Contents

List of Figures

which have the same precedence. Relationals have lower precedence than arithmetic operators.

`Integer1 < Integer2`	`Integer1 is less than Integer2`
`Integer1 > Integer2`	`Integer1 is bigger than Integer2`
`Integer1 <= Integer2`	`Integer1 is less than or equal to Integer2`
`Integer1 >= Integer2`	`Integer1 is bigger or equal to Integer2`
`Integer1 == Integer2`	`Integer1 is equal to Integer2`
`Integer1 != Integer2`	`Integer1 is not equal to Integer2`
`Real-Num1 < Real-Num2`	`Real-Num1 is less than Real-Num2`
`Real-Num1 > Real-Num2`	`Real-Num1 is bigger than Real-Num2`
`Real-Num1 <= Real-Num2`	`Number1 is less than or equal to Number2`
`Real-Num1 >= Real-Num2`	`Number1 is bigger or equal to Number2`
`Real-Num1 == Real-Num2`	`Real-Num1 is equal to Real-Num2`
`Real-Num1 != Real-Num2`	`Num1 is not equal to Num2`
`Matrix < 0`	`maxeigval(Matrix) < 0`
`Matrix <= 0`	`maxeigval(Matrix) <= 0`
`Matrix > 0`	`mineigval(Matrix) > 0`
`Matrix >= 0`	`mineigval(Matrix) >= 0`
`Matrix == 0`	`Matrix is a Zero Matrix`
`Matrix != 0`	`Matrix is not a Zero Matrix`
`0 < Matrix`	`mineigval(Matrix) > 0`
`0 <= Matrix`	`mineigval(Matrix) >= 0`
`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`

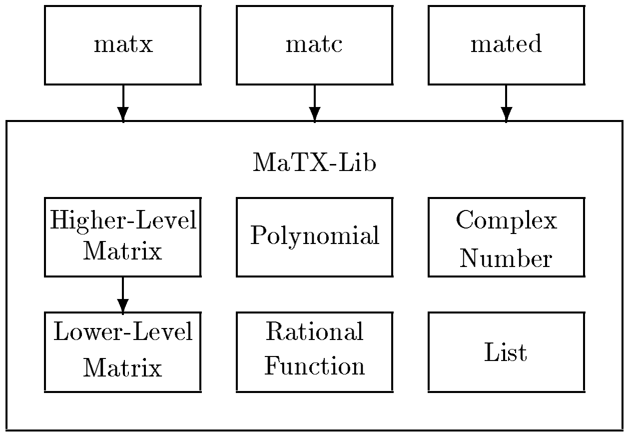

Figure 1.1: The Architecture of MATX

guage by itself. Matrix class library is composed of higher-level library and lower-level library. The lower-level library consists of nearly 300 functions, which provide fast access to the objects, but do little to make programming easier. It handles the operation of the objects and includes some optimizations that encourage efficient memory usage. The higher-level library is built upon lower-level library and consists of about 200 functions. The purpose of the higher-level library is to provide the supervisory layer that supports the user-interface facilities. It handles the memory allocation of object and calls suitable lower-level functions according to the attribute of the object.

Figure 1.2 indicates the internal process of MATX . A source file (`test.mm`) written in the MATX language is given to the lexical analyzer which recognizes the expressions in the input stream and partitions it into strings matching the MATX expressions. The Parser calls the lexical analyzer to pick up the partitioned basic items (tokens). These tokens are organized according to the MATX input structure rules; when one of these rules is recognized, the corresponding action for the rule is invoked. In each action, both the interpreter and the compiler defect type mismatches, inconsistent argument usage, unused or apparently uninitialized variables,

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

B.7 Power

When the index is negative, the operation means its reciprocal (inverse) to the absolute of the index. It should be noted that the inverse of a polynomial is a rational function.

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


Real Array             = Real Array ^ Real
Complex Array          = Real Array ^ Complex
Complex Array          = Complex Array ^ Real
Complex Array          = Complex Array ^ Complex


Real Matrix            = Real Array ^ Real Array
```


HP 9000/755 HP-UX
Mips RC6280 UMIPS4.52C
SPARC Station20 SunOS4.1.3, SOLARIS 2.X
SPARC Station10 SunOS4.1.3, SOLARIS 2.X
JCC JS5 SOLARIS 1.1.1
SPARC Station classic SOLARIS 2.X
Sun4 SunOS4.0.3, SunOS4.1.X
Sun3 SunOS4.0.3, SunOS4.1.X
NEWS-830(SONY) NEWS-OS 2.0, 3.3
J3100 UNIX System V/386 3.0, PC-DOS 3.1
Gateway 2000 LINUX (1.0.9, 1.1.18)
Gateway 2000 DOS/V, (go32 + gcc, djgcc)
EPSON Note 386BSD, FreeBSD2.0
PC9801(NEC) MS-DOS 3.X, (go32 + gcc, djgcc)
X68000(SHARP) Human68k (gcc + libc)

B.3 Multiplication

Even if the multiplication result of matrices becomes a `1 x 1` matrix, it is not a scalar value but still a matrix.

```
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)
Real Matrix             = Real Matrix .* Real(Mat|Arr)
Complex Matrix          = Complex Matrix .* Complex Matrix
Real-PolyMatrix         = Real-PolyMatrix .* Real-PolyMatrix
Comp-PolyMatrix         = Comp-PolyMatrix .* Comp-PolyMatrix
Real-Rat-FuncMatrix     = Real-Rat-FuncMatrix .* Real-Rat-FuncMatrix
Comp-Rat-FuncMatrix     = Comp-Rat-FuncMatrix .* Comp-Rat-FuncMatrix
```

B.4 Scalar Multiplication

```
Real(Mat|Arr)          = (Integer|Real Number) * Real(Mat|Arr)
Real(Mat|Arr)          = Real(Mat|Arr) * (Integer|Real Number)
Complex(Mat|Arr)       = (Integer|Real Number) * Complex(Mat|Arr)
Complex(Mat|Arr)       = Complex(Mat|Arr) * (Integer|Real Number)
Complex(Mat|Arr)       = Complex Number * Real(Mat|Arr)
Complex(Mat|Arr)       = Real(Mat|Arr) * Complex Number
Complex(Mat|Arr)       = Complex Number * Complex(Mat|Arr)
Complex(Mat|Arr)       = Complex(Mat|Arr) * Complex Number
```


execute MM-FILES like shell scripts and Perl script in UNIX. But the C-preprocessor facility is restrained.

[Example]

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

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

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

main();
```

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

2.2 How To Make Executable Programs

1. Create the source program in a file whose name extension is "`mm`", such as `test.mm`, using a favorite editor.

2. Compile it with the command "`matc test`", where the file-extension is optional on the command line. You can use the filename with another file-extension by specifying the option "`-nomm`." If you haven't botched anything, such as omitting a character or misspelling something, the compilation will proceed silently, and make a C language file called `test.c`. If you execute `matc` without arguments, it is invoked to become the interactive mode and shows a prompt "`matc (1)`". This mode is convenient for debugging the program. The source code of MATX is inserted in the C language code, if the option "-c" is specified.

Chapter 3

Commands

3.1 Print Out To Display

All class objects can be outputted to standard-output by **print** command. Multiple objects can be specified by separating objects with commas.

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

3.2 Read from Keyboard and Revision

All class objects can be inputted from standard-input by the **read** command. Multiple objects can be specified by separating objects with commas. If **read** command is used with matrices, the matrix-editor (mated) is invoked to read them.

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

Data Class	Name	Content
Matrix	cols	Column number
Integer	dummy	
Real Number	dummy	
Complex Number	dummy	
String	dummy	
Polynomial	dummy	
Rational	ddegree	Degree of Denominator
List	dummy	
Array	cols	Column number
Index	cols	Column number

4. The fourth integer depends on the class of data.

Data Class	Name	Content
Matrix	class	Class of matrix
Integer	dummy	
Real Number	dummy	
Complex Number	dummy	
String	dummy	
Polynomial	imagf	Real or Complex
Rational	imagf	Real or Complex
List	dummy	
Array	class	Class of matrix
Index	class	REAL NUMBER

5. The fifth integer indicates the length of the name of variable.

Data Class	Name	Content
Matrix	namelen	Length of name
Integer	namelen	Length of name
Real Number	namelen	Length of name
Complex Number	namelen	Length of name
String	namelen	Length of name
Polynomial	namelen	Length of name
Rational	namelen	Length of name
List	namelen	Length of name
Array	namelen	Length of name
Index	namelen	Length of name


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

3.7 Getting and Changing Environment Variable

The function "`putenv()`" change or add value to environment, and the function "`getenv()`" return value for environment name.

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

3.8 Generate amenu of choices

The function "`menu()`" displays the menu whose title is in the string variable (the first argument) and whose choices are string variables (the following arguments). `menu()` returns the value the user enters. We can specify the default item number of the menu as the second argument.

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

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

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

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

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

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

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

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

Chapter 4

Matrix Editor

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. Even if the user enters wrong characters such as alphabetic characters and special characters, it will cause no problem, since mated takes in only suitable characters. The matrix-editor mated has the following features.

