
  fclose(id1);

  if ((id2 = fopen("afo", "r")) < 0) {
      error("Can't open %s", "afo");
  }
  {a, b, c} = fscanf(id2, "%d %lf %s");
  fclose(id2);

  print a, b, c;
}
```



```
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.9　複素共役転置 (Complex Conjugate Transpose)

整数，実数，実 (多項式，有理多項式) は複素共役転置によって変わらない。実 (行列，配列) の複素共役転置は，転置と同じである。

```
整数                          = 整数' = 整数#
実数                          = 実数' = 実数#
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#
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#
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)#
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)#
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)#
```

```
load "filename1.mm", "filename2.mm";
load;

load "/usr/local/tmp/file.mm";  // UNIX の絶対パス
load "c:/tmp/file.mm";          // DOS の絶対パス (1)
load "c:\\tmp\file.mm";         // DOS の絶対パス (2)
```

DOS の絶対パスの区切り文字はスラッシュ / でもバックスラッシュ \ のどちらでもよい。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。

## 4.2　データのファイル入出力

データをファイルに保存したりファイルから読み込むための方法が，いくつかある。

### 4.2.1　行列データのファイル入出力

行列をファイルへ書き込むには，print コマンドを用い出力先を記号>>によってファイルにリダイレクトする。ファイル名の拡張子は，自由に決めることができるが，mat を用いることが推奨される。行列データは，MAT データフォーマットでファイルに書き込まれる。

```
print A >> "filename.mat";
```

行列をファイルから読み込むには，read コマンドを用い入力先を記号<<によってファイルにリダイレクトする。ファイル名の拡張子は，自由に決めることができるが，mat を用いることが推奨される。

```
read A << "filename.mat";
```



次の式は -a 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;
```

## C.13　論理演算 (Logical Operation)

論理演算子&&の優先度は||より高く，しかも両者は関係演算子や等値演算子より低い。論理値 0 は偽であり，0 以外の値は真である。単項否定演算子!は非零すなわち真である被演算数の値を 0 にし，0 すなわち偽である被演算数の値を 1 にする。

| | |
|---|---|
| 整数 1 && 整数 2 | 整数 1 と整数 2 が 0 でない時 1（論理積） |
| 整数 1 \|\| 整数 2 | 整数 1 か整数 2 が 0 でない時 1（論理和） |
| ! 整数 1 | 整数 1 が 0 の時 1（否定） |
| 実数 1 && 実数 2 | 実数 1 と実数 2 が 0.0 でない時 1（論理積） |
| 実数 1 \|\| 実数 2 | 実数 1 か実数 2 が 0.0 でない時 1（論理和） |
| ! 実数 1 | 実数 1 が 0.0 の時 1（否定） |
| 配列 1 && 配列 2 | 論理演算の結果を (0\|1) の配列で返す |
| 配列 1 \|\| 配列 2 | 論理演算の結果を (0\|1) の配列で返す |
| ! 配列 1 | 論理演算の結果を (0\|1) の配列で返す |

1. カレントディレクトリの .matxrc
   (MS-DOS バージョンでは，_matxrc)
2. ホームディレクトリ (環境変数 HOME に登録されているディレクトリ) の .matxrc(MS-DOS バージョンでは，_matxrc)
3. (a) カレントディレクトリの MaTXRC.mm
   (b) 環境変数 MATXINPUTS で定義されているディレクトリの MaTXRC.mm

   ( 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。 )

   (c) デフォルト mm ファイルディレクトリ (コンパイル時に決定される) の MaTXRC.mm

## 3.28　クイットファイル (matx のみ)

matx を quit コマンドや exit 関数を使って終了すると，以下の順序でクイット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，クイットファイルは，1 つしか読み込まれない。

1. カレントディレクトリの .matxout
   (MS-DOS バージョンでは，_matxout)
2. ホームディレクトリ (環境変数 HOME に登録されているディレクトリ) の .matxout(MS-DOS バージョンでは，_matxout)
3. (a) カレントディレクトリの MaTXOUT.mm
   (b) 環境変数 MATXINPUTS で定義されているディレクトリの MaTXOUT.mm

   ( 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。 )



RTMATX では，PC9801 や IBM PC 互換機のインターバルタイマからの割り込みにより次の 3 つの処理を行なう。[23]

1. 割り込み処理
2. 制御処理
3. 画面表示 / コマンド入力処理

プログラムは，まず，処理 3(変数の表示，キーボードからの入力待ちなど) を行なっている。タイマからの割り込みがかかると，処理 3 を中断して処理 1 (RTMATX 内部での割り込み処理) を行ない，処理 2(センサ処理，制御則の計算，アクチュエータ処理など) が終了しているかどうか調べる。終了していない場合には，タイマ割り込みを停止し，タイムアウトエラーを発生する。終了している場合には，処理 2 を行なう。処理 2 が終わると処理 3 の続きを行なう。したがって，処理 2 はサンプリング時間毎に行なわれる。

注意

インターバル・タイマによる割り込みは，タイマのカウンタに値を設定することで実現しているので，カウント・クロックの整数倍以外のサンプリング時間を用いると $\mu$sec のオーダで誤差が生じるので，注意する必要がある。後述するサンプリング時間を設定する関数 rtSetClock() は実際に行なわれる割り込みのサンプリング時間を返すので，制御器の離散化などには，この値を用いる方が良い。

注意

DJGPP を用いた現在の RTMATX では，タイムアウトエラーが発生するとシステムがハングアップすることがある。したがって，短すぎるサンプリング時間を設定してはならない。

## D.2　リアルタイム処理プログラムの作成

RTMATX には，リアルモードのリアルタイム処理プログラムを生成する Borland C 版と，プロテクトモードのリアルタイム処理プログラムを生成す

