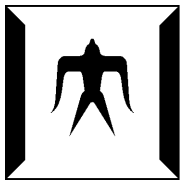
MATX (ver. 4.201)

Reference Manual



Tokyo Institute of Technology

Masanobu Koga

yacc, 14
`ylabel()`, 42

`Z()`, 98
zero matrix, 98
`zeros()`, 93

8.4 Switch 59
8.5 Break and Continue 59

9 Miscellaneous **61**
9.1 Class Conversion Functions 61
9.2 An Expression Across Lines (Command-line) 64
9.3 Assignment Expressions 64
9.4 "`ans`": answer when expression is not assigned 66
9.5 Complex-Valued Expressions 66
9.6 Comment 67
9.7 Preprocessor 67
9.7.1 File Inclusion 68
9.7.2 Macro Substitution 68
9.7.3 Conditional compilation 69
9.8 Startup File (matx) 69
9.9 Error Message 70
9.10 Bugs 70

10 Integer and Real Number **71**
10.1 Comparison of Integers and Real Numbers 72
10.2 Real Number Function 73

11 Complex Number **75**
11.1 How To Describe Complex Numbers 75
11.2 Comparison of complex numbers 76
11.3 Complex Number Function 76

12 String **79**
12.1 Element and Elements 79
12.2 Setting Elements 80
12.3 Comparison of strings 80
12.4 Evaluation of strings (matx) 80
12.5 String Functions 81

13 Polynomial **83**
13.1 How To Describe Polynomials 83
13.2 Complex Polynomials 84
13.3 Coefficients 84

help, 33
hess(), 131
higher()
  polynomial, 86
  polynomial matrix, 106
  rational function, 92
  rational matrix, 111
HOME, 69

I(), 99
identity matrix, 99
if, 57
ifft(), 134
Im(), 67
  complex matrix, 101
  complex number, 75
  polynomial, 84
  polynomial matrix, 103
  rational function, 90
  rational matrix, 108
imaginary unit, 76
increment, 181
Index, 95, 116
Inf, 72
infnorm(), 122
inline function, 55
Integer, 71
Integer(), 62
integral()
  polynomial, 86
  polynomial matrix, 106
interactive mode, 22, 23
interpreter, 14
inverse, 176
iscomplex()
  array, 132
  matrix, 121
  polynomial, 87
  rational function, 93
isnan()
  array, 132
  complex number, 77
  matrix, 121
  real number, 73
isreal(), 142
  array, 132
  matrix, 121
  polynomial, 87
  rational function, 93
isscalar(), 121

kernel(), 123

length(), 81, 121, 140
linear plot, 42
linspace(), 133
List, 137
list attachment, 139
list element, 138
list extraction, 139
ListFrees(), 25
ListInit(), 25
list_last, 25
load, 22, 28, 29
local variable, 26, 45
log(), 123
  array, 135
  complex number, 77
  real number, 73
log10(), 148
  array, 135
  complex number, 77
  real number, 73
logical, 184
logspace(), 133, 148
loop, 58

List of Tables

4.1 Key Bindings of mated 39

5.1 Graphic Functions 1 41
5.2 Graphic Functions 2 42

B.1 Precedence and Associativity of Operators 171

```
Rat-Func1 == Rat-Func2    Rat-Func1 is equal to Rat-Func2
Rat-Func1 != Rat-Func2    Func1 is not equal to Func2
```

B.13 Logical Operation

The precedence of `&&` is greater than that of `||`, and both are lower than relational and equality operators. Expressions connected by `&&` or `||` are evaluated left to right, and evaluation does not stop even if the truth or falsehood of the result is known. The unary negation operator `!` converts a non-zero or true operand into 0, and z zero or false operand into 1.

```
Integer1 && Integer2  (1 if both Int1 and Int2 are 1)
Ineger1 || Integer2   (1 if either Int1 or Int2 are 1)
!  Integer1           (1 if Int1 is 0)
Real1 && Real2        (1 if neither Real1 nor Real2 are 0.0)
Real1 || Real2        (1 if either Real1 or Real2 are not 0.0)
!  Real1              (1 if Int1 is 0)
Array1 && Array2      (Element-by-element logical operation)
Array1 || Array2      (Element-by-element logical operation)
!  Array1             (Element-by-element logical operation)
```

Chapter 1

Introduction

MATX is a high-performance programming language for scientific and engineering numerical and symbolic computation [6], [7]. Since MATX integrates numerical analysis, matrix computation, and symbolic manipulation in an easy way to use, problem solutions are expressed almost exactly as they are written mathematically. Therefor, the users can solve a lot of computational problem in much shorter time than describing programs in any other programming languages, such as FORTRAN, PASCAL, C, and BASIC. Symbolic expressions such as polynomial and rational-function, polynomial matrix, and rational-function matrix are essential to the study of system theory and signal processing. MATX allows the definition and manipulation of these objects in a natural symbolic fashion.

The bigger a matrix, the more complicated to enter a matrix. If you make a slight mistake in entering a matrix, you must start from the beginning. The matrix-editor "**mated**" provides us with a very convenient user-interface to enter and revise matrices. Scientific and engineering data, which was created in MATX , are easily examined graphically in **XPLOT** [6]. It is a useful graphical drawing environment which runs under X Window System. The flexibility and strength of MATX is easily seen to be important for designers who want to combine the interactivity and the performance of optimized FORTRAN and C libraries.

```
Integer                = Integer'
Real Number            = Real Number'
Complex Number         = Complex Number'
Real-Poly              = Real-Poly'
Comp-Poly              = Comp-Poly'
Real-Rat-Func          = Real-Rat-Func'
Comp-Rat-Func          = Comp-Rat-Func'
Real(Mat|Arr)          = Real(Mat|Arr)'
Complex(Mat|Arr)       = Complex(Mat|Arr)'
Real-Poly(Mat|Arr)     = Real-Poly(Mat|Arr)'
Comp-Poly(Mat|Arr)     = Comp-Poly(Mat|Arr)'
Real-Rat-Func(Mat|Arr) = Real-Rat-Func(Mat|Arr)'
Comp-Rat-Func(Mat|Arr) = Comp-Rat-Func(Mat|Arr)'
```

B.9 Complex Conjugate Transpose

Complex conjugate transpose of Integer, Real Number, and Real (Polynomial Rational Function) is the same as itself. Real(Matrix and Array) mean the transpose of themselves.

```
Integer                = Integer' = Integer#
Real Number            = Real Number' = Real Number#
Complex Number         = Complex Number#
Real-Poly              = Real-Poly'= Real-Poly#
Comp-Poly              = Comp-Poly#
Real-Rat-Func          = Real-Rat-Func'= Real-Rat-Func#
Comp-Rat-Func          = Comp-Rat-Func#
Real(Mat|Arr)          = Real(Mat|Arr)' = Real(Mat|Arr)#
Complex(Mat|Arr)       = Complex(Mat|Arr)#
Real-Poly(Mat|Arr)     = Real-Poly(Mat|Arr)' = Real-Poly(Mat|Arr)#
Comp-Poly(Mat|Arr)     = Comp-Poly(Mat|Arr)#
Real-Rat-Func(Mat|Arr) = Real-Rat-Func(Mat|Arr)'
                       = Real-Rat-Func(Mat|Arr)#
Comp-Rat-Func(Mat|Arr) = Comp-Rat-Func(Mat|Arr)#
```

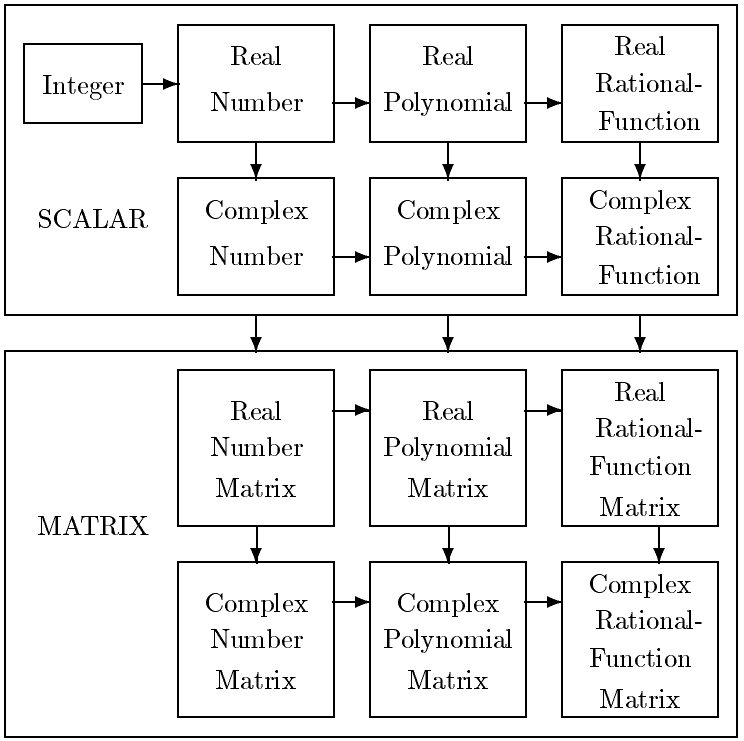



Figure 1.3: The Automatic Class Conversion Rule

MATX , the users ordinarily make several procedures (mm-files) using a favorite editor (`vi, emacs, or so`) according to the requirements (`specs`).

The interpreter (`matx`) enables us to test out the mm-files interactively file-by-file or line-by-line. The examined mm-files are compiled to portable C language files by the compiler (`matc`). It is possible to call your own C functions from MATX functions by linking C language files (`ctest.c`) to the output of matc (`test.c`). When all C language files are completed, they are compiled and linked up with the run-time library (`MaTX-Lib`) by C compiler (`cc`) to produce the desired executable program (`test`). Finally, the executable file is executed to check whether all requirements are

```
String = Integer * String
String = String * Integer
```

B.5 Inverse

```
Real Number      = Real Number    ~ = 1.0 / Real Number
Complex Number   = Complex Number ~ = 1.0 / Complex Number
Real Matrix      = Real Matrix ~
Complex Matrix   = Complex Matrix ~
Real Array       = Real Array ~
Complex Array    = Complex Array ~
Rat-FuncArray    = PolyArray ~
Rat-FuncArray    = Rat-FuncArray ~
Real-Rat-Func    = Real-Poly ~
Comp-Rat-Func    = Comp-Poly ~
Real-Rat-Func    = Real-Rat-Func ~
Comp-Rat-Func    = Comp-Rat-Func ~
String           = String ~

Real Matrix      = Real Matrix .~
Complex Matrix   = Complex Matrix .~
Rat-Func Matrix  = Poly-Matrix .~
Rat-Func Matrix  = Rat-Func Matrix .~
Real Array       = Real Array .~
Complex Array    = Complex Array .~
Rat-FuncArray    = PolyArray .~
Rat-FuncArray    = Rat-FuncArray .~
```

B.6 Division

Integer division truncates any fractional part. In the division of two matrices, "`X = A \ B`" is a solution of the equation "`A * X = B`", and "`X = A / B`" is a solution of the equation "`X * B = A`". If `A` is a square matrix, the equation is solved by using LU decomposition, and otherwise the least square solution is calculated.

Chapter 2

How To Make Programs and Execute Them

How to make programs and execute them depends on the machine and the operating system. In this manual, the programming developing process in UNIX operating system is illustrated.