1. To handle any size of matrices and change their size arbitrarily.
2. Automatically scrolls if necessary.
3. To cut and paste any subportion of the matrix.
4. The key binding is similar to that of famous editors, **vi** and **emacs**.
5. To read and write matrices from and to files.
6. Don't require special window-system facilities.
7. To do a few matrix operations.

17.5.4 Simulation of complex systems

We describe here how to write a program for simulation of complex systems. When we deal with the simulation of a complex system, it takes much time to write codes for extracting each state from an augumented state `X` at the top of the functions `diff_eqs()` and `link_eqs()`. Two functions `VectorConnect()` and `VectorChop()` are useful for it and are introduced in the following example program.

Each block P, D, C, and F are a plant, an uncertainty of the plant, a feedforward controller, and a feedback controller. They are described by a state equation and a output equation, where each coefficient of the equations are (Ap, Bp, Cp), (Ad, Bd, Cd), (Ac, Bc, Cc), and (Af, Bf, Cf).

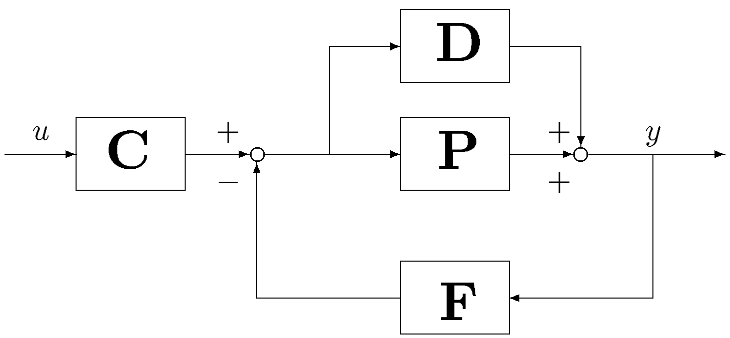

Figure 17.2: Simulation of a complex system

Table 4.1: Key Bindings of mated

Command	Emacs	Vi
Move forward by columns	^F	l
Move backward by columns	^B	h
Move backward by rows	^P	k
Move forward by rows	^N	j
Move to start of row	^A	^
Move to end of row	^E	$
Move to (first row, first column)	ESC <	g
Move to (last row, last column)	ESC >	G
Kill one row	^K r	d·r
Kill one column	^K c	d·c
Yank back from kill matrix	^Y	p
Redraw	^L	
Set mark	ESC SPACE	m
Copy submatrix to kill matrix	i	i
Yank back from kill matrix	o	o
Zero matrix		Z
Identity matrix		E
Real Part matrix		R
Imaginary Part matrix		I
Complex matrix		C
Transpose (Conjugate Transpose)		T
Read from a file	r	r
Write to a file	w	w
Exit from mated	q	q

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

  A = [[ 0   1   0]   /* System matrix   */
       [ 0   0   1]
       [-2, -3, -4]];

  b = [0 0 1]';       /* Input matrix    */

  dx = A * x + b * u; /* System equation */
}

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

  f = [-1, -1, -1];   /* State feedback control-law */
  u = f * x;
  return u;
}
```

List 6: Simulation of State Variable Feedback Control (2)

If `X` and `Y` are vectors of the same length, the command `plot(X,Y)` (`gplot(X,Y)`) draws an x-y plot of the elements of `X` versus the elements of `Y`. For example,

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

results in a sinusoidal plot of time, from `t = 0.0` to `t = 4*PI`.

5.2 Multiple Lines

There are two ways to plot multiple lines on a single graph. The first is to give `plot()` (`gplot()`) two arguments, as in `plot(X,Y)` (`gplot(X,Y)`), where either `X`, `Y`, or both, are matrices. Then

- If `Y` is a matrix, and `X` a vector, `plot(X,Y)` (`gplot(X,Y)`) successively plots the rows of `Y` versus the vector `X`, using a different line type for each.
- If `X` is a matrix, and `Y` a vector, then the above rules are applied except the family of lines from `X` are plotted versus vector `Y`.
- If `X` and `Y` are both matrices of the same size, `plot(X,Y)` (`gplot(X,Y)`) plots the rows of `X` versus the columns of `Y`.
- If no `X` is specified, as in `plot(Y)` (`gplot(Y)`), where `Y` is a matrix, then lines are plotted for each rows of `Y` versus the column index.

The second, and easier, way to plot multiple lines on a single graph is to use `plot` (`gplot`) with multiple arguments:

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

The variables, `X1`, `Y1`, `X2`, `Y2`, etc. are pairs of vectors. Each x-y pair is graphed, generating multiple lines on the plot. Multiple arguments have

17.5 Simulation

17.5.1 State Variable Feedback Control (1)

The following program does the continuous-time and hybrid-time simulation of a linear system

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

to which the state feedback control

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

is applied.

Hybrid-time simulation means the simulation of continuous-time plants with the digital controller. This list is the program of the simulation. The functions `Ode()` and `OdeHybrid()` are capable of integrating a broad range of non-linear system functions of both explicit and implicit nature. The function `Ode()` does the continuous-time simulation and the function `OdeHybrid()` does the hybrid simulation. The former function takes five arguments: the initial time (`0.0`), the terminal time (`5.0`), a vector initial condition (`x0`), the reference (`diff_eqs`) to the function which calculates the value of the derivative (`dx`) from the time `t` and the values of `x` and `u` at this time, the reference (`link_eqs`) to the function which calculates the value of the input (`u`) from the time `t` and the value of `x` at this time. The last argument (`link_eqs`) can be omitted in case of zero input. In addition to the five arguments, the function `OdeHybrid()` requires the sampling interval (`dt`) between which the constant input (`u`) is applied to the system.

(its type, size, etc.); a definition also causes storage to be allocated. In case of the interpreter (matx), a declaration does not only announcement of the properties of a variable but also its storage.

When one use the interpreter (matx), in the outside of functions any kinds of data can be assigned to an arbitrary variable. In other words the class of the variable in the left-hand side may be different from that of the data in the right-hand side. It is a special exception for providing users with convenient session.

17.3 Transfer Function

The transfer function of a linear system

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

is calculated according to Faddev's algorithm as follows.

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

where

$$\det(sI - A) = s^n + \alpha_{n-1}s^{n-1} + \alpha_{n-2}s^{n-2} + \cdots + \alpha_0$$

and

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

Here, $\mathrm{tr}(X)$ denotes the sum of diagonal elements of X.


```
Func ClassName FunctionName(arg1, arg2, arg3, ..., argN)
ClassName1 arg1, arg2;
ClassName2 arg3;
...............
ClassNameM argN;
{
    Declaration of variables.

    Statements.

    return;       // void function
    return value; // Otherwise
}
```

7.3 Multiple-Output Functions

In order to implement the multiple-output functions, we introduce the "List Class". Multiple-output function calls have the form like this

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

"`a, b, c`" are returned values and "`d, e, f`" are arguments of the function. Function "`fun()`" returns multiple output values as a list which contains them. The number of left-hand-side arguments must be less than or equal to that of the elements of the returned list. If the number of left-hand-side arguments is less than that of the elements of the list, the only top elements of the list which correspond to the left-hand-side arguments are used and the rest are discarded.

17.1 Continuous-Time Riccati Equations

The solution of riccati equation of continuous-time linear systems can be obtained by solving the eigenvalue problem of a Hamilton matrix [4]. Since the function `eigvec()` returns eigenvectors ordered according to the magnitude of the real-part of the eigenvalues, the statements on the 20th and 21st line extract the eigenvectors corresponding to the stable eigenvalues from the matrix `vec`. The following statements are explanations of the list.

- On the 1st and 11th line, beginning of functions.
- On the 3rd and 14th, declaration of local variables.
- On the 6th line, reading matrices from standard input.
- On the 7th line, call the function `CRiccati()`.
- On the 8th line, displaying the solution matrix `P`.
- On the 12th line, declaration of arguments of the function.
- On the 16th line, making a Hamilton matrix [4].
- On the 19th line, calculation of eigenvectors.
- On the 20th and 21st line, partitioning matrix `vec` using a piece-matrix `A`.

7.9 Inline Functions(matc)

Once an expression is described with some variables, this expression is reusable with other parameters like a macro since the compiler matc holds its definition as well as the value in the effective scope.

```
r1 = 1.0;
M1 = [r1 2.0];

r1 = 2.0;
M2 = @M1;  // M2 = [2.0 2.0];
```

7.10 Command-Line Arguments

In environments that support MaTX , there is a way to pass command-line arguments or parameters to a program when it begins executing. When "`main()`" is called to begin execution, it is called with two arguments. The first is the number of command-line arguments the program was invoked with; the second is a list that contain the arguments, one per string. The first string is the name of command. In order to pass an integer or a real number, the class conversion functions are exploited. For example,

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

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

prints out the command name and the sum of the first and second command-line argument integers.

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

produces a list which contains a real-number, complex-number, and a real-matrix. The function "`makelist()`" makes a list which contains arbitrary number of elements. The elements are undefined.

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

16.2 List Elements

Individual list elements can be specified with the index and the class name of the element separated by a comma inside parentheses "(" and ")". (Index begins from `1`.)