```
#ifdef __MATC__
Func void afo()
{
#endif
   Matrix A,B,C;

   A = [[1 2][3 4]];
   B = [[5 6][7 8]];
   C = A + B;
   print A,B,C;
#ifdef __MATC__
}
#endif
```

また，以下の例 (関数が afo.mm というファイルに含まれるとする) では matx をコマンドラインから

```
% matx afo.mm -e 'main();'
```

のように起動するのと同じ結果得られる。matc でコンパイルするとき，関数 afo() のみが定義され，"main();" は無視される。



```
sample.exe: sample.obj
  tlink /c /s c0h sample,sample,sample,@\rtmatx\bin\libs.rsp

sample.obj: sample.c
  bcc @\rtmatx\bin\matx.rsp sample.c

sample.c: sample.mm
  rtmatc sample.mm

sample.mm: sample.mmm
  cpp -traditional sample.mmm > sample.mm
```

## D.3　動作環境

現在，RTMATX が動作する環境は，次の通りである。今のところ，リアルタイムプログラムはインタプリタ上では動作しない。

- IBM PC 互換機の MS-DOS 上で，DJGPP 2.01 [2]を用いる。
- PC9801 シリーズの MS-DOS 上で，Borland C++ 3.1 を用いる。
- IBM PC 互換機の MS-DOS 上で，Borland C++ 3.1 を用いる。

今後，動作環境を拡張して行く予定である。

## D.4　プログラムの動作

リアルタイム処理プログラムの主な関数は，メイン関数 main()，リアルタイム処理関数 on_task()，画面表示 / コマンド入力ループ関数 off_task_loop() の 3 つである。

[2]DJGPP 2.01 は MATX の FTP サイトからフリーで入手できる

## 3.21　複数行に渡る式 (コマンドラインのみ)

演算子を行の最後に書くことによって，複数行に渡って式を書くことができる。関数呼び出しの引数並びの途中で改行するためには，コンマ “,” を行の最後に書く。ただし，不等式と複素値の定義の途中では改行することはできない。

```
a = 3 * 4 + 5 * 6 +
    7 * 8 + 9 * 10;

F = H_infty(A11, A12, A21, A22,
            B1, B2, C1, C2);
```

## 3.22　代入式

MATX における代入は，C 言語の代入式と同様に式として処理される。そして，代入式の評価順序は右から左である。ただし，リストから複数個の成分を取り出すための代入，行列の成分の代入，多項式の係数の代入，リストの成分の代入，は式ではなく文として処理される。例えば，

```
Matrix A;
Polynomial B;
List C;