2.1 How To Execute Programs

There are at least five ways to execute the program using the interpreter (matx) as follows.

1. Execute matx with source files as command line arguments. Source files are interpreted in turn. File extension "`mm`" can be omitted. It is convenient to run matx after adding a statement "`FunctionName();`" on the last line of the source file.

   [Example] Run matx after adding the statement "`main();`" on the last line of `sub3.mm`.

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

of decreasing precedence, so, for example, * and / have the same precedence which is higher than that of + and -.

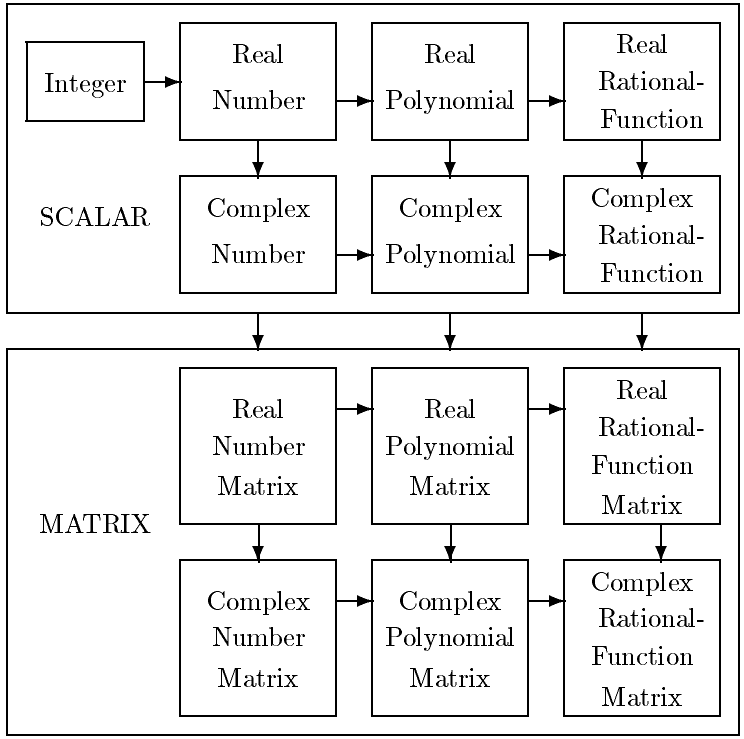


Figure B.1: The Automatic Class Conversion Rule

In case of operations between distinct class objects, "Class Conversion" is done according to the "automatic Class Conversion Rule" below. The class of the operation result, except scalar multiplication and power, is the same as that of the higher class one. Arithmetic operations of arrays are element-by-element in stead of the usual algebraic matrix operations, and the result is a new array whose elements are the result of element-by-element operation.

2.3 C Language Interface

It is very easy to call C functions from MATX functions simply by linking C functions to the output codes from `matc`. The user must obey the following interface rules when he links his own C functions to MATX programs. Please refer to a library reference manual for a detail library explanation.

1. Include some header files at the beginning of each file. At the beginning of the function "`main()`", call four class initialization functions for Matrix class, Polynomial class, Rational-Function class, and List class, that is, "`MatInit()`", "`PolyInit()`", "`RatInit()`", and "`ListInit()`". Thereafter, initialize global variables of matrices, polynomials, rational functions, and lists, respectively.

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

Matrix a, b;     /* Global Matrix variable     */
Polynomial c, d; /* Global Polynomial variable */
Rational e, f;   /* Global Rational variable   */
List g, h;       /* Global List variable       */

void main(argc, argv)
    int  argc;
    char *argv[];
{
    /* Initialization of Classes */
    MatInit(); PolyInit();
    RatInit(), ListInit();

    /* Initialization of Global Variables */
    a = MAT_DEF,  b = MAT_DEF;
    c = POLY_DEF, d = POLY_DEF;
    e = RAT_DEF,  f = RAT_DEF;
    g = LIST_DEF, h = LIST_DEF;
    ......................
}
```

	Complex Polynomial Matrix	`rows×cols` complex polynomials: `degree + 1` real numbers (4 bytes or 8 bytes).
	Real Rational Function Matrix	`rows×cols` real rational functions: `ndegree + 1` real numbers and `ddegree + 1` real numbers. (4 bytes or 8 bytes).
	Complex Rational Function Matrix	`rows×cols` complex rational functions: `2×(ndegree + 1)` real numbers and `2×(ddegree + 1)` real numbers (4 bytes or 8 bytes).
List	`length` 4-byte integer which indicates the classed of the elements, and the combination of the elements.	
Array	**Real Array**	`rows×cols` real numbers (4 bytes or 8 bytes).
	Complex Array	`2×(rows×cols)` real numbers (4 bytes or 8 bytes).
	Real Polynomial Array	`rows×cols` real polynomials: `degree + 1` real numbers (4 bytes or 8 bytes).
	Complex Polynomial Array	`rows×cols` complex polynomials: `degree + 1` real numbers (4 bytes or 8 bytes).
	Real Rational Function Array	`rows×cols` real rational functions: `ndegree + 1` real numbers and `ddegree + 1` real numbers. (4 bytes or 8 bytes).

3.4.2 Input and Output of General Data

To output all types of data to a file the **print** command is used with the "**redirect**" symbols "`->`". In case of `matx`, "`mx`" is used as the filename extension if the extension is omitted. General data is saved in "**MX Data Format**".

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

To input all types of data from a file the **read** command is used with the "**redirect**" symbols "`<-`". In case of `matx`, "`mx`" is used as the filename extension if the extension is omitted. If the all variable names are omitted, all data in the file is loaded to the variables whose name are identical with that of the variables stored.

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

If any data are saved by the method except this one, the accuracy of the data may be deteriorated.

3.4.3 Save and Load of General Data

The command `save` with variables separated by commas and a filename saves the variables in the file. These variable can be loaded from the file by the command `load` Without the filename extension, "`mm`" is used. The command `save` with a filename saves all the variables in the file, and `save`, by itself, saves everything in a file named "`MaTX.mm`".

```
save M1, r1, P1, [[1 2][3 4]], "Hello", "file.mm";
save "file"; // All variables are saved in "file.mm"
save;        // All variables are saved in "MaTX.mm"
```

The command `load` loads the variables from a file which was save by the command `save`. Since the variables are loaded as global variables, the command `load` can not be used inside of functions. Without the filename extension, "`mm`" is used.

and the MAT data format of a complex matrix

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

is

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

A.2 MX Data Format

Datafile contains more than one variable, which are stored successively. Each data starts with a 20-byte header which contains the properties of the variables. The 20-byte header consists of five 4-bytes integers.

1. Integer which indicates the type of the variable. Let represent this integer as `[M|O|P|C]`.

   "`M`" denotes the type of machine where this data was stored.

Machine	M
Sun	0
Appolo	0
Macintosh	0
PC	1
Other Intel	1
VAX D-float	2
VAX G-float	3

   "`O`" indicates the order of matrix data stored. "`O`" is 0, if column first. "`O`" is 1, if row first.

   "`P`" indicates the precision of the data.

3.13 Error

The function `error()` causes an error return to the keyboard and display the text on the screen. The message is outputed to the **stderr**. If the function is called with an argument of a 0-length string, the error is neglected, and if the function is called with no arguments, only the bell sounds and the execution is aborted.

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

3.14 Help

Executing `help` gives a list of HELP topics, including the MM-file in the various libraries. `help topic` gives help on the specified topic. If the topic is not found in the help file, the help facility looks on the disk for an MM-file with the filename `topic.mm`. If it is found, the first comments on the top of the file are displayed. This allows you to use HELP on your own MM-files.

```
help;      // General help message is displayed
help eig;  // Help of builtin function eig is displayed
help Ode;  // Help of user function of Ode is displayed
```

3.15 Pause

The command `pause` stops the execution for the specified time.

```
pause;       // Stop until any key inputs
pause(5);    // Stop for file seconds
```

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

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

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

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

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

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

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

List 8: Simulation of a complex system


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

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

Figure 4.2: Matrix Editor (Complex Matrix)

to begin many commands as well. This key is pressed, and then released before typing the next character.
How to copy a submatrix to an area ? The first step is to define the submatrix. Set the mark at the beginning of the submatrix with <ESC>-<SPACE> **set mark** and move the cursor to the end of the submatrix. Now enter "`i`" **copy submatrix to kill matrix**. The kill matrix is now identical with the specified submatrix. Then move to the beginning of the destination area and enter "`o`" **yank back from kill matrix**.

4.4 Matrix Editor (mated)

The matrix editor `mated` can be used on the command line. It deals with MAT format matrix files.

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

```
Matrix Ap, bp, cp;     /* Plant Parameters      */
Real   Ac, bc, cc, dc; /* Controller Parameters */

Func void init_params()
{
  /* Plant Parameters */
  Ap = [[0  1]
        [2  1]];
  bp = [0 1]';
  cp = [-1, 1];

  /* Controller Parameters */
  Ac = 5.0;
  bc = 1.0;
  cc = 54.0;
  dc = 9.0;
}

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} =
     OdeAuto(0.0, 20.0, x0, diff_eqs, link_eqs);

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

Chapter 5

Graphing

Scientific and engineering data are examined graphically in MATX using "graphic functions" to create plots on the screen.

Table 5.1: Graphic Functions 1

Name	Arguments	Functions
plot(Y)	Y: Array	linear x-y plot (XPLOT)
plot(X, Y)	X: Array, Y: Array	linear x-y plot (XPLOT)
splot(X, Y)	X: Array, Y: Array	semi-log x-y plot (XPLOT) (x-axis logarithmic)
gplot(Y)	Y: Array	linear x-y plot (gnuplot)
gplot(X, Y)	X: Array, Y: Array	linear x-y plot (gnuplot)
gsplot(X, Y)	X: Array, Y: Array	semi-log x-y plot (gnuplot) (x-axis logarithmic)

If one uses a graphic function (`plot()`, `splot()`), the `XPLOT` is invoked automatically and the data are plotted on the 1D-canvas, 2D-canvas, or Semilog-Canvas. Therefore, the graphic functions can be used in X-Window System.

If one uses a graphic function (`gplot()`, `gsplot()`), the `gnuplot` is invoked automatically and the data are plotted on a window.

Before running the MATX , the user should set the name of terminal, which he or she uses, to the environmental variable `DISPLAY`:

List 5: Simulation of State Variable Feedback Control (1)

Chapter 6

Variable

In the compiler (matc), all variables must be declared before use. The variables in MATX are divided into two categories, that is, local variables and global variables. Each local variable in a routine comes into existence only when the function is called, and disappears when the function is exited. Because automatic variables come and go with function invocation, they do not retain their values from one call to the next, and must be explicitly set upon each entry.

As an alternative to local variables, it is possible to define variables which are external to all functions, that is, global variables with can be accessed by name by any function that cares to. Furthermore, because external variables remain in existence permanently, rather than appearing and disappearing as functions are called and exited, they retain their values even after the functions that set them are done. All global variables must be declared before the function "`main()`". Because external variables are globally accessible, they provide an alternative to function arguments and returned values for communicating data between functions. Any function may access an external variable by referring to it by name, if the name has been declared somehow. If a large number of variables must be shared among functions, external variables are more convenient and efficient than long argument lists. However this reasoning should be applied with some caution, for it can have a bad effect on program structure, and lead to programs with many data connections between functions.