```
x = {sqrt(3.0), ((1.0+2.0)*4.0,:), [1 2] + [3 4]};
a = x(1, Real);     /* a = 1.732 */
b = x(2, Complex);  /* b = (12,0) */
```

Any data can be assigned to an element of the list which is specified with the index inside parentheses on the left-hand side. (Index begins from `1`.) And, a list on the right-hand side can be assigned to the position which is specified with an interval inside parentheses on the left-hand side. The length of the list on the right-hand side must be more than or equal to the length of the specified interval. If the length of the list on the right-hand side is less than that of the specified interval, only the first corresponding elements are assigned to the list on the left-hand side and the others are discarded.

8.3 Loops — Do-while

The `do-while` loop tests at the bottom after making each pass through the loop body; the body is always executed at least once. The syntax is

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

The `statements` are executed, then `expr` is evaluated. If it is true, `statements` are executed again, and so on. If the expression becomes false, the loop terminates.

8.4 Switch

The `switch` statement is a spcial multi-way decision maker that tests whether an expression matches onee of a number of constant values, and branches accordingly.

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

The `switch` evaluates the integer expression in parentheses and compares its value to all the cases. Each case must be labeled by an integer constant. If a case matches the expression value, execution starts at that case. The case labeled `default` is executed if none of the other cases if satisfied. The `break` statement causes an immediate exit from the `switch`.

8.5 Break and Continue

It is sometimes convenient to be able to control loop exits other than by testing at the top or bottom.

```
Array  vec2diag(Array M1);   Col-vector to diagonal array
Array  vec2diag(Array M1, Integer k);
                            Col-vector to diagonal array
                            (K > 0 is above the main diagonal)
                            (K < 0 is below the main diagonal)
Array  diag2vec(Array M1);   Diagonal array to col-vector
Array  diag2vec(Array M1, Integer k);
                            Diagonal array to col-vector
                            (K > 0 is above the main diagonal)
                            (K < 0 is below the main diagonal)
Array  fliplr(Array M1);     Flip arrays left-right
Array  flipud(Array M1);     Flip arrays up-down.
List   sort(Array M1);       Sorting in ascending order.
                             For complex arrays,
                             sorted by abs(A1)
      {A2, idx} = sort(A1);  A2:  Result array
                             idx: Indices used in the sort.
                             A2 = A1(idx)
List size(Array A1);         Row number and column number
                             {row, col} = size(A);
Array   rand(Array A);       Uniformly distributed array
                             as large as array A
Array   randn(Array A);      Normaly distributed  array
                             as large as array A
Array max(Array A1, Array A2); An array the same size as A1
                               and A2 with the largest elem-
                               ents taken from A1 or A2
Array min(Array A1, Array A2); An array the same size as A1
                               and A2 with the smallest elem-
                               ents taken from A1 or A2


Array fft(Array data);             Fast Fourie Transforms
Array fft(Array data, Integer n);  Fast Fourie Transforms
                                   n = 2^m
Array ifft(Array data);            Inverse Fast Fourie
                                   Transforms
Array ifft(Array data, Integer n); Inverse Fast Fourie
```



```
CP1 = CoPolynomial(Real Polynomial P) Re(CP1) = P, Im(CP1) = 0

R1 = Rational(Real Polynomial)  Numerator is the polynomial,
                                Denominator is 1.
R1 = Rational(Real Matrix M1)   Numerator is Polynomial(M1),
                                Denominator is 1.
R1 = Rational(Real Number r)    Numerator is Polynomial(r1),
                                Denominator is 1.
CR1 = Rational(Complex Number c)
                                Numerator is Polynomial(c1),
                                Denominator is 1.

CR1 = CoRational(Real Rational R) Re(CR1) = R, Im(CR1) = 0

M1 = Matrix(Array)              Array to Matrix.
M1 = Matrix(Polynomial)         Coefficients are identical
                                with the elements of the row
                                vector in ascending order.
M1 = Matrix(Real Number r)      1x1 real matrix.
CM1 = Matrix(Complex Number c)  (1,1) complex matrix.

CM1 = CoMatrix(Real Matrix|Array)       Re(CM1) = M, Im(CM1) = 0
PM1 = PoMatrix(Real Matrix|Array)       Const. term matrix is M
CPM1 = PoMatrix(Compex Matrix|Array)    Const. term matrix is M
RM1 = RaMatrix(Real Matrix|Array)       Const. term matrix of
                                        numerator matrix is M
CRM1 = RaMatrix(Complex Matrix|Array)   Const. term matrix of
                                        numerator matrix is M
RM1 = RaMatrix(Polynomial Matrix|Array) Numerator matrix is M

A1 = Array(Matrix)              Matrix to Array.
A1 = Array(Polynomial)          Coefficients are identical
                                with the elements of the row
                                vector in ascending order.
A1 = Array(Real Number r)       1x1 real array.
CA1 = Array(Complex Number c)   1x1 complex array.
```

```
                    : dtsav=1.0E-3,
                      dtmin=EPS, dtmax=abs(t1-t0)/5
List Ode45HybridAuto(Real t0, Real t1, Real dt, Matrix x,
             void dx(), void input(), Real eps, Real dtsav)
             : dtmin=EPS, dtmax=abs(t1-t0)/5
List Ode45HybridAuto(Real t0, Real t1, Real dt, Matrix x,
             void dx(), void input(), Real eps,
             Real dtsav, Real dtmin)
             : dtmax=abs(t1-t0)/5
List Ode45HybridAuto(Real t0, Real t1, Real dt, Matrix x,
             void dx(), void input(), Real eps,
             Real dtsav, Real dtmin, Real dtmax)
     To solve ordinary differential equations according to
     RKF45 methods automatically changing
     the step to guarantee the prespecified error between
     [t0 t1], where the external input is constant
     for the prespecified sampling interval.
     And the minimum and maximum step are dtmin and dtmax.
     {T,X,U} = Ode45HybridAuto();

void OdeStop();        Stop the execution of functions:
                       Ode(), OdeAuto(), OdeHybrid(),
                       OdeHybridAuto(), Ode45(), Ode45Auto(),
                       Ode45Hybrid(), Ode45HybridAuto()

List size(Matrix M1);  Row number and column number
                        {row, col} = size(A);
List eig(Matrix M1);    Eigen Problem
                        {val, vec} = eig(A);
                        A * vec = vec * val;
                        val: Eigenvalues(diagonal matrix)
                        vec: Eigenvectors
List eig(Matrix M1, Matrix M2);
                        Generalized Eigen Problem
                        {val, vec} = eig(A, B);
                        A * vec = B * vec * val;
                        val: Generalized eigenvalues
```


one uses four complex numbers as elements of the matrix. And entering the statements

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

results in the output

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

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

9.6 Comment

Any characters between "`/*`" and "`*/`", and any characters after "`//`" to the end of the line are ignored by the compiler and the interpreter; they may be used freely to make a program easier to understand. Comments may appear anywhere a blank or newline can. Comments between "`/*`" and "`*/`" can not be nested.

9.7 Preprocessor

MATX provides certain language extensions by means of the C preprocessor. The `#define` capability which we have used is the most common of these extensions; another is the ability to include the contents of other files. The default C preprocessor is `cpp`, and another preprocessor can be used by specifying its name with the environmental variable `MATXCPP`.

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

If "`-nocpp`" flag is specified on the command line, no preprocessor is used.

```
     to 4th order Runge-Kutta methods with the prespecified
     step 'h' between [t0 t1].
     {T,X,U} = Ode();

List OdeAuto(Real t0, Real t1, Matrix x, void dx());
              : eps=1.0E-6, dtsav=1.0E-3,
                dtmin=EPS, dtmax=abs(t1-t0)/5
List OdeAuto(Real t0, Real t1, Matrix x, void dx(),
             void input());
              : eps=1.0E-6, dtsav=1.0E-3
                dtmin=EPS, dtmax=abs(t1-t0)/5
List OdeAuto(Real t0, Real t1, Matrix x, void dx(),
             void input(), Real eps);
             : dtsav=1.0E-3, dtmin=EPS, dtmax=abs(t1-t0)/5
List OdeAuto(Real t0, Real t1, Matrix x, void dx(),
             void input(), Real eps, Real dtsav);
             : dtmin=EPS, dtmax=abs(t1-t0)/5
List OdeAuto(Real t0, Real t1, Matrix x, void dx(),
             void input(), Real eps,
             Real dtsav, Real dtmin);
             : dtmax=abs(t1-t0)/5
List OdeAuto(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
     4th order Runge-Kutta 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} = OdeAuto();

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

Chapter 10

Integer and Real Number

Declaration:

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

Example:

```
0, 3, -4                     (Integer)
0., 0.0, 3., 3.0, -4. -4.0   (Real-Number)
3E4, 3.E4., 3e4, 3.e4.       (Real-Number)
```

MATX distinguish between **Integer** and **Real Number** `1`is an integer and `1.0` is a real number. In MATX , as many other languages, integer division truncates, so any fractional part is discarded. All real number arithmetic in MATX is done in double precision.