A = Z(2);
A(1,1) = 1.0;           // 文
A(1,2) = [[1 2][3 4]];  // 文

B = Polynomial(Z(1,4));
B(1) = 3.0;             // 文

C = makelist(3);
C(1) = 1;               // 文
```

そして，



### D.4.4　実験装置のインターフェース関数

実験装置 (電源など) の準備をする関数 machine_ready() は

```
Func void machine_ready()
{
  void sensor_init(), actuator_init();

  sensor_init();      // センサの初期化
  actuator_init();    // アクチュエータの初期化

  pause "リターンキーを押すと、制御を開始します。";
}
```

のようにセンサの初期化とアクチュエータの初期化を行なう。

実験装置を停止する関数 machine_stop() は

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

  actuator_stop();
}
```

のようにアクチュエータを停止する。

割り込みキーが押された時に実行される関数 break_task() は

```
Func void break_task()
{
  void machine_stop();

  rtStop();
  machine_stop();
}
```

のようにリアルタイム処理を停止して，実験装置を停止する。

```
help;        // 関数一覧を表示
help eig;    // 組み込み関数 eig に関する情報を表示
help Ode;    // ユーザ関数 Ode に関する情報を表示
```

## 3.18　停止

プログラムを停止するには，pause コマンドを使用する。停止する時間を指定することができる。

```
pause;                              // 何かキー入力するまで停止
pause 5;                            // 5 秒間停止
pause "Hit Any Key\n";              // メッセージを表示して，
                                    // 何かキー入力するまで停止
pause "five seconds\n", 5;          // メッセージを表示して，
                                    // 5 秒間停止
```

## 3.19　終了

インタプリタ (matx) を終了するには quit コマンドを使用する。関数の実行中に exit() が呼ばれると，関数の実行が中断されコマンドラインに戻る。matc が生成した実行プログラムでは，exit() が呼ばれるとプログラムが終了し，引数がシェルに渡される。

```
  quit;
  exit;
  exit(-1);
```



### D.4.6　入出力カード (D/A,カウンタ) のインターフェース関数

カウンタカードや D/A 変換カードのインターフェース関数の例を次に示す。RTMATX にはパソコンの I/O ポートをアクセスする関数，16 進数表記，整数のビット操作関数が準備されているので，それらを用いれば，I/O カードのインターフェース関数を RTMATX の関数として記述できる。もちろん，C 言語で関数を記述しリンクすることも簡単にできる。

カウンタ (CONTEC, CNT24-4(PC)) のモード設定関数，初期値設定関数，初期化関数の例を次に示す。

```
// カウンタのモード設定
Func void count_mode(port)
  Integer port;
{
  Outportb(CNT_COM, port + 1);
  Outportb(CNT_DAT, MODE_CNT);
}
```

```
// カウンタの初期値設定
Func void count_set(port, data)
  Integer port;
  Integer data;
{
  Outportb(CNT_COM, port);

  Outportb(CNT_DAT, rem(data, 0x100));
  data = data / 0x100;

  Outportb(CNT_DAT, rem(data, 0x100));
  data = data / 0x100;

  Outportb(CNT_DAT, rem(data, 0x1000));
}
```

## 3.8　環境変数の設定と取得

環境変数を取得には，関数 getenv() を使用し，環境変数を設定するには，関数 putenv() を使用する。

```
putenv("DISPLAY=dazai:0.0");
disp = getenv("DISPLAY");
```

## 3.9　メニューを表示する

メニューを表示するには，関数 menu() を使用する。第 2 引数にデフォルトのメニュー番号を指定することができる。

```
mode = menu({"TITLE","item1","item2","item3","item4"});

menu0 = {"TITLE", "item1", "item2", "item3", "item4"};
default_item = 2;
mode = menu(menu0, default_item);
```

メニューのキーバインディングの次の表に示す。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，^N は，Ctrl キーを押したまま N キーを同時に押すことを意味する。

## 3.10　OS のコマンドを実行する

OS のコマンドを実行するには，関数 system() を使用する。この関数は OS からの終了コードを返す。シェルが実行できた場合は 0 が返される。インタプリタ (matx) では，コマンドラインの先頭を ! で始めると，その行はそのままシェルに渡される。



```
rtStartRehearsal()

条件 1;
on_task();

条件 2;
on_task();

条件 3;
on_task();

rtStopRehearsal()
```