17.4 Bode and Nyquist Frequency Response

The following program involves calculation of Bode frequency response and Nyquist frequency response. The following statements are explanations of the list.

- On the 1st, 19th, and 33rd line, beginning of functions.
- On the 3rd to 5th, 23rd, 24th, 37th, and 38th line, declaration of local variables.
- On the 6th line, declaration of functions.
- On the 8th line, definition of the polynomial variable.
- On the 9th line, entering a 3-degree real rational-function.
- On the 11th line, generation of a logarithmically spaced vector.
- On the 12th and 13th line, calling the multiple-output functions `Bode()` and `Nyquist()`.
- On the 20th, 21st, 34th, and 35th line, declaration of arguments of functions.
- On the 26th and 40th line, definition of the imaginary unit.
- On the 27th and 41st line, evaluation of rational-function arrays.
- On the 28th and 29th line, array operations
- On the 30th and 42nd line, returning the list which contains two elements.

Chapter 7

Function

A MATX program, whatever its size, consists of one or more "functions" which specify the actual computing operations that are to be done. MATX function are similar to the functions of C program. Normally you are at liberty to give functions whatever names you like, but "`main`" is a special name — your program begins executing at the beginning of `main` when you make the executable program using compiler (matc). This means that every program *must* have a `main` somewhere. However the program can begin executing wherever in the interpreter (matx).

The type of the function "`main()`" is `Integer` or `void`. When the type of "`main()`" is `Integer`, it can return a value to the caller as the other functions. In this case the caller is the environment where the program is executed. The return value 0 means normal exit and the other value means error or abnormal exit.

MATX functions may be used recursively; that is, a function may call *itself* either directly or indirectly. The local variables of the function are generated as the function is called. The definition of functions can not be nested. A program may reside on one or more source files in any convenient way; the source files may be compiled separately and loaded together, along with previously compiled functions from libraries.

17.2 Discrete-Time Riccati Equations

The following list is the program which obtains the solution of the riccati equation of a discrete-time linear system by iteration. The solution `P` and a temporary matrix `PP` are initialized with identity matrix which is the same size as matrix `A`. In order to reduce the calculation error, matrix `P` is symmetrized. The calculation of the discrete-time riccati equation is iterated until the frobenius norm of the difference between matrix `P` and the previous solution matrix `PP` is less than `eps`. The following statements are explanations of the list.

- On the 1st and 16th line, beginning of functions.
- On the 3rd, 19th, and 20th line, declaration of local variables.
- On the 6th to 8th, 10th, and 23rd line, entering constant matrices.
- On the 9th, 26th, and 27th line, calculation of matrices.
- On the 12th line, calling the function `DRiccati()`.
- On the 13th line, displaying the solution matrix `P`.
- On the 17th line, declaration of arguments of the function.
- On the 26th line, examination of frobenius norm of "`P - PP`".

but not defined, a MM-FILE with the same name as the function is loaded. MM-FILES are searched as the following order: the current directory, the directory which is defined in the environmental variable `MATXINPUTS`, and the default MM-FILE directory. The environmental variable can contain multiple pathes separated by colons.

7.6 Arguments of Functions

One method of communicating data between functions is by arguments. The parentheses following the function name surround the argument list.

When an integer, a real number, a string, or a complex number appears as an argument to a function, they are passed by value, that is, the called function receives a private, temporary copy of each argument. This means that the function cannot affect the original argument in the calling function. On the other hand, when a matrix, a polynomial, a rational function, or a list appears as an argument to a function, the reference to them is passed; elements are not copied. The function can alter elements of the variable.

```
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]
}
```

16.5 List Functions

MATX provides the following list functions.

```
Integer length(List L1);               Length of the list
List    appendlist(List L1, Object A); Append a data to the list
```

Chapter 8

Control Flow

The control flow of MATX is similar to that of C language.

8.1 If-Else

The `if-else` statement is used to make decisions. Formally, the syntax is

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

where the `else` part is optional. This makes an ambiguity when an `else` part is omitted from a nested `if` sequence. Therefore the braces "{" and "}" can not be omitted. If there is an `else` part, `else` part is supposed to begin at the last line of `if` part, since `matx` interprets line by line. The `expr` is evaluated; if it is "true" ,that is, if `expr` has a non-zero value, `statement-1` is done. If it is "false" (`expr` is zero) and if there is an `else` part, `statement-2` is executed instead.

Chapter 9

Miscellaneous

9.1 Class Conversion Functions

Although the operations between distinct class objects invoke the automatic class conversions, some operations require explicit class conversions. The class conversion functions can be used for this. Their name is identical with the name of the destination class. For example, the statements

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

make matrix `C` equal to matrix `D`. The following statements uses the function "`Real()`" to converts a `1`×`1` real-matrix to a real number.

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

The function "`String()`" in the statements,

```
                      R: Upper triangular matrix
                      P: Permutation matrix
List qz(Matrix M1, Matrix M2);
                      QZ decomposition
                      {AA, BB, Q, Z, VEC} = qz(A,B)
                      Q# * A * Z = AA;
                      Q# * B * Z = BB;
                      val = diag2vec(AA) ./ diag2vec(BB);
                      A * VEC = B * VEC * vec2diag(val);
                     AA: Upper triangular matrix
                     BB: Upper triangular matrix
                      Q: Orthogonal or unitary matrix
                      Z: Orthogonal or unitary matrix
                    VEC: Generalized eigenvectors
                    val: Generalized eigenvalues
```

15.8 Array Functions

MATX provides the following array functions.

```
Integer Rows(Array A1);       Row number
Integer Cols(Array A1);       Column number
Integer isreal(Array A1);     Is real ?
                              (True = 1, False = 0)
Integer  iscomplex(Array A1); Is complex ?
                              (True = 1, False = 0)
Integer  all(Array A1);       All elements nonzero ?
                              (True = 1, False = 0)
Integer  all(Matrix M1);      All elements nonzero ?
                              (True = 1, False = 0)
Integer  any(Array A1);       Any nonzero elements ?
                              (True = 1, False = 0)
Integer  any(Matrix M1);      Any nonzero elements ?
                              (True = 1, False = 0)
Array    finite(Array A1);    ones where the elements of M1
                              are finite and zeros where
                              they are infinite or NaN.
```



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

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

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

C = makelist(3);
C(1) = 1;               /* Statement */
```

And

```
Matrix A, B, C;

A = B = C = Z(2,3);
```

are correct, but

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

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

are wrong.

```
Matrix  rkf45(Real t, Matrix x, void dx())
                : dt = 1.0E-7, u = Z(0);
Matrix  rkf45(Real t, Matrix x, void dx(), Matrix u)
                : dt = 1.0E-7
Matrix  rkf45(Real t, Matrix x, void dx(), Matrix u, Real dt)
        To solve ordinary differential equations according to
        RKF45 methods.


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);
     To solve ordinary differential equations according to
     RKF45 methods with the prespecified
     step 'h' between [t0 t1].
     {T,X,U} = Ode45();


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,
               Real dtsav, Real dtmin);
```


9.7.3 Conditional compilation

A compiler (interpreter) control line of the form

```
#if constant-expression
```

checks whether the constant expression evaluates to non-zero. A control line of the from

```
#ifdef identifier
```

checks whether the identifier is currently defined in the preprocessor, that is, whether it has been the subject of a `#define` control line. A control line of the form

```
#ifndef identifier
```

checks whether the identifier is currently undefined in the preprocessor.

All the forms above are followed by an arbitrary number of lines, possibly containing a control line

```
#else
```

and then by a control line

```
#endif
```

If the checked condition is true then any lines between `#else` and `#endif` are ignored. If the checked condition is false then any lines between the test and an `#else` or `#endif` are ignored.

9.8 Startup File (matx)

When the `matx` or `matc` is invoked, the startup file is searched and loaded, if it exits, as follows

1. "`.matxrc`" in the current directory. (For MS-DOS version, "`_matxrc`")
2. "`.matxrc`" in the home directory (The directory which is registered in the environmental variable `HOME`. (For MS-DOS version, "`_matxrc`")
3. (a) "`MaTXRC.mm`" in the current directory.

```
Matrix  vectorprod(Matrix M1, Matrix M2); Vector product
Matrix  exp(Matrix M1);        Exponential function
Matrix  sqrt(Matrix M1);       Square root function
Matrix  log(Matrix M1);        Natural logarithm function
Matrix  kernel(Matrix M1);     Kernel (Null space)
Matrix  conj(Matrix M1);       Complex conjugate
Matrix  pseudoinv(Matrix M1);  Pseudoinverse
Matrix  pseudoinv(Matrix M1, Real tol);  Pseudoinverse
                               tol: Tolerance


Matrix  vec2diag(Matrix M1);   Col-vector to diagonal matrix
Matrix  vec2diag(Matrix M1, Integer k);
                              Col-vector to diagonal matrix
                              (K > 0 is above the main diagonal)
                              (K < 0 is below the main diagonal)
Matrix  diag2vec(Matrix M1);   Diagonal matrix to col-vector
Matrix  diag2vec(Matrix M1, Integer k);
                              Diagonal matrix to col-vector
                              (K > 0 is above the main diagonal)
                              (K < 0 is below the main diagonal)
Matrix  fliplr(Matrix M1);     Flip matrices left-right
Matrix  flipud(Matrix M1);     Flip matrices up-down.
List    sort(Matrix M1);       Sorting in ascending order.
                               For complex matrices,
                               sorted by abs(M1)
      {M2, idx} = sort(M1);    M2:  Result matrix
                               idx: Indices used in the sort.
                               M2 = M1(idx)
List    sort(Array A1);        Sorting in ascending order.
                               For complex matrices,
                               sorted by abs(A1)
      {A2, idx} = sort(A1);    A2:  Result array
                               idx: Indices used in the sort.
                               A2 = A1(idx)
Matrix max(Matrix M1, Matrix M2); A matrix the same size as M1
                                  and M2 with the largest elem-
                                  ents taken from M1 or M2
```


10.2 Real Number Function

MATX provides the following real number functions. The arguments of the function must be real number. The class conversion function "`Real()`" is used in order to use any integers as the arguments.