MATX provides five special constants `PI, E, GAMMA, DEG, PHI`. as primitives.

```
                               (True = 1, False = 0)
Integer  isscalar(Matrix M1);  Is scalar value ?
                               (True = 1, False = 0)
Array    finite(Matrix M1);    ones where the elements of M1
                               are finite and zeros where
                               they are infinite or NaN.
Array    isnan(Matrix M1);     ones where the elements are NaN
                               zeros where they are not.
Matrix   sgn(Matrix M1);       Element-by-element sgn()

Real max(Matrix M1);      Maximum element
Real min(Matrix M1);      Minimum element
List maximum(Matrix M1);  Maximum element and its location
                          {r1, i, j} = maximum(M1);
                          r1 == M1(i,j)
List minimum(Matrix M1);  Minimum element and its location
                          {r1, i, j} = maximum(M1);
                           r1 == M1(i,j)
Real maxsing(Matrix M1);   Maximum singular value
Real minsing(Matrix M1);   Minimum singular value
Real det(Matrix M1);       Determinant
Real sum(Matrix M1);       Sum of all elements
Real prod(Matrix M1);      Product of all elements
Real mean(Matrix M1);      Mean of all elements
Real trace(Matrix M1);     Trace of real matrix
Real frobnorm(Matrix M1);  Frobenius norm
                           (Euclidean norm)
Real infnorm(Matrix M1);   Infinity norm
Real norm(Matrix M1, Integer p);
                               M1 : Matrix
                               p=1 1-norm
                               p=2 maximum singular value
                               M1 : Vector p-norm
Real norm(Matrix M1);          norm(Matrix M1, 2)

Complex max(Matrix M1);     Maximum element of a complex matrix
Complex min(Matrix M1);     Minimum element of a complex matrix
```

Chapter 11

Complex Number

Declaration:

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

Example:

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

11.1 How To Describe Complex Numbers

Complex numbers are entered as a couple of real numbers separated by a comma "," inside parentheses, "(" and ")". Parentheses and a comma is call a **complex operator**. The real-part or imaginary-part can be omitted by using "*" or ":" instead. If the element is an integer, it is converted to a real number.

```
c1 = (r1,r2) /* c1 = r1 + r2*i; (r1, r2: real numbers) */
c1 = (r1, *) /* c1 = r1 + 0*i;  (r1: real number)      */
c1 = (r1, :) /* c1 = r1 + 0*i;  (r1: real number)      */
c1 = (*, r1) /* c1 = 0  + ri*i; (r1: real number)      */
c1 = (:, r1) /* c1 = 0  + ri*i; (r1: real number)      */
```

matrix M of MM

`M1(IDX2)`	`= M2;`	Set M2 at IDX2-elem of M1
`M1(1,IDX2)`	`= M2;`	Set M2 at 1-row, IDX2-col of M1
`M1(IDX1,IDX2)`	`= M2;`	Set M2 at IDX1-row, IDX2-col of M1
`M1(IDX1,1)`	`= M2;`	Set M2 at IDX1-row, 1-col of M1
`M1(IDX1,1:2)`	`= M2;`	Set M2 at IDX1-row, 1st-2nd-col of M1
`M1(1:2,IDX2)`	`= M2;`	Set M2 at 1st-2nd-row, IDX2 of M1
`MM(IDX2,M)`	`= M2;`	Set M2 at IDX2-elem for a piece matrix M of M1
`MM(IDX1,IDX2,M)`	`= M2;`	Set M2 at IDX1-row, IDX2-col for a piece matrix M of M1
`MM(1,IDX2,M)`	`= M2;`	Set M2 at 1-row, IDX2-col for a piece matrix M of M1
`MM(IDX1,1,M)`	`= M2;`	Set M2 at IDX1-row, 1-col for a piece matrix M of M1

15.5.6 Piece-Matrix (Array of Matrix)

Matrices may be considered as 1-dimensional or 2-dimensional arrays of a certain object, such as real numbers, complex number, polynomials, and rational functions, while arrays of matrix can be represented by introducing the piece-matrix notion. The piece-matrix is used to specify the unit size of arrays of matrix. List class is available to describe an array which contains various size matrices. For example,

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

lead to a 4×6 zero matrix `AA`. And

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

Chapter 12

String

Declaration:

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

Example:

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

A string constant is a sequence of zero or more characters surrounded by double quotes “"” and “"”. The quotes are not part of the string, but serve only to delimit it. The sequence “`\t`” and “`\n`” in the string is MaTX notation for the tab and the newline character, respectively. Strings can be concatenated into larger strings with “+” operators.

12.1 Element and Elements

`S(1)`	1st element of a string S
`S(2:4)`	From 2nd to 4th elements of a string S

`M1(1:2,2:3)`	1st-2nd rows and 2nd-3rd columns
`M1(2:3,*)`	2nd-3rd rows
`M1(2:3,:)`	2nd-3rd rows
`M1(2:3,4)`	2nd-3rd rows and 4th column
`M1(*,2:3)`	2nd-3rd columns
`M1(:,2:3)`	2nd-3rd columns
`M1(4,2:3)`	4th row and 2nd-3rd columns

15.5.2 Setting Elements

Real numbers as well as complex numbers can be assigned to a complex matrix. However, only real numbers may be assigned to a real matrix.

`M1(1,1) = 1.0;`	Set 1.0 to (1,1)-element
`M1(2) = 1.0;`	Set 1.0 to (1,2)-element At the first row, the row number can be omitted
`CM1(1,1) = (1, 2);`	Set (1, 2) to (1,1)-element
`M1(0,0,M2) = M3;`	(1,1) is the beginning of right-hand side matrix according to the piece matrix M2
`M1(1,2,M2) = M3;`	(2,3) is the beginning of right-hand side matrix according to the piece matrix M2
`M1(1,M2) = M3;`	(1,2) is the beginning of right-hand side matrix according to the piece matrix M2
`M1(2,M2) = M3;`	(1,3) is the beginning of right-hand side matrix according to the piece matrix M2
`M1(3:5) = M3;`	Set martix M3 at 3rd-5th elements of matrix M1
`M1(1,*) = M3;`	Set martix M3 at 1st row of matrix M1
`M1(1,:) = M3;`	Set martix M3 at 1st row of matrix M1
`M1(*,2) = M3;`	Set martix M3 at 2nd column of matrix M1

Chapter 13

Polynomial

Declaration:

```
Polynomial a, b, c;
```

Example:

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

MATX handles polynomials of one variable with real and complex coefficients. Polynomials can be added, subtracted, multiplied, and divided as usual. At the moment, the coefficients of a polynomial are stored as a row vector containing the coefficients ordered by ascending powers. Polynomials are easily created and manipulated in MaTX.

13.1 How To Describe Polynomials

At least one polynomial variable must be defined first in order to describe polynomials. Polynomial variable is defined by using the symbol "`$`" and the character "`s`" is always used for the output polynomial variable. Integer coefficients are converted to real numbers. The class conversion function "`Polynomial()`" and "`Matrix()`" can be used to convert a row vector to a polynomial and visa versa. The coefficients of the polynomial are identical with the elements of the row vector in ascending order.

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

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

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

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

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


13.7 Comparison of polynomials

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

```
poly1 == poly2   poly1 is equal to poly2
poly1 != poly2   poly1 is not equal to poly2
```

13.8 Polynomial Functions

MATX provides the following polynomial functions.

```
Matrix  roots(Polynomial P1);     Root of polynomial
Integer degree(Polynomial P1);    Degree of polynomial
Integer isreal(Polynomial P1);    Classification of poly
                                  (True = 1, False = 0)
Integer iscomplex(Polynomial P1); Classification of poly
                                  (True = 1, False = 0)
```

15.3.3 Derivative and Indefinite Integral

Derivatives and indefinite integrals of polynomial matrices can be obtained by using the functions "`derivative()`" and "`integral()`", respectively. The order of the derivative and the indefinite integral can be specified as the argument of the functions.

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

CPM1 = (PM1, PM2);
CPM2 = integral(CPM1);     /* 1st order indefinite */
CPM3 = integral(CPM1,2);   /* 2nd order indefinite */
CMP4 = derivative(CMP2);   /* 1st order derivative */
CMP5 = derivative(CMP3,2); /* 2nd order derivative */
```

15.3.4 Shift of Coefficients

The coefficients of the polynomial matrices 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 functions.

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

CPM2 = higher(CPM1);   /* One shift to higher order  */
CPM3 = higher(CPM1,2); /* Two shifts to higher order */
CPM4 = lower(CPM2);    /* One shift to lower order   */
CPM5 = lower(CPM3,2);  /* Two shift to lower order   */
```



```
P1 = Nu(P1 / P2);  /* Numerator   */
P2 = De(P1 / P2);  /* Denominator */
```

14.4 Operations of Rational Functions

Operation of rational-functions are done according to the following rules, and no cancellation is applied to the operation. Please pay attention to the increase of degree of rational-functions.