のように rtStartRehearsal() と rtStopRehearsal() で囲んで行えば，on_task() の実行に必要なメモリをメモリ管理リストに登録できる。ただし，メモリ管理リストには無駄なメモリも登録されてしまう。

## D.6　リアルタイム処理用関数

RTMATX のサポートするリアルタイム処理用の関数を以下に示す。

| | |
|---|---|
| `void rtStart();` | サンプリング時間毎に rtSetTask() で設定された関数を実行する。 |
| `void rtStop();` | サンプリング時間毎の関数の実行を停止する。 |
| `void rtSetTask(on_task);`<br>`void on_task();` | サンプリング時間毎に実行される関数を設定する。 |
| `void rtSetBreak(break_task);`<br>`void break_task();` | CTRL-C などの割り込みキーが押されたときに，実行される関数を設定する。 |

とすると，倒立振子の状態方程式は次のように書ける。

$$\dot{x} = f(x,u) \stackrel{\triangle}{=} \begin{bmatrix} x_3 \\ x_4 \\ (1+\alpha \sin^2 x_2)^{-1} \times \{a_{32} \sin x_2 \cos x_2 + a_{33} x_3 \\ (1+\alpha \sin^2 x_2)^{-1} \times \{a_{42} \sin x_2 + a_{43} \cos x_2 x_3 \end{bmatrix}$$
$$\begin{bmatrix} \\ \\ \\ +a_{34} \cos x_2 x_4 + a_{35} \sin x_2 x_4^2 + b_3 u\} \\ +a_{44} x_4 + a_{45} \sin x_2 \cos x_2 x_4^2 + b_4 \cos x_2 u\} \end{bmatrix}$$

ここで，

$$\alpha = m^2 l^2 / \alpha_0$$

$$\begin{bmatrix} a_{32} & a_{33} & a_{34} & a_{35} & b_3 \\ a_{42} & a_{43} & a_{44} & a_{45} & b_4 \end{bmatrix}$$
$$= \frac{1}{\alpha_0} \begin{bmatrix} -(ml)^2 g & -F(J+ml^2) \\ (M+m)mlg & Fml \\ cml & (J+ml^2)ml & (J+ml^2)a \\ -(M+m)c & -(ml)^2 & -(ml)a \end{bmatrix}$$

ただし，$\alpha_0 = (M+m)J + Mml^2$である。

　平衡点近傍で線形近似を行なって得た線形モデルに対して，オブザーバと状態フィードバック則を設計して，シミュレーションおよびリアルタイム制御 (実験) を行なう。

表 2.3: コマンドライン編集機能のキーバインディング

| 機能 | キー 1 | キー 2 |
|---|---|---|
| 左移動 | ^B | ← |
| 右移動 | ^F | → |
| 前のコマンド | ^P | ↑ |
| 後のコマンド | ^N | ↓ |
| 行の始め | ^A | |
| 行の終り | ^E | |
| 1 文字を削除 | ^D | DEL |
| 前の 1 文字を削除 | ^H | BS |
| 履歴を表示 (行の先頭で) | ^H | |
| 行末まで削除 | ^K | |
| 削除した文字列を出力 | ^Y | |
| 画面のクリア | ^L | |
| 画面の再表示 | ^R | |

表 2.4: 履歴の設定

| 環境変数名 | 目的 | デフォルト |
|---|---|---|
| MATX_HISTSIZE | 履歴の長さ | 50 |
| MATX_HISTFILE | 履歴を保存するファイル | .matx_history |
| MATX_HISTFILESIZE | ファイルに保存する履歴の長さ | 50 |

## 2.6　mated を用いた行列の入力・編集

行列エディタ (mated) を用いて，あらかじめプログラムで使う行列を入力してファイルに保存しておくことができる。



# 付 録E　エラーメッセージ

matc を使ったコンパイルや matx による関数の解釈 (MM ファイルの読み込み)，またインタプリタによるプログラムの実行やコンパイルの結果生成された MATX のプログラムを実行するとき，いくつか種類のエラーメッセージが表示される。これらのエラーメッセージについて説明する。