```
Integer rem(i1, i2)       Remainder (i1 / i2)
Integer max(i1, i2)       Maximum value
Integer min(i1, i2)       Minimum value
Integer sgn(i1)           Sign function (i1 > 0, 1;
                                         i1 = 0, 0;
                                         i1 < 0, -1)
Real rem(r1, r2)      Remainder (r1 - r2 * fix(r1/r2))
Real max(r1, r2);     Maximum value
Real min(r1, r2);     Minimum value
Real sin(Real r1);    Sine function
Real sinh(Real r1);   Hyperbolic sine function
Real asin(Real r1);   Arcsine function
Real asinh(Real r1);  Arc-hyperbolic sine function
Real cos(Real r1);    Cosine function
Real cosh(Real r1);   Hyperbolic cosine function
Real acos(Real r1);   Arccosine function
Real acosh(Real r1);  Arc-hyperbolic cosine function
Real tan(Real r1);    Tangent function
Real tanh(Real r1);   Hyperbolic tangent function
Real atan(Real r1);   Arctangent function (1)
Real atanh(Real r1);  Arc-hyperbolic tangent function
Real atan2(Real y1, Real x1); Arctangent function (2)
Real abs(Real r1);    Absolute value
Real exp(Real r1);    Exponential function
Real sqrt(Real r1);   Square root function
Real log(Real r1);    Natural logarithm function
Real log10(Real r1);  Common logarithm function
Real rand();          Uniformly distributed number
Real rand("seed");    Seed of u-distributed number
Real rand("seed", Integer val);
                      Set the seed of u-dist. number
Real randn();         Normaly distributed number
```

```
0 > Matrix          maxeigval(Matrix) < 0
0 >= Matrix         maxeigval(Matrix) <= 0
0 == Matrix         Matrix is a Zero Matrix
0 != Matrix         Matrix is not a Zero Matrix
Matrix1 < Matrix2   maxeigval(Matrix1 - Matrix2) < 0
Matrix1 <= Matrix2  maxeigval(Matrix1 - Matrix2) <= 0
Matrix1 > Matrix2   mineigval(Matrix1 - Matrix2) > 0
Matrix1 >= Matrix2  mineigval(Matrix1 - Matrix2) >= 0
Matrix1 == Matrix2  Matrix1 is equal to Matrix2
Matrix1 != Matrix2  Matrix1 is not equal to Matrix2
```

```
Array1 < Array2           Element-by-element relation:
Array1 > Array2           Element-by-element relation:
                          (0|1)-Array
Array1 <= Array2          Element-by-element relation:
                          (0|1)-Array
Array1 >= Array2          Element-by-element relation:
                          (0|1)-Array
Array1 == Array2          Element-by-element relation:
                          (0|1)-Array
Array1 != Array2          Element-by-element relation:
                          (0|1)-Array
(Int|Real-Num)< Array     Element-by-element relation:
(Int|Real-Num)> Array     Element-by-element relation:
                          (0|1)-Array
(Int|Real-Num)<= Array    Element-by-element relation:
                          (0|1)-Array
(Int|Real-Num)>= Array    Element-by-element relation:
                          (0|1)-Array
(Int|Real-Num)== Array    Element-by-element relation:
                          (0|1)-Array
(Int|Real-Num)!= Array    Element-by-element relation:
                          (0|1)-Array
Array1 < (Int|Real-Num)   Element-by-element relation:
Array1 > (Int|Real-Num)   Element-by-element relation:
                          (0|1)-Array
Array1 <= (Int|Real-Num)  Element-by-element relation:
```



```
Complex acosh(Complex c1);  Arc-hyperbolic cosine function
Complex tan(Complex c1);    Tangent function
Complex tanh(Complex c1);   Hyperbolic tangent function
Complex atan(Complex c1);   Arctangent function
Complex atanh(Complex c1);  Arc-hyperbolic tangent function
Complex log(Complex c1);    Natural logarithm function
Complex log10(Complex c1);  Common logarithm function
Complex exp(Complex c1);    Exponential function
Complex conj(Complex c1);   Complex conjugate function
Complex sqrt(Complex c1);   Square root function
Complex sgn(Complex c1);    Sign function (c1 / abs(c1))
Integer finite(Complex c1); returns 1 if c1 are finite or zero,
                            and 0 if c1 are infinite or NaN
Integer isnan(Complex c1);  returns 1 if c1 are NaN,
                            and 0 if c1 are not.
```

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

15.5.5 Operation of elements by Index

Index, which consists of integers, is usefull for the operation, such as assignment and retrieving of submatrices from a matrix. An index is described by using the class conversion function "`Index()`". Double or triple integers separated by colons, which is used for setting and getting of elements from an matrix, have similar function as Index.

If `A` is a `n`-dimenstional vector and `v` is a `n`-dimensional index, `A(v)` is

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

More sophisticated effects are obtained by using index referencing on both sides of an assignment statement. For instance

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

replaces the thired, fifth, and tenth columns of `A` with the first three columns of `B` In general, if `v` and tt `w` are indices,

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

is the matrix obtained by taking the elements of `A` with row subscripts from `v` and column subscripts from `w`. So,

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

reverses the columns of `A`.

Index can be used with a piece-matrix for the operation of an array of matrices For example, if `A` is an `n` $\times$ `n` matrix, `AA` is an `n` $\times$ $\mathtt{n}^2$ array of matrix, and `v` is an `n`-dimensional index, `AA(v, A)` is


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

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

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

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

If we use the function `eval()` with the argument `"who"` or `"whos"`, we can see both local variables of the function and global variables.

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

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

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

12.5 String Functions

MATX provides the following string functions. "`printf()`" and "`fprintf()`" are general-purpose format conversion functions, which convert, formats, and output its arguments on the standard output and a specified file. The function "`sprintf()`" converts, formats, and creates strings from its arguments. Arguments may be integers, real numbers, and strings. The function "`fprintf()`" opens files in the append mode.

```
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 = higher(CRM1);   /* One shift to higher order  */
CRM3 = higher(CRM1,2); /* Two shifts to higher order */
CRM4 = lower(CRM2);    /* One shift to lower order   */
CRM5 = lower(CRM3,2);  /* Two shifts to lower order  */
```

15.4.5 Numerator Array and Denominator Array

The function "Nu()" and "De()" extract the numerator part and the denominator part from a rational function arrays, respectively.

```
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]]; */
```


real numbers can be assigned to coefficients of a real polynomial. The index must be lower than the degree of the polynomial.

```
r1 = P1(1);   /* 1st-order coefficient of P1        */
c1 = CP1(2)   /* 2nd-order coefficient of CP1       */
P1(0) = r1;   /* Assign r1 to constant term         */
CP1(3) = c1;  /* Assign c1 to 3rd-order coefficient */
```

13.4 Evaluation of Polynomials

Evaluation of polynomials at a specified value can be done by using the function "eval()".

```
s  = $;
P1 = 3*s^2 + 4*s + 5;
r1 = eval(P1, 2.0);   /* Evaluation at 2.0          */
P2 = eval(P1, s + 1); /* Evaluation at "s + 1"      */
A2 = eval(P1, [1:3]); /* Evaluation at  [1, 2, 3]   */
M2 = eval(p1, M);     /* 3*M^2 + 4*M + 5*I(M)       */
```

Evaluation of a polynomial may involve a class conversion according to the following conversion rules.

```
Real number        = eval(Real-poly, Real number)
Complex number     = eval(Real-poly, Complex number)
Complex number     = eval(Comp-poly, Real number)
Complex number     = eval(Comp-poly, Complex number)
Real polynomial    = eval(Real-poly, Real-poly)
Complex polynomial = eval(Comp-poly, Real-poly)
Complex polynomial = eval(Real-poly, Comp-poly)
Real matrix        = eval(Real-poly, Real matrix)
Complex matrix     = eval(Comp-poly, Real matrix)
Complex matrix     = eval(Real-poly, Complex matrix)
Complex matrix     = eval(Comp-poly, Complex matrix)
Real array         = eval(Real-poly, Real array)
```

```
CRM1 = (RM1,RM2); /* CRM1 = RM1 + RM2 i,          */
                  /* RM1, RM2: real-rat matrices. */
CRM1 = (RM1,*);   /* CRM1 = RM1 + Z(RM1) i,       */
                  /* RM1: real-rat matrix.        */
CRM1 = (RM1,:);   /* CRM1 = RM1 + Z(RM1) i,       */
                  /* RM1: real-rat matrix.        */
CRM1 = (*,RM1);   /* CRM1 = Z(RM1) + RM1 i,       */
                  /* RM1: real-rat matrix.        */
CRM1 = (:,RM1);   /* CRM1 = Z(RM1) + RM1 i,       */
                  /* RM1: real-rat matrix.        */
```

Real-part and imaginary part of a complex rational matrix can be extracted by using the functions "`Re()`" and "`Im()`", respectively.

```
RM1 = Re((RM1, RM2));  /* Real part matrix      */
RM2 = Im((RM1, RM2));  /* Imaginary part matrix */
```

Create a complex rational function matrix from two real rational function matrices.

```
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)]];
```

Chapter 14

Rational Function

Declaration:

```
Rational a, b, c;
```

Example:

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

MATX handles rational-functions of one variable with real and complex coefficients. Rational-functions can be added, subtracted, multiplied, and divided as usual. At the moment, a rational-function is stored as a pair of polynomials, i.e., a numerator polynomial and a denominator polynomial. Rational-functions are easily created and manipulated in MaTX.

14.1 How To Describe Rational Functions

A rational-function is described as a fraction of a pair of polynomials, and the character "`s`" is always used for the output rational-function variable.

```
s  = $;
R1 = (s + 2)/(2*s + 3);
                      /* Real rational function    */
CR1 = (1,2)/((3,4)*s + (5,6));
                      /* Complex rational function */
```

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

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

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

make a matrix B as

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

With an array,

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

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

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

make an array B as

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

The following is a class conversion rule in the evaluation of polynomial matrices.

functions. The function used without 2nd argument shift the coefficients once.

```
s  = $;
R1 = (s + 2)/(2*s + 3);
R2 = higher(R1);    /* 1 shift  for the higher order */
R2 = higher(R1,2);  /* 2 shifts for the higher order */
R3 = lower(R2);     /* 1 shift  for the lower order  */
R3 = lower(R2,2);   /* 2 shifts for the lower order  */
```

14.8 Comparison of rational functions

We can compare two rational functions by using relational operators `==`, `!=`, and the result is an integer 1 (true) or 0 (false).

```
rat1 == rat2  rat1 is equal to rat2
rat1 != rat2  rat1 is not equal to rat2
```

14.9 Rational-Function Functions

MATX provides the following rational-function functions.

```
Matrix  zeros(Rational R1);     Zero of rational-functions
Matrix  poles(Rational R1);     Pole of rational-functions
Integer isreal(Rational R1);    Classification of rat-func
                                (True = 1, False = 0)
Integer iscomplex(Rational R1); Classification of rat-func
                                (True = 1, False = 0)
```

```
M1 = ONE(3, P);     3x3 real-poly matrix,     P = Real-Poly
M1 = ONE(3, CP);    3x3 comp-poly matrix,    CP = Comp-Poly
M1 = ONE(3, R);     3x3 real-rat-func matrix,  R = Real-Rat
M1 = ONE(3, CR);    3x3 comp-rat-func matrix, CR = Comp-Rat
M1 = ONE(3, 4);     3x4 real matrix.
M1 = ONE(3, 4, c);  3x4 complex matrix,  c = Complex Number
M1 = ONE(3, 4, P);  3x4 real-poly matrix,     P = Real-Poly
M1 = ONE(3, 4, CP); 3x4 comp-poly matrix,   CP = Comp-Poly
M1 = ONE(3, 4, R);  3x4 real-rat-func matrix,  R = Real-Rar
M1 = ONE(3, 4, CR); 3x4 comp-rat-func matrix, CR = Comp-Rat
M2 = ONE(M1);       Same size and same class as matrix M1
M2 = ONE(3, M1);    Same class and three times
                    as large as matrix M1
M2 = ONE(2, 3, M1); Same class and two times by columns and
                    three times by rows as matrix M1
```

15.2.4 Arithmetic Sequence Matrices

Arithmetic sequence can be created by using the special character, the colon ":", in side square brackets, "[" and "]". The vectors created by the colon notation are not matrices but arrays.