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

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

14.5 Evaluation

Evaluation of rational-functions at a specified value can be done by using the function "`eval()`".

```
s  = $;
R1 = (s + 2)/(2*s + 3);
r1 = eval(R1, 2.0);          /* Eval at 2.0              */
R2 = eval(R1, s + 1);        /* Eval at s+1              */
R2 = eval(R1, (s+1)/(s+2)); /* Eval at (s+1)/(s+2)      */
A2 = eval(R1, [1:4]);        /* Eval at [1,2,3,4]        */
M2 = eval(R1, M);            /* (M+2*I(M))/(2*M+3*I(M)) */
```

Evaluation of a rational-function may involve a class conversion according to the following conversion rules.

```
Real number      = eval(Real-rat-func, Real number)
Complex number   = eval(Real-rat-func, Complex number)
```

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

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

15.3 Polynomial Matrix

15.3.1 How To Describe Polynomial Matrices

Essentially the manipulation of polynomial matrices is identical with that for manipulation for numerical matrices, i.e., real matrices and complex matrices. There are at least two convenient ways to enter polynomial matrices. One uses polynomials as elements of a matrix. The other uses matrices as coefficients of a polynomial.

```
s   = $;

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

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

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

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

Chapter 15

Matrix, Array, and Index

Declaration:

```
Matrix A, B, C;
Array D, E, F;
Index II, JJ;
```

Example:

```
[[1 2][3 4]], [(1,2), (3,4)]
Array([[1 2][3 4]]), Array([(1,2), (3,4)])
Index([1 2 3 4]), Index([10 8 6 4 2])
```

MATX distinguishes Array class objects from Matrix class objects. Array class objects have the same data structure as that of Matrix class objects, but arithmetic operations of arrays are element-by-element in stead of the usual algebraic matrix operations. For array operations of matrices, Matrix class has array operators denoted by the symbols, `.*`, `./`, `.\`, `.^`. The functions "`Matrix()`" is used to convert Matrix class objects to Array class object and the function "`Array()`" vice versa. The subscripting abilities of MATX allow manipulation of rows, columns, individual elements, and submatrices of matrices.

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()`".

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

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

15.2 Miscellaneous Matrices

MaTX provides a low of ways to enter various kinds of matrices..

15.2.1 Zero Matrices

The piece matrix is used to specify the unit size of big zero matrices.

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


15.2.2 Identity Matrices

In case of nonsquare matrix, all diagonal elements are 1 and others are 0. In case of polynomial matrix, all diagonal elements are polynomials whose constant term is 1 and other coefficients are 0. In case of rational function matrix, all diagonal elements are rational functions whose constant term is 1 and other coefficients are 0. The piece matrix is used to specify the unit size of big identify matrices.

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

15.2.3 Matrix Filled with “1”

In case of polynomial matrix, all elements are polynomials whose constant term is 1 and other coefficients are 0. In case of rational function matrix, all elements are rational functions whose constant term is 1 and other coefficients are 0. The piece matrix is used to specify the unit size of big matrices.

```
M1 = ONE(3);         3x3 real matrix.
M1 = ONE(3, c);      3x3 complex matrix,  c = Complex Number
```

```
CPM1 = (PM1,PM2); /* CPM1 = PM1 + PM2 i,          */
                  /* PM1, PM2: real poly matrices */
CPM1 = (PM1,*);   /* CPM1 = PM1 + Z(PM1) i,       */
                  /* PM1: real polynomial matrix. */
CPM1 = (PM1,:);   /* CPM1 = PM1 + Z(PM1) i,       */
                  /* PM1: real polynomial matrix. */
CPM1 = (*,PM1);   /* CPM1 = Z(PM1) + PM1 i,       */
                  /* PM1: real polynomial matrix. */
CPM1 = (:,PM1);   /* CPM1 = Z(PM1) + PM1 i,       */
                  /* PM1: real polynomial matrix. */
```

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

```
PM1 = Re((PM1, PM2));  /* Real part matrix      */
PM2 = Im((PM1, PM2));  /* Imaginary part matrix */
```

Create a complex polynomial matrix from two real polynomial matrices.

```
s   = $;

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

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

15.3.2 Evaluation of Polynomial Matrices

Evaluation of polynomial 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 polynomial matrix. For example, substituting a matrix into the variable

14.2 Complex Rational Functions

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

```
CR1 = (R1,R2); /* CR1 = R1 + R2*i; (R1, R2:real rat) */
CR1 = (R1,*);  /* CR1 = R1 + 0*i;  (R1: real rat)    */
CR1 = (R1,:);  /* CR1 = R1 + 0*i;  (R1: real rat)    */
CR1 = (*,R1);  /* CR1 = 0  + R1*i; (R1: real rat)    */
CR1 = (:,R1);  /* CR1 = 0  + R1*i; (R1: real rat)    */
```

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

```
R1 = Re((R1, R2));  /* Real part      */
R2 = Im((R1, R2));  /* Imaginary part */
```

where

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

and, a, b, c, d are real polynomials.

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

14.3 Numerator and Denominator

Numerator and denominator of a rational-function can be extracted by using the functions, "Nu()" and "De()", respectively.

15.4 Rational Function Matrix

15.4.1 How To Describe Rational Function Matrices

Essentially the manipulation of rational function matrices is identical with that for manipulation for numerical matrices, i.e., real matrices and complex matrices. There are at least two convenient ways to enter rational function matrices. One uses rational functions as elements of a matrix. The other uses two polynomial matrices and an array operation.

```
s   = $;

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

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

A = [[1 3][5 7]];  B = [[2 4][6 8]];
C = (ONE(2)*x + A) ./ (ONE(2)*x + B);
```

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

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

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

13.5 Derivative and Indefinite Integral

Derivatives and indefinite integrals of polynomials can be obtained by using the functions "**derivative()**" and "**integral()**", respectively. The order of the derivative and the indefinite integral can be specified as the argument of the functions. The function used without 2nd argument produce 1st derivative or 1st indefinite integral

```
s  = $;
P1 = 3*s^2 + 4*s + 5;
P2 = integral(P1);     /* 1st order indefinite integral */
P2 = integral(P1,2);   /* 2nd order indefinite integral */
P3 = derivative(P2);   /* 1st order derivative          */
P3 = derivative(P2,2); /* 2nd order derivative          */
```

13.6 Shift of Coefficients

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

```
s  = $;
P1 = 3*s^2 + 4*s + 5;
P2 = higher(P1);    /* 1 shift  for the higher order */
P2 = higher(P1,2);  /* 2 shifts for the higher order */
P3 = lower(P2);     /* 1 shift  for the lower  order */
P3 = lower(P2,2);   /* 2 shifts for the lower  order */
```



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

15.4.3 Derivative

Derivatives of rational function matrices can be obtained by using the functions "**derivative()**". The order of the derivative can be specified as the argument of the functions. At the moment, the indefinite integral of rational function matrices is not implemented.

```
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 = derivative(CRM1);   /* 1st order derivative */
CRM3 = derivative(CRM1,2); /* 2nd order derivative */
```

15.4.4 Shift of Coefficients

The coefficients of the rational matrices 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 functions.

```
Integer printf(String format, ...);   Display (matx).
Integer fprintf(String file, String format, ...); To files.
String  sprintf(String format, ...);  Conversion to a string.
Integer length(String str);           Length of a string.
```


`M1(:,2) = M3;`	Set martix M3 at 2nd column of matrix M1
`M1(1:2,1:2) = M3;`	Set matrix M3 at 1st-2nd row and 1st-2nd columns of matrix M1
`M1(1:,1:) = M3;`	(1,1) is the beginning of M3
`M1(1:,1) = M3;`	(1,1) is the beginning of M3
`M1(1,1:) = M3;`	(1,1) is the beginning of M3
`M1(1,2:3) = M3;`	Set matrix M3 at 1st row and 2nd-3rd columns of matrix M1
`M1(*,2:3) = M3;`	Set matrix M3 at 2nd-3rd columns of M1
`M1(:,2:3) = M3;`	Set matrix M3 at 2nd-3rd columns of M1
`M1(2:3,1) = M3;`	Set matrix M3 at 2nd-3rd rows and 1st column of matrix M1
`M1(2:3,*) = M3;`	Set matrix M3 at 2nd-3rd rows of M1
`M1(2:3,:) = M3;`	Set matrix M3 at 2nd-3rd rows of M1

15.5.3 Removing rows and columns

An efficient way of removing rows and columns of a matrix is to assign them to an empty matrix.