- matx によるプログラムの解釈，matc によるコンパイル段階では，広範囲にわたるエラーメッセージや警告メッセージを生成し，ソースファイルのエラーの発生箇所や問題を含む部分が簡単に分かるようになっている。
- C ファイルから実行ファイルを生成する段階では，C コンパイラがエラーメッセージを表示する。
- プログラム (関数) 実行の段階に発生したエラーメッセージは，すべてランタイムエラーメッセージで，ライブラリ中の各関数がそれらを表示する。
- エラーメッセージが画面に表示されると同時に，詳しいメッセージがログファイル MaTX.log に出力される。

## E.1　プログラムの解釈，コンパイル時のエラーメッセージ

コンパイル時のエラーメッセージは，実際のプログラムのエラーを示す。matc によるコンパイル時のメッセージは，次の書式で表示される。

```
#! /usr/local/bin/matx -nocpp

Func void main()
{
  Matrix a, b, c;

  read a, b;
  c = a + b;
  prnit a, b, c;
}

main();
```

```
% emacs mmtest
% chmod a+x mmtest
% ./mmtest
```

## 2.2　matx のオプション

matx のオプションを次の表に示す。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，-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。プログラムコードのデフォルトのサイズは 10000 ステップ (MS-DOS 版は 100000 ステップ) である。

## 2.3　matc を用いた実行可能ファイルの作り方

MATX のプログラムは C 言語のプログラムと同様に複数の関数から構成される。普通は，関数には自分の好きな名前を付けてもよいが，main は特別な名前である。コンパイラ (matc) を使って実行可能型ファイルを作成したとき，プログラムの実行は関数 main() の先頭から始まる。これは，プログラムにはどこかに関数 main() がなければならないことを意味する。 ただ



**Can't open startup file**
　スタートアップファイルがみつからない。

**Confliction between classes**
　定義されていない 2 項演算が行われた。

**Confliction between left and right**
　異なる型の代入が行われた。

**Double dots is not defined**
　ダブルドット (..) が使用された。

**Element must be a real polynomial**
　複素多項式の定義に複素多項式が成分として使われた。

**Element must be a real rational**
　複素有理関数の定義に複素有理関数が成分として使われた。

**Filename must be a string**
　行列データを読み込むファイルが指定されなかった。

**Incompatible types in return**
　関数が宣言と異なる型の値を返している。

**Incorrect range**
　行列のブロック成分の指定，文字列の範囲の指定に，誤りが検出された。

**Index must be an integer**
　行列の成分の代入において整数でない指数が使われた。

**Input line started with #**
　不適当なプリプロセッサ命令が使われた。

## E.2　実行時のエラーメッセージ

実行時のエラーメッセージは，次の 3 個に分類される。

1. インタプリタ matx が検出してエラーを知らせるエラーメッセージ。
2. MATX のクラスライブラリの関数が表示するエラーメッセージ。エラーが発生すると，関数は標準エラー出力とエラーログファイル `MaTX.log` にメッセージを出力する。
3. C のランタイムライブラリの中の数値演算ルーチンが表示するエラーメッセージ。エラーが発生すると，数値演算ルーチンのいくつかは標準エラー出力にメッセージを出力する。