```
/* Row array containing the numbers from a1 to a3 by a2 */
A1 = [a1:a2:a3];

/* Row array containing the numbers from a1 to a2 by 1 */
A1 = [a1:a2];

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

15.2.5 Diagonal and Block-Diagonal Matrices

In defining a diagonal or block-diagonal matrix, all elements must be the same class. Scalar values, such as integer, real number, and polynomials,


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

15.1.2 Row Vector

Row vectors are described as the explicit list of elements separated by blanks, tabs, commas ",", or carriage returns, and which is surrounded by square brackets "[" and "]".

```
M1 = [];            /* 0 x 0 matrix                      */
M1 = [1 2 3];       /* Separated by blanks               */
M1 = [1 2 , -3];    /* - operator separated by a comma */
M1 = [1 2 (-3)];    /* - operator inside parentheses   */
CM1 = [(1,2), (3,4)]; /* Complex row vector              */
```

15.1.3 Column Vector

A single quote " ' " is used to transpose a row vector.

```
M1 = [1 2 3]';       /* Separated by blanks             */
M1 = [1 2 , -3]';    /* - operator separated by a comma */
M1 = [1 2 (-3) ]';   /* - operator inside parentheses   */
CM1 = [(1,2), (3,4)]'; /* Complex column vector         */
```

15.1.4 Matrices

The way of creating matrices is with some row vectors.

15.1 How To Describe Matrices

The easiest method of entering small matrices is to use explicit lists. The explicit list of elements separated by blanks, tabs, commas ",", or carriage returns, is surrounded by square brackets "[" and "]". The element which contains the unary − operator must be separated by commas or surrounded by parentheses, "(" and ")". "`A = []`" denotes a `0 x 0` matrix. Brackets are also used for indicating the begins and ends of the rows. Arbitrary number of spaces and carriage returns may be inserted in matrices. This enables us to describe the program in a natural fashion. Big matrices can be constructed by using small matrices as elements.

15.1.1 Conversion Rule of Elements

If there are distinct class elements in a matrix, class conversion is done according to the following rules.

1. Integers are converted to real numbers.
2. If there are complex numbers or complex matrices, all elements are converted to complex numbers.
3. If there are polynomials or polynomial matrices, all elements are converted to polynomials.
4. If there are rational-functions or rational-function matrices, all elements are converted to rational-functions.

EXCEPTION If all elements are strings, they are concatenated to a string. (This exception exists only for the compatibility with MATLAB. Users should not use this expression.)

If there are some expressions in the element, those expression are evaluated at first and the class conversion follows. Please be carefull not to make the following mistake:

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

The result is totally different from the one expected, and becomes a 1×4 zero matrix. In order to obtain a desirable resut one has to describe as

should be converted to `1 x 1` matrix to define a block-diagonal matrix.

```
M1 = diag(1.0, 2.0, 3.0); /* Real diagonal matrix    */
CM1 = diag((1,2), (3,4)); /* Complex diagonal matrix */
M4 = diag(M1, M2, M3);    /* Block-diagonal matrix   */
M3 = diag(M1, M2, [1]);   /* Block-diagonal matrix   */
```

15.2.6 Complex Matrices

There are at least two convenient ways to enter complex matrices. One uses a pair of real matrices as the real-part and the imaginary-part of a matrix. The other uses complex numbers as elements of a matrix.

The real-part or imaginary-part can be omitted by using "*" or ":" instead.

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

, where `M1` and `M2` are real matrices.

Real-part and imaginary part of a complex matrix can be extracted by using the functions "`Re()`" and "`Im()`", respectively.

```
M1 = Re((M1,M2));  /* Real part matrix      */
M2 = Im((M1,M2));  /* Imaginary part matrix */
```

15.2.7 Composed Matrix

Big matrices can be constructed by using little matrices as elements. It should be noted that the size of the individual element matrices are appropriate.

```
Complex number   = eval(Comp-rat-func, Real number)
Complex number   = eval(Comp-rat-func, Complex number)
Real rat-func    = eval(Real-rat-func, Real-poly)
Real rat-func    = eval(Real-rat-func, Real-rat-func)
Complex rat-func = eval(Real-rat-func, Comp-poly)
Complex rat-func = eval(Comp-rat-func, Real-poly)
Complex rat-func = eval(Real-rat-func, Comp-rat-func)
Complex rat-func = eval(Comp-rat-func, Real-rat-func)
Complex rat-func = eval(Comp-rat-func, Comp-rat-func)
Real matrix      = eval(Real-rat-func, Real matrix)
Complex matrix   = eval(Comp-rat-func, Real matrix)
Complex matrix   = eval(Real-rat-func, Complex matrix)
Complex matrix   = eval(Comp-rat-func, Complex matrix)
Real array       = eval(Real-rat-func, Real array)
Complex array    = eval(Comp-rat-func, Real array)
Complex array    = eval(Real-rat-func, Complex array)
Complex array    = eval(Comp-rat-func, Complex array)
```

14.6 Derivative

The function "`derivative()`" calculates the derivative of rational functions. The order of derivative can be specified by the arguments. The function with only 1 argument produces 1st order derivative. Indefinite integral of rational functions is not implemented for the moment.

```
s  = $;
R1 = (s + 2)/(2*s + 3);
R2 = derivative(R1);       /* 1st order derivative */
R2 = derivative(R1,2);     /* 2nd order derivative */
```

14.7 Shift of Coefficients

The coefficients of the rational-functions can be shifted toward the lower and higher order by using the functions "`lower()`" and "`higher()`", respectively. The number of shift can be specified as the argument of the


```
Real Matrix      = eval(Real-Poly Matrix, Real Number)
Complex Matrix   = eval(Real-Poly Matrix, Complex Number)
Complex Matrix   = eval(Comp-Poly Matrix, Real Number)
Complex Matrix   = eval(Comp-Poly Matrix, Complex Number)
Real-Poly Matrix = eval(Real-Poly Matrix, Real-Poly)
Comp-Poly Matrix = eval(Comp-Poly Matrix, Real-Poly)
Comp-Poly Matrix = eval(Real-Poly Matrix, Comp-Poly)
Real-Rat Matrix  = eval(Real-Poly Matrix, Real-Rat)
Comp-Rat Matrix  = eval(Comp-Poly Matrix, Real-Rat)
Comp-Rat Matrix  = eval(Real-Poly Matrix, Comp-Rat)
Comp-Rat Matrix  = eval(Comp-Poly Matrix, Comp-Rat)
```

```
Real Matrix      = eval(Real-Poly Matrix, Real Mat)
Complex Matrix   = eval(Real-Poly Matrix, Complex Mat)
Complex Matrix   = eval(Comp-Poly Matrix, Real Mat)
Complex Matrix   = eval(Comp-Poly Matrix, Complex Mat)
Real-Poly Matrix = eval(Real-Poly Matrix, Real-Poly Mat)
Comp-Poly Matrix = eval(Comp-Poly Matrix, Real-Poly Mat)
Comp-Poly Matrix = eval(Real-Poly Matrix, Comp-Poly Mat)
Real-Rat Matrix  = eval(Real-Poly Matrix, Real-Rat Mat)
Comp-Rat Matrix  = eval(Comp-Poly Matrix, Real-Rat Mat)
Comp-Rat Matrix  = eval(Real-Poly Matrix, Comp-Rat Mat)
Comp-Rat Matrix  = eval(Comp-Poly Matrix, Comp-Rat Mat)
```

```
Real Array      = eval(Real-Poly Matrix, Real Arr)
Complex Array   = eval(Real-Poly Matrix, Complex Arr)
Complex Array   = eval(Comp-Poly Matrix, Real Arr)
Complex Array   = eval(Comp-Poly Matrix, Complex Arr)
Real-Poly Array = eval(Real-Poly Matrix, Real-Poly Arr)
Comp-Poly Array = eval(Comp-Poly Matrix, Real-Poly Arr)
Comp-Poly Array = eval(Real-Poly Matrix, Comp-Poly Arr)
Real-Rat Array  = eval(Real-Poly Matrix, Real-Rat Arr)
Comp-Rat Array  = eval(Comp-Poly Matrix, Real-Rat Arr)
Comp-Rat Array  = eval(Real-Poly Matrix, Comp-Rat Arr)
Comp-Rat Array  = eval(Comp-Poly Matrix, Comp-Rat Arr)
```

15.4.2 Evaluation of Rational Matrices

Evaluation of rational function matrices at a specified value can be done by using the function "`eval()`". If the argument is an array, it is substituted to the variable of each element of a rational function matrix. For example, substituting a matrix into the variable,

```
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);
```

make a matrix `B` as

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

$$= \begin{bmatrix} 0.66667 & 0.75000 & 0.80000 & 0.83333 \\ 0.80000 & 0.83333 & 0.85714 & 0.87500 \\ 0.85714 & 0.87500 & 0.88889 & 0.90000 \\ 0.88889 & 0.90000 & 0.90909 & 0.91667 \end{bmatrix}$$

With an array,

```
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));
```

make an array `B` as

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

```
s  = $;                    /* Polynomial variable      */
P1 = 2*s^2 + 3*s + 4;      /* 2-degree real polynomial */
P2 = Polynomial([4 3 2]); /* P2 = 2*s^2 + 3*s + 4     */
CP1 = (1,2)*s + (3,4);     /* 1-deg complex polynomial */
```

13.2 Complex Polynomials

There are two ways to enter complex polynomials. The first is to describe polynomial with complex coefficients, and the second is to use two real polynomials separated by a "," inside parentheses "(" and ")". Real or imaginary part can be omitted by specifying "*" or ":" as the other part.

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

, where `P1` and `P2` are real polynomials.

Real-part and imaginary-part of a complex polynomial can be extracted using the functions "`Re()`" and "`Im()`", respectively.

```
P1 = Re((P1, P2));  /* Real part      */
P2 = Im((P1, P2));  /* Imaginary part */
```

13.3 Coefficients

Individual polynomial coefficients can be specified with the index inside parentheses, "(" and ")". Real numbers and complex numbers can be assigned to coefficients of a complex polynomial. On the other hand, only

15.5 Manipulation of Elements

The subscripting abilities in MATX allow manipulation of rows, columns, individual elements, and submatrices of matrices. Individual matrix elements can be specified with indices separated by a comma "," inside parentheses, "(" and ")". In case of the manipulation of the first row, the row number can be omitted. Any submatrices can be extracted from a big matrix specifying the subportions with the left-top and right-bottom locations, or the intervals of rows and columns with :.

Index enables us to make submatrices form arbitrary elements of a matrix. It is also possible to assign elements of a matrix to arbitrary position of another matrix by using Index.

The piece matrix enables us to treat a big matrix as an array of a small matrix. Index and piece-matrix can be used at the same time for the operation of an arbitrary array of matrix.