`M1(1,*)   = [];`	Delete 1st row of M1
`M1(2:3,*) = [];`	Delete 2nd and 3rd rows of M11
`M1(*,1)   = [];`	Delete 1st column of M1
`M1(*,2:3) = [];`	Delete 2nd and 3rd columns of M11
`M1(2:3)   = [];`	Delete (1,2)-,(1,3)-element of M1 At the first row, the row number can be omitted

15.5.4 Reshape of Matrices

The dimensions of matrices can be changed by using the function `reshape()` with the new dimensions of a matrix. The element ordering in the matrix is left to right and top to bottom which explains the ordering of the rearranged matrix in M1.

result in

$$\texttt{AA} = \left[\begin{array}{ccc} \begin{matrix}1 & 2\\ 3 & 4\end{matrix} & \texttt{Z}(2) & \texttt{Z}(2) \\ \texttt{Z}(2) & \texttt{Z}(2) & \begin{matrix}5 & 6\\ 7 & 8\end{matrix} \end{array}\right]$$

where matrix `A` is called **piece-matrix** of matrix `AA`, and the index of arrays of matrix start from 0. Using another piece-matrix `B (= [1])` instead of $\hat{A}$ and

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

yield another result:

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

15.6 Comparison of Matrices and Arrays

We can check the positivity of a matrix and compare two matrices by using relational operators `>`, `>=`, `<`, `<=`. If an array is compared, the comparison is done element-by-element and the result is an array with (0—1) elements. This result can be used for the argument of the functions, `all()` and `any()`.

```
Matrix < 0          maxeigval(Matrix) < 0
Matrix <= 0         maxeigval(Matrix) <= 0
Matrix > 0          mineigval(Matrix) > 0
Matrix >= 0         mineigval(Matrix) >= 0
Matrix == 0         Matrix is a Zero Matrix
Matrix != 0         Matrix is not a Zero Matrix
0 < Matrix          mineigval(Matrix) > 0
0 <= Matrix         mineigval(Matrix) >= 0
```

```
Real randn("seed");   Seed of n-distributed number
Real randn("seed", Integer val);
                      Set the seed of n-dist. number
Real fact(Real r1);   Factorial function
Real ceil(Real r1);   Rounding function
Real floor(Real r1);  Rounding function
Real fix(Real r1);    Rounding function towards zero
Real round(Real r1);  Rounding function to
                      the nearest integers
Real sgn(Real r1);    Sign function (r1 > 0.0, 1.0;
                                     r1 = 0.0, 0.0;
                                     r1 < 0.0, -1.0)
Integer finite(Real r1);  returns 1 if r1 are finite or zero,
                          and 0 if r1 are infinite or NaN
Integer isnan(Real r1);   returns 1 if r1 are NaN,
                          and 0 if r1 are not.
```



```
List maximum(Matrix M1);    Maximum element and its location
                            {c1, i, j} = maximum(M1);
                            c1 == M1(i,j)
List minimum(Matrix M1);    Minimum element and its location
                            {c1, i, j} = maximum(M1);
                            c1 == M1(i,j)
Complex det(CMatrix CM1);   Determinant
Complex trace(CMatrix CM1); Trace
Complex sum(CMatrix CM1);   Sum of all elements
Complex prod(Matrix M1);    Product of all elements
Complex mean(CMatrix CM1);  Mean of all elements


Polynomial det(PMatrix PM1);      Determinant
Polynomial trace(PMatrix PM1);    Trace
Polynomial sum(PMatrix PM1);      Sum of all elements
Polynomial prod(Matrix M1);       Product of all elements
Polynomial mean(PMatrix PM1);     Mean of all elements
Polynomial frobnorm(PMatrix PM1); Frobenius norm


Rational  det(RMatrix RM1);       Determinant
Rational  trace(RMatrix RM1);     Trace
Rational  sum(RMatrix RM1);       Sum of all elements
Rational  prod(Matrix M1);        Product of all elements
Rational  mean(RMatrix RM1);      Mean of all elements
Rational  frobnorm(RMatrix RM1);  Frobenius norm


Matrix eigval(Matrix M1);            Eigenvalues
Matrix eigval(Matrix M1, Matrix M2); Generalized eigenvalues
Matrix eigvec(Matrix M1);            Eigenvectors
Matrix eigvec(Matrix M1, Matrix M2); Generalized eigenvectors


Matrix  singval(Matrix M1);      Singular values
Matrix  singleftvec(Matrix M1);  Left singular vectors
Matrix  singrightvec(Matrix M1); Right singular vectors
```

(b) "MaTXRC.mm" in the directory which is defined by the environmental variable "MATXINPUTS"
(Multiple directories separated with colons ":" can be specified. For MS-DOS version, multiple directories are separated with semicolons ";")

(c) "MaTXRC.mm" in the default MM-FILE directory, which is determined at the compile time.

where, only one startup file is loaded.

9.9 Error Message

If any erros are detected by matx or matc, an error message is displayed on the screen and more detail error information is written out to the logfile "MaTX.log". There are two kinds of error messages. One is outputed by matx or matc, and the other is outputed by the Run-Time libraries. The style of error message from matx or matc is like

```
(1) "Filename",   LineNumber, VariableName, Reason
(2)             FunctionName, VariableName, Reason
```

and the style of error message from the Run-Time libraries is like

```
FunctionName: [Reason] in VariableName
```

9.10 Bugs


```
List OdeHybrid(Real t0, Real t1, Real dt, Matrix x,
          void dx(), void input(), Real h, Real dtsav)
     To solve ordinary differential equations according to
     4th order Runge-Kutta 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} = OdeHybrid();

List OdeHybridAuto(Real t0, Real t1, Real dt, Matrix x,
                  void dx())
                  : eps=1.0E-6, dtsav=1.0E-3,
                    dtmin=EPS, dtmax=abs(t1-t0)/5
List OdeHybridAuto(Real t0, Real t1, Real dt, Matrix x,
                   void dx(), void input())
                  : eps=1.0E-6, dtsav=1.0E-3
                    dtmin=EPS, dtmax=abs(t1-t0)/5
List OdeHybridAuto(Real t0, Real t1, Real dt, Matrix x,
                   void dx(), void input(), Real eps)
              : dtsav=1.0E-3, dtmin=EPS, dtmax=abs(t1-t0)/5
List OdeHybridAuto(Real t0, Real t1, Real dt, Matrix x,
             void dx(), void input(), Real eps, Real dtsav)
                  : dtmin=EPS, dtmax=abs(t1-t0)/5
List OdeHybridAuto(Real t0, Real t1, Real dt, Matrix x,
                   void dx(), void input(), Real eps,
                   Real dtsav, Real dtmin)
                  : dtmax=abs(t1-t0)/5
List OdeHybridAuto(Real t0, Real t1, Real dt, Matrix x,
                   void dx(), void input(), Real eps,
                   Real dtsav, Real dtmin, Real dtmax)
     To solve ordinary differential equations according to
     4th order Runge-Kutta methods automatically changing
     the step to guarantee the prespecified error between
     [t0 t1], where the external input is constant
     for the prespecified sampling interval.
     And the minimum and maximum step are dtmin and dtmax.
     {T,X,U} = OdeHybridAuto();
```

9.4 “ans”: answer when expression is not assigned

When an expression is not assigned to anything else, its value is assigned to the variable “ans”. For example

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

9.5 Complex-Valued Expressions

Complex valued expressions are allowed in all operations in MATX . They are entered as a couple of real valued expressions separated by a comma “,” inside parentheses, “(” and “)”. For example, entering the statement

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

results in

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

There are at least two convenient ways to enter complex matrices. They are illustrated by the statements

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

and

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

which produce the same result. The former statement uses a pair of real matrices as the real-part and the imaginary-part of the matrix. The latter


```
                          (diagonal matrix)
                       vec: Generalized eigenvectors

List svd(Matrix M1);   Singular Value Decomposition
                       {U, D, V} = svd(A);
                       U * D * V# = A;
                       U: Left Singular vectors
                       V: Right Singular vectors
                       D: Singular Values
                          (same size as A)
List lu(Matrix M1);    LU Decomposition
                       {L, U, P} = lu(A);
                       L * U = P * A;
                       L: Lower triangular matrix,
                       U: Upper triangular matrix,
                       P: Permutation matrix
List hess(Matrix M1);  Hessenberg Decomposition
                       {Q, H} = hess(A);
                       Q# * A * Q = H;
                       Q: Orthogonal or unitary matrix
                       H: Hessenberg matrix
List schur(Matrix M1); Schur Decomposition
                       {U, T} = schur(A);
                       U# * A * U = T;
                       U: Orthogonal or unitary matrix
                       T: Pseudo-upper triangular or
                          upper triangular matrix


List qr(Matrix M1);    QR decomposition
                       {Q, R} = qr(A);
                       Q * R = A;
                       Q: Orthogonal or unitary matrix
                       R: Upper triangular matrix
List qr_p(Matrix M1);  QR decomposition (with permutation)
                       {Q, R, P} = qr_p(A);
                       Q * R = A * P;
                       Q: Orthogonal or unitary matrix
```