### E.2.1　matx の実行時のエラーメッセージ

matx によるプログラム実行時のメッセージは，次の書式で表示される。

```
"file", at or near line number, for "variable(class)", message
"file", 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表示されるとき，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。

ルが認識されると，対応するアクションが起動される。それぞれのアクションにおいて，データ型の不一致，不適当な引数の使用，初期化されていない変数の使用，等が検出される。

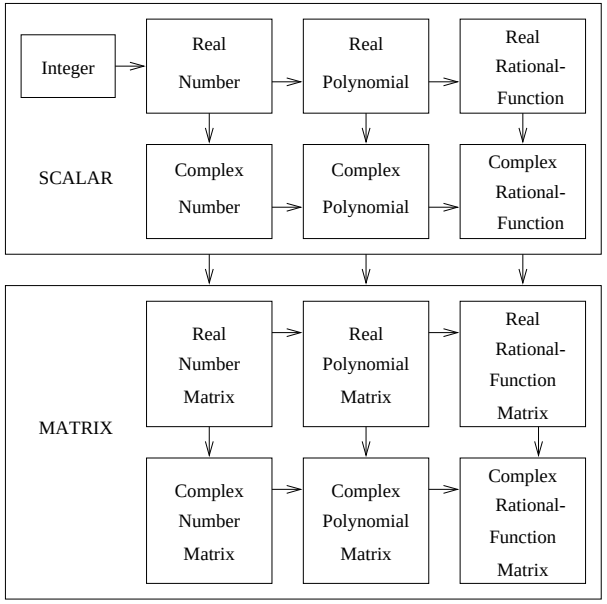



図 1.3: 自動型変換規則

型の異なるデータの演算においては，図 1.3 に示す “自動型変換規則” にしたがって，型変換が行なわれる。プログラムの多くの点ついての厳密なチェックが終ると，matc は，C 言語コード 生成部を呼びだし，matx は，内部コード 生成部を呼び出した後，生成されたコードを実行する。

## 1.3　プログラム作成手順

図 1.4 と図 1.5 は，Matlab と MATX のプログラムを作成する手順を示している。Matlab でプログラムを作成するには，ユーザは，仕様にしたがっ



# 索 引


!, 33
　OS のコマンド実行, 42
　配列の比較, 180
!=, 269
　行列の比較, 177
　配列の比較, 179
#, 267
#define, 56
#else, 56
#endif, 56
#if, 56
#ifdef, 56
#ifndef, 56
#include, 55
&&, 271
　配列の比較, 180
’, 266, 267
*, 261
　複素行列, 153
　複素数, 125
　複素多項式, 136
　複素多項式行列, 155
　複素有理多項式, 141
　複素有理多項式行列, 161
*/, 52
+, 259
++, 268
-, 28, 30, 260, 268
--, 30, 268
->, 63
-c, 32
-D, 30, 32
-e, 28, 30
-mm, 32
-Nc, 32
-nocpp, 29, 30, 32, 53
-nofep, 30
-nohist, 30
-nolog, 30
-nomm, 31, 32
-Np, 30, 32
-o, 32
-static, 32
-v, 32
.!, 180
.!=, 177

行列の大きさが大きくなればなるほど，行列の入力・編集作業は難しくなる。入力中にちょっとした間違いを犯しても，もう一度最初からやり直さなければならないのは大変な苦痛である。行列エディタ (Matrix Editor) を使えば，そういった面倒な作業から解放される。行列エディタは，ウィンドウシステムやグラフィック端末上ではもちろん，**curses** がサポートする文字端末，エスケープシーケンスをサポートする文字端末で使用可能である。零行列や単位行列等の行列を簡単に入力することができ，実行列から複素行列への変換や行列の転置を簡単に行える。

MATX で生成したデータをグラフィック表示するには，gnuplot やグラフィック描画・編集環境 xplot を使う [19]。MATX の環境から直接 gnuplot や xplot を呼び出すことができ，1 次元プロット，2 次元プロット，片対数プロットなどを簡単に表示できる。

## 1.1　プログラミング言語 MATX

制御系の設計において外乱やプラントパラメータの変化に対する閉ループ系のシミュレーションを行なうことは重要であるので，制御系設計に計算機を使用することはシミュレーションから始まった。代表的なものに非線形システムのシミュレーションを行なうために開発された Simnon[1] がある。残念ながら Simnon は行列演算を効率良く記述することができなかった。制御系の解析・設計におけるベクトルや行列演算を簡単に行ないたいという要望から行列を基本データ構造とする言語 Matlab[13] や Matrixx[17] が生まれた。Matlab や Matrixx は，Linpack や Eispack 等の信頼性のある行列計算アルゴリズムを元に構築された行列演算言語であり，紙に書いた行列演算をほとんどそのままプログラムとして使用できるという記述性の良さから近年多く使用されるようになった。

MATX は行列の演算や成分の操作法について Cleve Moler らによって開発された Matlab 言語 [12, 13] といくつかの特徴的性質を共有している。しかしながら，Matlab はデータの型を「区別しない」言語であり，唯一のデータ型は行列であり，他の型のデータの演算や操作をするには，特別な演算子



cumprod_row()
　行列の—, 185
　配列の—, 202
cumsum()
　行列の—, 184
　配列の—, 201
cumsum_col()
　行列の—, 185
　配列の—, 202
cumsum_row()
　行列の—, 185
　配列の—, 202
curses, 18
C 言語, 19, 273
C プリプロセッサ, 29

D/A 変換器, 281
De()
　有理多項式配列の—, 166
　有理多項式の—, 142