15.5.1 Elements and Submatrices

`M1(1,1)`	(1,1)-element
`M1(5)`	The 5th elements in the order: (1,1) → (1,Cols(M1)) → (2,1) → (2,Cols(M1)) → ⋯ → (Rows(M1),Cols(M1))
`M1(3:5)`	The vector of 3-5th elements
`M1(1,*)`	The first row vector
`M1(1,:)`	The first row vector
`M1(*,1)`	The first column vector
`M1(:,1)`	The first column vector
`M1(0,M2)`	The first submatrix according to the piece matrix M2
`M1(1,M2)`	The second submatrix according to the piece matrix M2
`M1(0,1,M2)`	The (1,2)-submatrix according to the piece matrix M2
`M1(1,2,M2)`	The (2,3)-submatrix according to the piece matrix M2

12.2 Setting Elements

`S(1) = "A"`	Set "A" to 1st element of S
`S(2:4) = "AB"`	Set "AB" between 2nd and 4th element of S

12.3 Comparison of strings

We can compare two strings by using relational operators `==, !=`, and the result is an integer 1 (true) or 0 (false).

```
string1 == string2   string1 is equal to string2
string1 != string2   string1 is not equal to string2
```

12.4 Evaluation of strings (matx)

Evaluation of a string which consists of any MATX codes can be done by using the function `eval()`. The return value is consistent with the evaluation result, i.e, the class of the return value depends on the string which should be evaluated. If the statement of MATX is evaluated instead of an expression, an integer 0 is always returned. For example, in the following statements

```
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");
```

the expression `eval("a + b")` calculates sum of real numbers `a` and `b`, whereas the expression `eval("A + B")` calculates the sum of matrices `A` and `B`.

It is very usefull for debuging the program to use the function `eval()` in a function as follows.


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

More sophisticated effects are obtained by using index referencing on both sides of an assignment statement. For instance

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

replaces the thired, fifth, and tenth columns of `AA` with the first three columns of `BB` In general, if `v` and tt w are indices,

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

is the sub-matrix obtained by taking the block-elements of `AA` with row subscripts from `v` and column subscripts from `w`. So,

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

reverses the columns of `AA`.

`M1(IDX2)`	Submatrix with IDX2-elements of M1
`M1(1,IDX2)`	Submatrix with 1st-row, IDX2-col of M1
`M1(IDX1,IDX2)`	Submatrix with IDX1-row, IDX2-col of M1
`M1(IDX1,1)`	Submatrix with IDX1-row, 1st-col of M1
`M1(IDX1,1:2)`	Submatrix with IDX1-row, 1st-2nd-col of M1
`M1(1:2,IDX2)`	Submatrix with 1st-2nd-row, IDX2-col of M1
`MM(IDX2,M)`	Submatrix 1st-row, IDX2-col for a piece matrix M of MM
`MM(IDX1,IDX2,M)`	Submatrix IDX1-row, IDX2-col for a piece matrix M of MM
`MM(1,IDX2,M)`	Submatrix 1-row, IDX2-col for a piece matrix M of MM
`MM(IDX1,1,M)`	Submatrix IDX1-row, 1-col for a piece

Real-part and imaginary part of a complex number can be extracted by using the functions "Re()" and "Im()", respectively.

```
r1 = Re((r1, r2));  /* Real part      */
r2 = Im((r1, r2));  /* Imaginary part */
```

Imaginary unit can be defined at least by the following methods.

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

11.2 Comparison of complex numbers

We can compare two complex numbers by using relational operators ==, !=, and the result is an integer 1 (true) or 0 (false).

```
comp1 == comp2   comp1 is equal to comp2
comp1 != comp2   comp1 is not equal to comp2
```

11.3 Complex Number Function

MaTX provides the following Complex-number functions.

```
Real    abs(Complex c1);     Absolute value
Real    arg(Complex c1);     Argument
Complex max(c1, c2);         Maximum value
Complex min(c1, c2);         Minimum value
Complex sin(Complex c1);     Sine function
Complex sinh(Complex c1);    Hyperbolic sine function
Complex asin(Complex c1);    Arcsine function
Complex asinh(Complex c1);   Arc-hyperbolic sine function
Complex cos(Complex c1);     Cosine function
Complex cosh(Complex c1);    Hyperbolic cosine function
Complex acos(Complex c1);    Arccosine function
```



```
                              (0|1)-Array
Array1 >= (Int|Real-Num)      Element-by-element relation:
                              (0|1)-Array
Array1 == (Int|Real-Num)      Element-by-element relation:
                              (0|1)-Array
Array1 != (Int|Real-Num)      Element-by-element relation:
                              (0|1)-Array


Array1 && Array2   Element-by-element logical operation:
                   (0|1)-Array
Array1 || Array2   Element-by-element logical operation:
                   (0|1)-Array
!  Array1          Element-by-element logical operation:
                   (0|1)-Array
```

15.7 Matrix Functions

MATX provides the following matrix functions. The algorithms in EISPACK [12], [11], [2] is used for eigenvalue problems. . The algorithms in [10] is used for solving ordinary differential equations according to 4th order Runge-Kutta methods automatically changing the step to guarantee the prespecified error. The reference [1] is used for solving ordinary differential equations according to RKF45 methods with the prespecified step. LU decomposition, QR decomposition, and QZ decomposition are implemented by refering [3].

```
Integer  rank(Matrix M1);           Rank
Integer  rank(Matrix M1, Real tol); Rank
                                    tol: Tolerance
Integer  Rows(Matrix M1);           Row number
Integer  Cols(Matrix M1);           Column number
Integer  length(Matrix M1);         Length of row
                                    or column vector
Integer  isreal(Matrix M1);         Is real matrix ?
                                    (True = 1, False = 0)
Integer  iscomplex(Matrix M1);      Is complex matrix ?
```

```
PI    =  3.14159265358979323846; /* Circumf./Diameter */
E     =  2.71828182845904523536; /* Napier's Number   */
GAMMA =  0.57721566490153286060; /* Euler's Constant  */
DEG   = 57.29577951308232087680; /* Degree/Radian     */
PHI   =  1.61803398874989484820; /* Golden Ratio      */
PID   =  ??????;                 /* Process ID        */
EPS   =  pow(2.0, -54.0);        /* Machine Epsilon   */
Inf   =  Infinity;               /* Infinity          */
                                 /* Ex. 1.0/0.0       */
                                 /* positive infinity */
NaN   =  NaN;                    /* Not a Number      */
                                 /* Ex. 0.0/0.0       */
```

10.1 Comparison of Integers and Real Numbers

We can compare two integers (and real numbers) by using relational operators `>`, `>=`, `<`, `<=`, `==`, `!=`, and the result is an integer 1 (true) or 0 (false).

```
int1 > int2     int1 is greater than int2
int1 >= int2    int1 is greater than or equal to int2
int1 < int2     int1 is less than int2
int1 <= int2    int1 is less than or equal to int2
int1 == int2    int1 is equal to int2
int1 != int2    int1 is not equal to int2
re-n1 > re-n2   re-n1 is greater than re-n2
re-n1 >= re-n2  re-n1 is greater than or equal to re-n2
re-n1 < re-n2   re-n1 is less than re-n2
re-n1 <= re-n2  re-n1 is less than or equal to re-n2
re-n1 == re-n2  re-n1 is equal to re-n2
re-n1 != re-n2  re-n1 is not equal to re-n2
```



```
Matrix min(Matrix M1, Matrix M2); A matrix the same size as M1
                                  and M2 with the smallest elem-
                                  ents taken from M1 or M2
Array rem(Array A1, Real r1);  rem(,r1) for echo element of A1
Array rem(Array A1, Array A2); rem() for echo element of A1
                               and element of A2


Matrix  rand(Integer m);   m x m Uniformly distributed matrix
Matrix  rand(Integer m, Integer n);
                           m x n Uniformly distributed matrix
Matrix  rand(Matrix A);    Uniformly distributed matrix
                           as large as matrix A
Matrix  randn(Integer m);  m x m Normaly distributed matrix
Matrix  randn(Integer m, Integer n);
                           m x n Normaly distributed matrix
Matrix  randn(Matrix A);   Normaly distributed matrix
                           as large as matrix A


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)
        To solve ordinary differential equations according
        to 4th order Runge-Kutta methods.


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);
     To solve ordinary differential equations according
```

9.7.1 File Inclusion

In order to facilitate handling collections of `#define`'s and declarations MATX provides a file inclusion feature. Any line that looks line

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

is replaced by the contents of the file `filename.mm`. `#include` is the preferred way to tie the declarations together for a large program. The named file is searched for first in the directory of the original source file, and then in a sequence of standard places (/usr/local/MaTX/inputs). Alternatively, a control line of the form

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

searches only the standard places, and not the directory of the source file.

9.7.2 Macro Substitution

A definition of the from

```
#define     YES     1
```

calls for a macro substitution of the simplest kind — replacing a name by a string of characters. Normally the replacement text is the rest of the line; a long definition may be continued by placing a \ at the end of the line to be continued. The "scope" of a name defined with `#define` is from its point of definition to the end of the source file.

It is also possible to define macros with arguments, so the replacement text depends on the way the macro is called. For example, define a macro called `ABCD` like this:

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

Now the line

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

will be replace by the line

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

This provides a "ABCD function" that expands into in-line code rather than a function call.


```
                : 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);
    To solve ordinary differential equations according to
    RKF45 methods automatically changing
    the step to guarantee the prespecified error between
    [t0 t1], where the minimum and maximum step are
    dtmin and dtmax, respectively.
    {T,X,U} = Ode45Auto();


List Ode45Hybrid(Real t0, Real t1, Real dt, Matrix x,
        void dx())
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)
    To solve ordinary differential equations according to
    RKF45 methods with the prespecified
    step 'h' between [t0 t1], where the external input
    is constant for the prespecified sampling interval.
    If the step 'h' is not prespecified and 4.0E-2 < h,
    h is equal to dt.
    {T,X,U} = Ode45Hybrid();


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

```
CM1 = CoArray(Real Array|Matrix)       Re(CM1) = M, Im(CM1) = 0
PM1 = PoArray(Real Array|Matrix)       Const. term array is M
CPM1 = PoArray(Compex Array|Matrix)    Const. term array is M
RM1 = RaArray(Real Array|Matrix)       Const. term array of
                                       numerator array is M
CRM1 = RaArray(Complex Array|Matrix)   Const. term array of
                                       numerator array is M
RM1 = RaArray(Polynomial Array|Matrix) Numerator array is M
```

9.2 An Expression Across Lines (Command-line)

An operator or a comma "," at the end of a line means that the expression continues to the next line. However, a new line can not be started among inequalities and complex-value expressions.

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

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

9.3 Assignment Expressions

MATX assignment has a value and assignment associate right to left. However, the assignment for extracting some elements from a list, the assignment for setting elements of matrices, the assignment for setting coefficients of polynomials, and the assignment for setting elements of lists, are not expressions but statements. For example,


```
Array    isnan(Array A1);     ones where the elements are
                              NaN zeros where they are not.

Index find(Array A1);         Index of nonzero elements
                              If A1 is not a vector, the
                              elements are checked row-wise.
Index find(Matrix M1);        Index of nonzero elements
                              If M1 is not a vector, the
                              elements are checked row-wise.

Real max(Array A1);               Maximum element
Real min(Array A1);               Minimum element
Real frobnorm(Array A1);          Frobenius norm
Real sum(Array A1);               Sum of all elements
Real mean(Array A1);              Mean of all elements

Complex sum(CArray CA1);          Sum of all elements
Complex mean(CArray CA1);         Mean of all elements

Polynomial frobnorm(PArray PA1); Frobenius norm
Polynomial sum(PArray PA1);       Sum of all elements
Polynomial mean(PArray PA1);      Mean of all elements

Rational frobnorm(RArray RA1);    Frobenius norm
Rational sum(RArray RA1);         Sum of all elements
Rational mean(RArray RA1);        Mean of all elements

Array logspace(Real d1, Real d2)
                Logarithmically spaced vector (split=50)
Array logspace(Real d1, Real d2, Integer split)
                Logarithmically spaced vector
Array linspace(Real d1, Real d2)
                Linearly spaced vector (split=100)
Array linspace(Real d1, Real d2, Integer split)
                Linearly spaced vector
```

```
break;
```

The `break` statement provides an early exit from `for` and `while`. A `break` statement causes the innermost enclosing loop to be exited immediately.