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

is used to convert an integer to a string.
MATX provides the following class conversion functions.

```
s = String(Integer)           Integer to string.
s = String(Real Number)       Real number to string.

i1 = Integer(String)          String to integer.
i1 = Integer(Real Number)     Integer nearest to real number.
i1 = Integer(Real Matrix)     Integer part of (1,1).
i1 = Integer(Complex Matrix)  Integer part of Re((1,1)).
i1 = Integer(Real Array)      Integer part of (1,1).
i1 = Integer(Complex Array)   Integer part of Re((1,1)).

r1 = Real(String)             String to real number.
r1 = Real(Real Matrix)        (1,1)-element.
r1 = Real(Complex Matrix)     Real-part of (1,1).
r1 = Real(Real Array)         (1,1)-element.
r1 = Real(Complex Array)      Real-part of (1,1).
r1 = Real(Complex Number)     Real part.

c1 = Complex(Real Matrix)     Re(c1) = (1,1).
c1 = Complex(Complex Matrix)  (1,1).
c1 = Complex(Real Array)      Re(c1) = (1,1).
c1 = Complex(Complex Array)   (1,1).
c1 = Complex(Real Number)     Complex number.

P1 = Polynomial(Real Number)   Coefficients are identcal
                               with the elements of the row
                               row vector in ascending order.
P1 = Polynomial(Real Number)     Constant real polynomial.
CP1 = Polynomial(Complex Number) Constant complex polynomial.
```



```
                             Transforms
                             n = 2^m

Array sgn(Array A1)          Element-by-element sgn()
Array abs(Array A1);         Element-by-element abs()
Array arg(Array A1);         Element-by-element arg()
Array exp(Array A1);         Element-by-element exp()
Array sqrt(Array A1);        Element-by-element sqrt()
Array log(Array A1);         Element-by-element log()
Array log10(Array A1);       Element-by-element log10()
Array conj(Array A1);        Element-by-element conj()
Array sin(Array A1);         Element-by-element sin()
Array asin(Array A1);        Element-by-element asin()
Array sinh(Array A1);        Element-by-element sinh()
Array asinh(Array A1);       Element-by-element asinh()
Array cos(Array A1);         Element-by-element cos()
Array acos(Array A1);        Element-by-element acos()
Array cosh(Array A1);        Element-by-element cosh()
Array acosh(Array A1);       Element-by-element cosh()
Array tan(Array A1);         Element-by-element tan()
Array atan(Array A1);        Element-by-element atan()
Array tanh(Array A1);        Element-by-element tanh()
Array atanh(Array A1);       Element-by-element atanh()
Array ceil(Array A1);        Element-by-element ceil()
Array floor(Array A1);       Element-by-element floor()
Array fix(Array A1);         Element-by-element fix()
Array round(Array A1);       Element-by-element round()
```

8.2 Loops — While and For

In

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

the `expr` is evaluated. If it is non-zero, `statement` is executed and `expr` is re-evaluated. This cycle continues until `expr` becomes zero, at which point execution resumes after `statement`. The `for` statement

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

is equivalent to the following.

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

The three elements in `for` statement are expressions syntacticly. In general `expr1` and `expr3` are assignment or function-call, and `expr2` is a relational expression. All three elements may be omitted, but semicolons are not. If the checking part (`expr2`) is omitted, it is treated as true forever. Therefore

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

is a “infinite” loop, and the loop exits are controlled by other methods (such as `break` and `return`).


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

16.3 Append a New Data

The function “`appendlist()`” is used in order to append a new data to the list.

```
x = {sqrt(3.0), ((1.0+2.0)*4.0,:)};
appendlist(x, "Hello");
/* x --> {1.7320508, (12, 0), "Hello"} */
```

16.4 Attachment and Extraction

Long lists can be produced from several short lists. For example, we can make a long list `z` from two short lists `x` and `y` as follows.

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

Any part of a list can be extracted by specifying the interval as two indices. For example,

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

extracts the first 2 elements from `x`.

7.7 Return Statement

The `return` statement is the mechanism for returning a value from the called function to its caller. Any expression can follow `return`. The calling function is free to ignore the returned value if it wishes. There must be one expression after `return`; in case of value-return-functions. Furthermore, there must be no expression after `return`; in case of `void` function.

```
return;       // void function
return expr;  // Otherwise
```

7.8 Reference to Functions

In MaTX a function is not a variable, but it is possible to pass its reference to functions. The reference to a function behaves like its name. However there is no description corresponding to a pointer of function in C.

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

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

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

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



```
Func void main()
{
  Matrix A, B, Q, R, P;  /* Declaration of variables */
  Matrix CRiccati();     /* Declaration of function  */

  read A, B, Q, R;            /* Read matrices   */
  P = CRiccati(A, B, Q, R); /* Call CRiccati() */
  print P;                    /* Print out P     */
}

Func Matrix CRiccati(A, B, Q, R)
  Matrix A, B, Q, R;
{
  Matrix H, U, V, vec, P;

  H = [[ A  , -B*R~*B#]   /* Make a Hamilton matrix */
       [-Q  , -A#     ]];

  vec = eigvec(H);  /* Eigenvectors of H.         */
  U = vec(0, 1, A); /* Extract the eigenvectors   */
  V = vec(1, 1, A); /* correspoding to the stable */
                    /* eigenvalues.               */
  if (isreal(A) && isreal(B) &&
      isreal(Q) && isreal(R)) {
      P = Re(V*U~);
  } else {
      P = V*U~;
  }

  return P;
}
```

List 1: The Program of Continuous-Time Riccati Equation

7.1 Declaration of Functions

After the declaration of a function, it is called wherever. The name of functions is a sequence of letters and digits; the first character must be a letter. The underscore "_" counts as a letter. Upper and lower case letters are different. Some identifiers are reserved for use as keywords, and may not be used otherwise.

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

7.2 Definition of Functions

MATX functions are almost same as K&R C language functions except that it starts with a functions descriptor `Func`. A class name follows `Func` and `void` is used for the functions which don't return any values. Each statement in function is separated by semi colons. There is at least one `return` statement in a function which is not a `void` function.


```
Func void main()
{
  Matrix A, B, C, G;    /* Declaration of variables. */
  Matrix Faddeev();

  read A, B, C;         /* Read matrices  */
  G = Faddeev(A, B, C); /* Call Faddeev() */
  print G;              /* Print out G    */
}

Func Matrix Faddeev(A, B, C)
  Matrix A, B, C;
{
  Matrix Gamma, N;
  Polynomial s, ch;
  Integer i, n;

  s = $;         /* Polynomial variable */
  n = Rows(A);   /* System order        */

  Gamma = Z(n,A);
  ch = Polynomial(Z(1,n+1)); /* Characteristic
                                polynomial        */
  ch(n) = 1.0;               /* Monic polynomial  */

  Gamma(n-1,A) = I(n);
  ch(n-1) = - trace(A);

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

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

  return N / ch;  /* Return the transfer function. */
}
```

List 3: The Program of the Transfer Function

6.1 Declaration of Variables

A declaration specifies a type, and is followed by a list of one or more variables of that type. Variables can be distributed among declarations in any fashion. The name of variables is a sequence of letters and digits; the first character must be a letter. The underscore "_" counts as a letter. Upper and lower case letters are different. Some identifiers are reserved for use as keywords, and may not be used otherwise. Variables may not be initialized in their declaration at the moment.

```
Integer    i1, i2, i3;   // Integer
Real       r1, r2, r3;   // Real Number
Complex    c1, c2, c3;   // Complex Number
String     S1, S2, S3;   // String
Polynomial P1, P2, P3;   // Polynomial
Rational   R1, R2, R3;   // Rational Function
Matrix     M1, M2, M3;   // Matrix
Array      A1, A2, A3;   // Array
List       L1, L2, L3;   // List
```

6.2 Scope Rule

The functions and external variables that make up a MaTX program need not all be compiled at the same time; the source text of the program may be kept in several files, and previously compiled routines may be loaded from libraries.

The *scope* of a name is the part of the program over which the name is defined. For a local variable declared at the beginning of a function, the scope is the function in which the name is declared, and variables of the same name in different functions are unrelated. The same is true of the arguments of the function.

On the other hand, if a global variable is to be referred to before it is defined, or if it is used in a different source file from the one where the function "`main()`" is defined, then an `extern` declaration is mandatory.