default, 115
degree(), 139
derivative()
　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
det(), 182
　多項式行列の—, 184
　複素行列の—, 184
　有理多項式行列の—, 184
diag2vec()
　行列の—, 187
　配列の—, 203
diff_eqs, 297
DISPLAY, 77, 90
DJGPP, 276
do-while, 115
double, 63

eig(), 197
eigval(), 186
eigvec(), 186, 212
EISPACK, 21, 182
else, 113
endian, 63
EPS, 117
EPS フォーマット, 88
error(), 45
eval(), 219
　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
exist(), 41
exit, 46
exp()
　行列の—, 186
　実数の—, 120
　配列の—, 205
　複素数の—, 126

6.12 グラフィック関数 3(xplot) . . . . . 93

C.1 演算子の優先度と結合規則 . . . . . 257



MatInit(), 36
matlab, 18
mat_last, 37
matp, 35
Matrix(), 132
Matrix(), 48, 135, 147
matx, 20, 27
MaTX-Lib, 20
MaTX.log, 299, 307
MaTX.mm, 65
matx.rsp, 278
MATXCPP, 53
MATXGNUPLOT, 78
MATXINPUTS, 57, 58, 61, 105, 109
MaTXOUT.mm, 58
MaTXRC.mm, 57, 61
MAT データフォーマット, 35, 62, 74, 239
max(), 182
  行列の—, 187
  実数の—, 120
  整数の—, 119
  配列の—, 201, 203
  複素数の—, 126
maximum(), 182
maxsing(), 182
mean()
  行列の—, 184, 185
  配列の—, 201
mean_col()
  行列の—, 185
  配列の—, 202
mean_row()
  行列の—, 185
  配列の—, 202
menu(), 42
mgplot(), 77
mgplot_clear(), 85
mgplot_figcode(), 88
mgplot_grid(), 86
mgplot_hold(), 88
mgplot_key(), 86
mgplot_loglog(), 81
mgplot_options(), 86
mgplot_psout(), 88
mgplot_quit(), 85
mgplot_reset(), 85
mgplot_semilogx(), 81
mgplot_semilogy(), 81
mgplot_subplot(), 84
mgplot_text(), 86
mgplot_title(), 86
mgplot_xlabel(), 86
mgplot_ylabel(), 86
mgreplot(), 80
mgreplot_loglog(), 84
mgreplot_semilogx(), 84
mgreplot_semilogy(), 84
min(), 182
  行列の—, 187

17.5.4 複雑なシステムのシミュレーション . . . . . 230

**謝辞** **233**

**参 考 文 献** **234**

**付 録 A データフォーマット** **239**
A.1 MAT データフォーマット . . . . . 239
A.2 MX データフォーマット . . . . . 240

**付 録 B メモリ管理** **249**
B.1 行列クラスのメモリ管理 . . . . . 249
B.2 メモリの状態遷移 . . . . . 251

**付 録 C 演算** **257**
C.1 和 (Addition) . . . . . 259
C.2 差 (Subtraction) . . . . . 260
C.3 積 (Multiplication) . . . . . 261
C.4 スカラ倍 (Scalar Multiplication) . . . . . 261
C.5 逆 (Inverse) . . . . . 263
C.6 商 (Division) . . . . . 263
C.7 累乗 (Power) . . . . . 265
C.8 転置 (Transpose) . . . . . 266
C.9 複素共役転置 (Complex Conjugate Transpose) . . . . . 267
C.10 符合反転 (Negation) . . . . . 268
C.11 { ポスト, プリ }{ インクリメント, デクリメント } . . . . . 268
C.12 関係演算 (Relational Operation) . . . . . 269
C.13 論理演算 (Logical Operation) . . . . . 271

**付 録 D リアルタイム処理 (RTMATX)** **273**
D.1 リアルタイム処理 . . . . . 274