```
continue;
```

The continue statement is related to `break`, but less often used; it causes the *next iteration* of the enclosing loop (`for` and `while`) to begin. In the `while` loop, this means that the re-initialization step.

Chapter 16

List

Declaration:

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

Example:

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

MATX has a list data type. The list is a collection of data objects not necessarily of the same type. A list can contain any of the MATX data types as well as other lists. List enables us to define multiple-output functions.

16.1 How To Describe Lists

The method of describing a list class object is to use an explicit list of objects which are separated by the commas "`,`" and surrounded by a couple of braces, "{" and "}". For example, entering the statement

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

results in a list which contains an integer, a complex-number, and a real matrix. List elements can be any MATX expressions. For example, the statement

Chapter 17

Examples

There are several illustrative example programs in this chapter, such as a program to solve continuous-time riccati equations, a program to solve discrete-time riccati equations, a program to obtain the transfer functions and transfer function matrices of linear systems, a program to calculate the bode and nyquist frequency responses, and three simulation programs of the state variable feedback control and the dynamic feedback control.

7.4 Variable Number of Arguments

In order to implement functions with variable number of arguments, we avail ourselves "List Class". As an example of a function with variable number of arguments, we observe the following function which calculates the sum of the argument matrices and returns it. Here, we assume that all matrices have the same size.

```
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;
}
```

In order to call the function with variable number of arguments, we make a list from the arguments and send it to the function. The function "`sum_mat()`" is called like

```
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);
```

7.5 Calling Functions (matx)

A function is call by its name and the parentheses following the function name surround the argument list. The parentheses can not be omitted even if there is no arguments. When the function which has been declared


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

     P = DRiccati(A, B, Q, R); /* Call DRiccati(). */
     print P;                  /* Print P.         */
   }
15:
   Func Matrix DRiccati(A, B, Q, R)
     Matrix A, B, Q, R;
   {
     Matrix P, PP;
20:  Real eps;

     P = PP = I(A); /* Initialization of the solution. */
     eps = 1.0E-7;

25:  /* Repeat the loop until the solution converges. */
     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;   /* Symmetrise matrix P. */
30:  }

     return P;  /* Return the solution. */
   }
```

List 2: The Program of Discrete-Time Riccati Equation

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

     s = $;                          /* Polynomial variable */
     G = (s^2 + 3*s + 4)/(s^3 + 5*s^2 + 6*s + 7);
10:                                  /* Transfer function   */
     w = logspace(0.01, 100.0);      /* Range of frequency  */

     {ga, ph} = Bode(G, w);          /* Gain and phase      */
     {re, im} = Nyquist(G, w);       /* Real and Imag. part */
15:
     print [[w][re][im]] >> "nyquist.mat"; /* Save data */
     print [[w][ga][ph]] >> "bode.mat";    /* Save data */
   }

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

     j = (0,1);                      /* Imaginary Unit   */
     data = eval(G, j*w);            /* G(j*w)           */
     ga = 20.0 * log10(abs(data)); /* Gain in decibels */
30:  ph = arg(data) * 180.0 / PI;  /* Phase in degrees */
     return {ga, ph};                /* Gain and Phase   */
   }

   Func List Nyquist(G, w)
35:  Rational G;
     Array w;
   {
     Complex j;
     Array data;
40:
     j = (0,1);                      /* Imaginary unit   */
     data = eval(G, j*w);            /* G(j*w)           */
     return {Re(data), Im(data)};  /* Re- and im- part */
   }
```

List 4: Bode Nyquist Frequency Response

the benefit of allowing vectors of different lengths to be displayed on the same graph. As before, each pair uses a different line type.

5.3 Logarithmic Plots

The use of the function `splot()` (`gsplot()`) is identical to the use of `plot()` (`gplot()`). The function `splot()` (`gsplot()`) is a plot using semi-log scales. The x-axis is $\log_{10}$ while the y-axis is linear.

17.5.2 State Variable Feedback Control (2)

The following program does the hybrid-time simulation of the linear system above by using 4th order Runge-kutta method directory.

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

  t1 = 0.0;  /* Initial time  */
  t2 = 5.0;  /* Terminal time */
  dt = 0.01; /* Time Interval */
  ii = Integer((t2 - t1) / dt);

  x = [1 1 1]';  /* Initial state */
  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 void diff_eqs(DX, t, X, UY)
  Real t;
  Matrix X, DX, UY;
{
  Matrix xp, dxp;
  Real xc, dxc, up, uc;

  xp = X(1:2,1);          /* State of plant      */
  up = UY(1,1);           /* Input to plant      */
  xc = X(3,1);            /* State of controller */
  uc = UY(2,1);           /* Input to controller */

  dxp = Ap * xp + bp * up; /* System eq. of plant      */
  dxc = Ac * xc + bc * uc; /* System eq. of controller */

  DX = [[dxp][dxc]];
}

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

  xp = X(1:2,1);          /* State of plant         */
  xc = X(3,1);            /* State of controller    */

  yp = Real(cp * xp);     /* Output from plant      */
  uc = yp;                /* Link 'yp' to 'uc'      */

  yc = cc * xc + dc * uc; /* Output from controller */
  up = - yc;              /* Link 'yc' to 'up'      */

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

List 7: Simulation of Dynamic Feedback Control

4.1 Edition of Real Matrices

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

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

Figure 4.1: Matrix Editor (Real Matrix) — mated

4.2 Edition of Complex Matrices

The command `C` transforms real matrices to complex matrices and gives back the complex matrix from real-part or imaginary-part matrix. (matx only) In editing the real-part or imaginary-part matrix, the size of the matrix can not be changed, the matrix can not be written out to a file and read from a file. Figure 4.2 shows the matrix editor which deals with a complex matrix, and Figure 4.3 and Figure 4.4 show the matrix editor which deal with the real-part matrix and imaginary-part matrix, respectively.

4.3 Key Bindings

The control key can be used before any alphabetic character and some symbols. For example, "^C" means to hold down the <CONTROL> key and type the "`C`" key at the same time. The <ESC> is a special key used

Bibliography

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

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

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

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

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

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

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

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

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

3.9 Command of OS

The function "`system()`" send the arguments to operating system as a command.

```
system("xplot -silent A.mat");
```

3.10 Clear Screen (matx only)

The command `clear` clears the screen.

```
clear;
```

3.11 bell

The function `bell()` rings a bell.

```
bell();
```

3.12 Warning

The function `warning()` displays the text on the screen. The message is outputed to the **stderr**. If the function is called with an argument of a 0-length string, the warning is neglected.

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


Data Precision	P
Double-precision real number	0
Single-precision real number	1
Signed 32-bit integer	2
Signed 16-bit integer	3
Unsigned 16-bit integer	4

"C" indicates the class of the data.

Data Class	C
Matrix	0
Integer	1
Real Number	2
Complex Number	3
String	4
Polynomial	5
Rational	6
List	7
Array	8
Index	9

2. The second integer depends on the class of data.

Data Class	Name	Content
Matrix	rows	Row number
Integer	dummy	
Real Number	dummy	
Complex Number	dummy	
String	stringlen	Length of string
Polynomial	degree	Degree
Rational	ndegree	Degree of Numerator
List	length	Length of element
Array	rows	Row number
Index	rows	Row number

3. The third integer depends on the class of data.

3.3 Load MM-FILES (matx only)

The command `load` with a **MM-FILE** filename loads the file. Multiple files separated by commas can be specified at a time. Without the filename extension, "`mm`" is used. MM-FILES are searched through the following order:

1. The current directory
2. The directory which is defined by the environment variable `MATXINPUTS`
   (Multiple directories separated with colons ":" can be specified. For MS-DOS version, multiple directories are separated with semicolons ";")
3. The default MM-FILE directory, which is determined at the compile time.

The command `load`, by itself, loads the variables from the file "`MaTX.mm`".

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

3.4 Input and Output of Data

MATX provides several ways to save and load data into and from files.

3.4.1 Input and Output of Matrix Data

To input matrix data from files and output matrix data to files, the **read** and **print** commands are used with the "**redirect**" symbols "<<" and ">>" like the UNIX shell. "`mat`" is recommended as the filename extension for matrix data files. Matrix data are saved in "**MAT Data Format**".

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


Complex Rational Function Array	`rows×cols` complex rational functions: `2×(ndegree + 1)` real numbers and `2×(ddegree + 1)` real numbers (4 bytes or 8 bytes).
Index	`cols` real numbers (4 bytes or 8 bytes).

3. Compile it with the command "`matcc test.c`", and make an executable file called `test`.

4. Running that by the command `test`. If any errors are found in the execution, error messages are displayed on the screen and the execution stops. More detail error information is written out to the file "`MaTX.log`".

[Example] Making an executable file `main` from four source files.

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


B.1 Addition

```
Integer                 = Integer + Integer
Real Number             = Real Number + Real Number
Complex Number          = Complex Number + Complex Number
Real-Poly               = Real-Poly + Real-Poly
Comp-Poly               = Comp-Poly + Comp-Poly
Real-Rat-Func           = Real-Rat-Func + Real-Rat-Func
Comp-Rat-Func           = Comp-Rat-Func + Comp-Rat-Func
Real(Mat|Arr)           = Real(Mat|Arr) + Real(Mat|Arr)
Complex(Mat|Arr)        = Complex(Mat|Arr) + Complex(Mat|Arr)
Real-Poly(Mat|Arr)      = Real-Poly(Mat|Arr) + Real-Poly(Mat|Arr)
Comp-Poly(Mat|Arr)      = Comp-Poly(Mat|Arr) + Comp-Poly(Mat|Arr)
Real-Rat-Func(Mat|Arr)  = Real-Rat-Func(Mat|Arr)
                                    + Real-Rat-Func(Mat|Arr)
Comp-Rat-Func(Mat|Arr)  = Comp-Rat-Func(Mat|Arr)
                                    + Comp-Rat-Func(Mat|Arr)
String                  = String + String
```

B.2 Subtraction