D.2 リアルタイム処理プログラムの作成 . . . . . 275



行列の—, 187
配列の—, 203
shiftLeft()
行列の—, 187
配列の—, 203
shiftRight()
行列の—, 187
配列の—, 203
shiftUp()
行列の—, 187
配列の—, 203
short, 63
sin()
実数の—, 120
配列の—, 205
複素数の—, 126
singed char, 63
singed int, 63
singed short, 63
singleftvec(), 186
singrightvec(), 186
singval(), 186
sinh()
実数の—, 120
配列の—, 205
複素数の—, 126
size()
行列の—, 197
配列の—, 203
sort()
行列の—, 187
配列の—, 203
sprintf(), 132
実数の—, 122
整数の—, 119
文字列の—, 133
sqrt()
行列の—, 186
実数の—, 120
配列の—, 205
複素数の—, 126
sscanf()
文字列の—, 133
static, 99
std()
行列の—, 184, 185
配列の—, 201
std_col()
行列の—, 185
配列の—, 202
std_row()
行列の—, 185
配列の—, 202
strchr(), 130
String(), 132
String(), 47
sum()
行列の—, 184, 185
配列の—, 201
sum_col()

**第 7 章　変数** **97**
7.1　変数の宣言 . . . . . 97
7.2　通用範囲に関する規則 . . . . . 98
7.3　静的変数 . . . . . 99
7.4　暗黙の型変換 . . . . . 100

**第 8 章　関数** **103**
8.1　関数の宣言 . . . . . 103
8.1.1　require 修飾子付き関数宣言 . . . . . 104
8.2　関数の定義 . . . . . 105
8.3　複数の値を返す関数 . . . . . 106
8.4　可変長引数関数 . . . . . 107
8.4.1　リストを利用した可変長引数関数 . . . . . 107
8.4.2　一般的な可変長引数関数 . . . . . 108
8.5　関数呼び出し (matx) . . . . . 109
8.6　関数の引数渡し . . . . . 109
8.7　return 文 . . . . . 110
8.8　関数の参照 . . . . . 111
8.9　コマンドラインの引数 . . . . . 111

**第 9 章　制御の流れ** **113**
9.1　If-Else . . . . . 113
9.2　For ループ, While ループ . . . . . 114
9.3　Do-while ループ . . . . . 115
9.4　Switch 文 . . . . . 115
9.5　Break 文と Continue 文 . . . . . 116

**第 10 章 整数と実数** **117**
10.1 整数定数 . . . . . 117
10.2 整数と実数の比較 . . . . . 118
10.3 整数関数 . . . . . 119



整数, 117
整数定数, 117
静的変数, 99
成分の削除, 172
成分の代入, 170
成分の取り出し, 167
積, 261
積分
　多項式行列の—, 159
　多項式の—, 138
線形空間ベクトル, 203
線形プロット, 78, 91
宣言, 97, 103
センサ, 274, 281
全成分の積, 184, 185, 201, 202

ソート, 187

大域変数, 97
対角行列, 153, 187
対数行列, 186
対数空間ベクトル, 203
代入式, 50
タイムアウト, 275
多項式, 135
多項式行列, 154
多項式の係数, 136
多重処理, 274
単位行列, 151

通用範囲, 98
定義, 105
停止, 46
ディジタル制御, 274
データ型, 241
データの精度, 241
デクリメント, 268
デフォルト mm ファイルディレクトリ, 61, 105, 109
伝達関数, 216
転置, 266
転置行列, 187

等差数列配列, 152
同定, 273
動的フィードバック, 225
倒立振子, 293
登録関数, 40
登録変数, 40
特異値, 186
特異値分解, 198
トレース, 182

ナイキスト線図, 218
長さ, 182

ノルム, 182

配列, 147
配列演算, 219
配列演算子, 147

ピース行列, 167, 171, 175

© Copyright 1989 – 1998 Masanobu KOGA
All Rights Reserved.

No part of this publication may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical or otherwise, without prior permission of Masanobu Koga.

© 著作権　古賀 雅伸　1989 – 1998

本説明書の一部または全部を古賀雅伸の許可なく，複写，複製することは，その形態・手段を問わず禁じます。