```
Integer                 = Integer - Integer
Real Number             = Real Number - Real Number
Complex Number          = Complex Number - Complex Number
Real-Poly               = Real-Poly - Real-Poly
Comp-Poly               = Comp-Poly - Comp-Poly
Real-Rat-Func           = Real-Rat-Func - Real-Rat-Func
Comp-Rat-Func           = Comp-Rat-Func - Comp-Rat-Func
Real(Mat|Arr)           = Real(Mat|Arr) - Real(Mat|Arr)
Complex(Mat|Arr)        = Complex(Mat|Arr) - Complex(Mat|Arr)
Real-Poly(Mat|Arr)      = Real-Poly(Mat|Arr) - Real-Poly(Mat|Arr)
Comp-Poly(Mat|Arr)      = Comp-Poly(Mat|Arr) - Comp-Poly(Mat|Arr)
Real-Rat-Func(Mat|Arr)  = Real-Rat-Func(Mat|Arr)
                                    - Real-Rat-Func(Mat|Arr)
Comp-Rat-Func(Mat|Arr)  = Comp-Rat-Func(Mat|Arr)
                                    - Comp-Rat-Func(Mat|Arr)
```

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

Integer        = Integer / Integer
Real Number    = Real Number / Real Number
Complex Number = Complex Number / Complex Number

Real-Poly      = Real-Poly / (Integer|Real Number)
Comp-Poly      = Real-Poly / Complex Number
Comp-Poly      = Comp-Poly / (Integer|Real Number)
Comp-Poly      = Comp-Poly / Complex Number

Real-Rat-Func  = Real-Poly / Real-Poly
Comp-Rat-Func  = Comp-Poly / Comp-Poly
Real-Rat-Func  = Real-Rat-Func / Real-Rat-Func
Comp-Rat-Func  = Comp-Rat-Func / Comp-Rat-Func

Real(Mat|Arr)          = Real(Mat|Arr) / Real(Mat|Arr)
Complex(Mat|Arr)       = Complex(Mat|Arr) / Complex(Mat|Arr)
Real-Poly(Mat|Arr)     = Real-Poly(Mat|Arr) / (Integer|Real Number)
Comp-Poly(Mat|Arr)     = Real-Poly(Mat|Arr) / Complex Number
Comp-Poly(Mat|Arr)     = Comp-Poly(Mat|Arr) / (Integer|Real Number)
Comp-Poly(Mat|Arr)     = Comp-Poly(Mat|Arr) / Complex Number
Real-Rat-Func(Mat|Arr) = Real-Rat-Func(Mat|Arr) / (Int|Real Number)
Comp-Rat-Func(Mat|Arr) = Real-Rat-Func(Mat|Arr) / Complex Number
Comp-Rat-Func(Mat|Arr) = Comp-Rat-Func(Mat|Arr) / (Int|Real Number)
Comp-Rat-Func(Mat|Arr) = Comp-Rat-Func(Mat|Arr) / Complex Number

Real Matrix       = Real Matrix ./ Real Matrix
Complex Matrix    = Complex Matrix ./ Complex Matrix
Rat-Func Matrix   = (Real|Comp|Poly|Rat-Func)Matrix ./ Poly-Matrix

Integer           = Integer \ Integer
Real Number       = Real Number \ Real Number
Complex Number    = Complex Number \ Complex Number

Real-Rat-Func     = Real-Poly \ (Integer|Real Number)
Comp-Rat-Func     = Real-Poly \ Complex Number
Comp-Rat-Func     = Comp-Poly \ (Integer|Real Number)
Comp-Rat-Func     = Comp-Poly \ Complex Number
```

potential portability difficulties, and the like. In case of operations be-

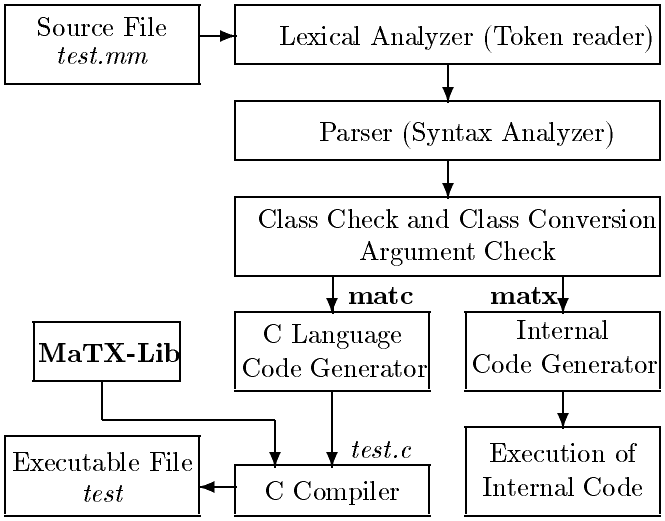


Figure 1.2: The Internal Process of MATX

tween distinct class objects, "Class Conversion" is done according to the "Automatic Class Conversion Rule" shown in Fig. 1.3. After the strict check to many aspects of a program, **matc** calls the "C Language Code Generator", while **matx** calls the "Internal Code Generator" and puts the generated code into execution.

1.3 Program Developing Process

Figure 1.4 and Fig. 1.5 show the program developing process in Matlab and MATX , respectively. In the program developing process in Matlab, the users usually realize algorithms in m-files (`test.m`) according to the requirements (`specs`). The m-files are tested several times until they satisfy the requirements. It is possible to call the user's C or Fortran subroutines (`ctest.mtex`), from Matlab which are produced from source code (`ctest.c`) by C or Fortran compiler. In the program developing process in

B.10 Negation

```
Integer                 = - Integer
Real Number             = - Real Number
Complex Number          = - Complex Number
Real-Poly               = - Real-Poly
Comp-Poly               = - Comp-Poly
Real-Rat-Func           = - Real-Rat-Func
Comp-Rat-Func           = - Comp-Rat-Func
Real(Mat|Arr)           = - Real(Mat|Arr)
Complex(Mat|Arr)        = - Complex(Mat|Arr)
Real-Poly(Mat|Arr)      = - Real-Poly(Mat|Arr)
Comp-Poly(Mat|Arr)      = - Comp-Poly(Mat|Arr)
Real-Rat-Func(Mat|Arr)  = - Real-Rat-Func(Mat|Arr)
Comp-Rat-Func(Mat|Arr)  = - Comp-Rat-Func(Mat|Arr)
```

B.11 Increment and Decrement Operators

MATX provides two operators for incrementing and decrementing variables. The increment operator "`++`" adds 1 to its operand; the decrement operator "`--`" subtracts 1. The interesting aspect is that "`++`" and "`--`" may be used either as prefix operators (before the variable, as in "`++n`)", or postfix (after the variable "`n++`)". In both cases, the effect is to increment `n`.

```
Integer++         /* Integer = Integer + 1           */
Real Number++     /* Real Number = Real Number + 1.0 */
Integer--         /* Integer = Integer - 1           */
Real Number--     /* Real Number = Real Number - 1.0 */
++Integer         /* Integer = Integer + 1           */
++Real Number     /* Real Number = Real Number + 1.0 */
--Integer         /* Integer = Integer - 1           */
--Real Number     /* Real Number = Real Number - 1.0 */
```

B.12 Relational Operation

The relational operators are `>, >=, <, <=`. They all have the same precedence. Just below them in precedence are the equality operators `==, !=`.

Index

-, 22
-c, 23
-nocpp, 23, 67
-nomm, 23
.matxrc, 69
:, 113
_matxrc, 69
*
  complex matrix, 101
  complex number, 75
  polynomial, 84
  polynomial matrix, 102
  rational function, 90
  rational matrix, 107
->, 29
1×1 matrix, 61
:, 100
  complex matrix, 101
  complex number, 75
  polynomial, 84
  polynomial matrix, 102
  rational function, 90
  rational matrix, 107
<-, 29
<<, 28
>>, 28
[], 96
[, 96
], 96
$, 83
0 x 0 matrix, 96

abs()
  array, 135
  complex number, 76
  real number, 73
acos()
  array, 135
  complex number, 77
  real number, 73
acosh()
  array, 135
  complex number, 77
  real number, 73
addition, 173
all(), 119, 132
ans, 66
any(), 119, 132
appendlist(), 139, 140
arg(), 148
  complex number, 76
arguments, 53
arithmetic sequence, 100
Array, 95, 148
array of matrix, 118
array operator, 95
Array(), 61, 63, 95
asin()

B Operation **171**
B.1 Addition . . . 173
B.2 Subtraction . . . 173
B.3 Multiplication . . . 174
B.4 Scalar Multiplication . . . 174
B.5 Inverse . . . 176
B.6 Division . . . 176
B.7 Power . . . 178
B.8 Transpose . . . 179
B.9 Complex Conjugate Transpose . . . 180
B.10 Negation . . . 181
B.11 Increment and Decrement Operators . . . 181
B.12 Relational Operation . . . 181
B.13 Logical Operation . . . 184

Index **185**

`lower()`
polynomial, 86
polynomial matrix, 106
rational function, 92
rational matrix, 111
`lu()`, 131

m-files, 14
machine, 164
`main()`, 45, 49
`makelist()`, 138
mat, 28
MAT Data Format, 28, 163
MAT Format, 37
matc, 14, 23
matcc, 23
mated, 13, 14, 35, 37
`MatFrees()`, 25
`MatInit()`, 25
matlab, 14
`mat_last`, 25
Matrix, 95
`Matrix()`, 63, 83, 95
matx, 14, 21
MaTX-Lib, 14, 15
MaTX.log, 24
`MATXCPP`, 67
MATXINPUTS, 28, 52
`MATXINPUTS`, 69
`MaTXRC.mm`, 69
`max()`, 122, 133
array, 124
complex number, 76
integer, 73
real number, 73
`maximum()`, 122
`maxsing()`, 122
`mean()`, 122, 123, 133
`menu()`, 31
`min()`, 122, 133
array, 124
complex number, 76
integer, 73
real number, 73
`minimum()`, 122
`minsing()`, 122
MM-FILE, 22, 28, 52
mm-files, 14
multiple output, 51
multiplication, 174
mx, 29
MX Data Format, 29, 164

`NaN`, 72
negation, 181
`frobnorm()`, 122
`Nu()`, 90, 112
numerator, 90
numerator array, 112
numerical computation, 13
nyquist, 148

`Ode()`, 125, 150
`Ode45()`, 128
`Ode45Auto()`, 128
`Ode45Hybrid()`, 129
`Ode45HybridAuto()`, 129
`OdeAuto()`, 126, 156
`OdeHybrid()`, 126, 150
`OdeHybridAuto()`, 127
`OdeStop()`, 130
`ONE()`, 99
operating system, 18
operation, 171

pause, 32, 33

3.11 bell 32
3.12 Warning 32
3.13 Error 33
3.14 Help 33
3.15 Pause 33
3.16 Quit 34

4 Matrix Editor **35**
4.1 Edition of Real Matrices 36
4.2 Edition of Complex Matrices 36
4.3 Key Bindings 36
4.4 Matrix Editor (mated) 37

5 Graphing **41**
5.1 Basic Form 42
5.2 Multiple Lines 43
5.3 Logarithmic Plots 44

6 Variable **45**
6.1 Declaration of Variables 46
6.2 Scope Rule 46

7 Function **49**
7.1 Declaration of Functions 50
7.2 Definition of Functions 50
7.3 Multiple-Output Functions 51
7.4 Variable Number of Arguments 52
7.5 Calling Functions (matx) 52
7.6 Arguments of Functions 53
7.7 Return Statement 54
7.8 Reference to Functions 54
7.9 Inline Functions(matc) 55
7.10 Command-Line Arguments 55

8 Control Flow **57**
8.1 If-Else 57
8.2 Loops — While and For 58
8.3 Loops — Do-while 59

＜著者略歴＞

こ が まさのぶ
古賀 雅伸
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　東京工業大学情報理工学研究科助手

MATX Reference Manual

1995 年 1 月 5 日　初 版 発 行

執筆者　古賀 雅伸
発行者　古賀 雅伸
印刷者　古賀 雅伸
製本者
発行所　東京工業大学
〒 152 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。