It is important to distinguish between the *declaration* of an global variable and its *definition*. A declaration announces the properties of a variable


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

  x0 = [1 1 1]'; /* Initial state     */
  {TC,XC,UC} = Ode(0.0, 5.0, x0, diff_eqs, link_eqs);

  dt = 0.01;     /* Sampling interval */
  x0 = [1 1 1]'; /* Initial state     */
  {TH,XH,UH} =
     OdeHybrid(0.0, 5.0, dt, x0, diff_eqs,link_eqs);

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

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

  A = [[ 0   1   0]   /* System matrix   */
       [ 0   0   1]
       [-2, -3, -4]];
  b = [0 0 1]';       /* Input matrix    */
  dx = A * x + b * u; /* System equation */
}

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

  f = [-1, -1, -1];   /* State feedback control-law */
  u = f * x;
}
```

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

Once a graph is on the screen, by using the mouse and popup menus, graph may be labeled, titled, or have scale lines drawn. It is also possible to chage the font of the labels and titles, to send a screen dump of the picture in the graph canvas to the printer, which is connected to the network, and to obtain the FIG code of the graph, which can be putted in the TEXtext.

Moreover the graph may be labeled, titled, or have grid lines drawn in:

Table 5.2: Graphic Functions 2

Name	Arguments	Functions
title(text)	text: String	graph title (XPLOT)
xlabel(text)	text: String	x-axis label (XPLOT)
ylabel(text)	text: String	y-axis label (XPLOT)
grid()		grid lines (XPLOT)

5.1 Basic Form

If `Y` is a vector, `plot(Y)` (`gplot(Y)`) produces a linear plot of the elements of `Y` versus the index of the elements of `Y`. For example, to plot the numbers $\{0., .48, .84, 1., .91, .6, .14\}$, enter them into a vector and call the function `plot()` (`gplot()`):

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

This result in a graph on your screen. Notice that the data are auto-scaled and that X- and Y-axes are drawn. Once a graph is on the screen, by using the mouse and popup menus, graph may be labeled, titled, or have scale lines drawn.

17.5.3 Simulation of Dynamic Feedback Control

The following program does the simulation of the closed-loop system with a strictly proper system and a proper controller.

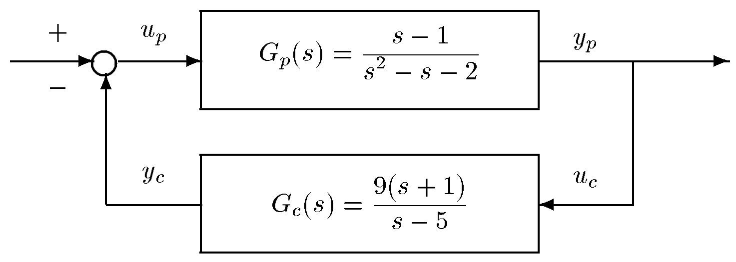

Figure 17.1: Plant and Controller

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

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

Figure 4.3: Matrix Editor (Real Part Matrix)

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

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

Figure 4.4: Matrix Editor (Imaginary Part Matrix)


```
Matrix Ap, Bp, Cp; /* Plant Parameters          */
Matrix Ad, Bd, Cd; /* Uncertainty Parameters    */
Matrix Ac, Bc, Cc; /* Controller-1 Parameters   */
Matrix Af, Bf, Cf; /* Controller-2 Parameters   */
List state_idx;    /* List of indices of states */
List input_idx;    /* List of indices of inputs */

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

  void diff_eqs(), link_eqs();

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

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

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

3.16 Quit

The command `quit` or `exit` exit from the interactive mode of **matx** and **matc**. The command `exit` returns the state.

```
quit;
exit;
exit -1;
```

Appendix A

Data Format

A.1 MAT Data Format

1. The first line begins with a character "`#`", and two integers, i.e. the number of rows and the number of columns, follow.
2. For a complex matrix, the capital "`C`" follows next.
3. Data is separated by blanks, tabs, or newlines.

The MAT data format of a real matrix

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

is

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

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

3.5 List of Variables and Functions (matx only)

The command `who` lists all the variables defined in the environment.

```
who;
```

The command `whos` shows the detail information of all the variables.

```
whos;
```

The command `what` lists all the functions defined in the environment.

```
what;
```

3.5.1 Clear Variables

Variables can be removed from the memory space by `clear` command. Multiple objects can be specified by separating variables with commas.

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

3.6 Existence of Variables (matx only)

In order to check if a variable is defined or not, we can use the function `exist()` with the name of the variable. The function returns 1, 2, or 0, If the variable is defined as a local variable, if the variable is defined only as a global variable, if the variable is not defined.

The name of variable, whose length is `namelen`, follows the 20-byte header, and the last character is `NUL`.
The following data depends on the class of variable.

Integer		Integer (2 bytes or 4 bytes).
Real Number		Real number (4 bytes or 8 bytes).
Complex Number		Two real numbers (4 bytes or 8 bytes).
String		`stringlen`-length string. The last character is `NUL`.
Polynomial	**Real Polynomial**	`degree + 1` real numbers (4 bytes or 8 bytes).
	Complex Polynomial	2×`(degree + 1)` real numbers (4 bytes or 8 bytes).
Rational Function	**Real Rational Function**	`ndegree + 1` real numbers and `ddegree + 1` real numbers (4 bytes or 8 bytes).
	Complex Rational Function	2×`(ndegree + 1)` real numbers and 2×`(ddegree + 1)` real numbers (4 bytes or 8 bytes).
Matrix	**Real Matrix**	`rows`×`cols` real numbers (4 bytes or 8 bytes).
	Complex Matrix	2×`(rows`×`cols)` real numbers (4 bytes or 8 bytes).
	Real Polynomial Matrix	`rows`×`cols` real polynomials: `degree + 1` real numbers (4 bytes or 8 bytes).

2. In order to release the memory area for local variables at the end of the function, write the following declarations and statements corresponding to the classes used in the function at the beginning and at the end of the function, respectively.

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

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

3. Local variables of matrices, polynomials, rational functions, and lists should be declared and initialized at the one time.

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

Appendix B

Operation

The binary arithmetic operators are +, -, *, .*, /, ./, \, .\, ^, .^. The + and - operators have the same precedence, which is lower than the (identical) precedence of *, .*, /, ./, \, .\, ^, and .^, which are in turn lower than that of unary minus -, inverse ~, transpose', and conjugate transpose #. Arithmetic operators group left to right. Table 1 below summarizes the rules for precedence and associativity of all operators.

Table B.1: Precedence and Associativity of Operators

Operator	Associativity
( ) $ [ ]	left to right
++ --	right to left
^ ~ ' # .^ .~	right to left
@ ! -	right to left
* / \ .* ./ .\	left to right
+ -	left to right
< <= > >=	left to right
== !=	left to right
&&	left to right
||	left to right
=	right to left

Operators on the same line have the same precedence; rows are in order

2. If a minus "–" is specified at the end of the command line, matx becomes the interactive mode and shows a prompt "`MaTX (1)`" after it interprets all source files. Thereafter, enter the command "`FunctionName();`" to execute the function.

   [Example] In the interactive mode, enter the command "`main();`".

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

MaTX (1) main();
```

3. Run `matx` without arguments in order to invoke it as the interactive mode. Thereafter, read MM-FILES by the command `load`, and execute it. The default file extension is "`mm`".

   [Example] Add a statement "`main();`" at the end of `sub3.mm`, and read MM-FILES by the command `load`. beginfullbox beginexp*1em

   MaTX (1) load "main" "sub1" "sub2" "sub3" endexp* endfullbox

4. When the declared but not defined function is called, a MM-FILE which has the same name as the function is loaded. It is possible to call a function simply by the command "`FunctionName();`" after declaring it. The name of a function must be identical with the name of the MM-FILE for this facility.

   [Example] Declare the function and call it by its name.

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

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

5. Adding a line "`#! /usr/local/bin/matx`" at the top of MM-FILES and changing the attribute (executable permition) of the file, we can


```
Real-Poly(Mat|Arr)     = Real-Poly * Real(Mat|Arr)
Real-Poly(Mat|Arr)     = Real(Mat|Arr) * Real-Poly
Comp-Poly(Mat|Arr)     = Real-Poly * Complex(Mat|Arr)
Comp-Poly(Mat|Arr)     = Complex(Mat|Arr) * Real-Poly
Comp-Poly(Mat|Arr)     = Comp-Poly * Real(Mat|Arr)
Comp-Poly(Mat|Arr)     = Real(Mat|Arr) * Comp-Poly
Comp-Poly(Mat|Arr)     = Comp-Poly * Complex(Mat|Arr)
Comp-Poly(Mat|Arr)     = Complex(Mat|Arr) * Comp-Poly

Real-Rat-Func(Mat|Arr) = Real-Rat-Func * Real(Mat|Arr)
Real-Rat-Func(Mat|Arr) = Real(Mat|Arr) * Real-Rat-Func
Comp-Rat-Func(Mat|Arr) = Real-Rat-Func * Complex(Mat|Arr)
Comp-Rat-Func(Mat|Arr) = Complex(Mat|Arr) * Real-Rat-Func
Comp-Rat-Func(Mat|Arr) = Comp-Rat-Func * Real(Mat|Arr)
Comp-Rat-Func(Mat|Arr) = Real(Mat|Arr) * Comp-Rat-Func
Comp-Rat-Func(Mat|Arr) = Comp-Rat-Func * Complex(Mat|Arr)
Comp-Rat-Func(Mat|Arr) = Complex(Mat|Arr) * Comp-Rat-Func

Real-Poly(Mat|Arr)     = (Integer|Real Number) * Real-Poly(Mat|Arr)
Real-Poly(Mat|Arr)     = Real-Poly(Mat|Arr) * (Integer|Real Number)
Comp-Poly(Mat|Arr)     = (Integer|Real Number) * Comp-Poly(Mat|Arr)
Comp-Poly(Mat|Arr)     = Comp-Poly(Mat|Arr) * (Integer|Real Number)
Comp-Poly(Mat|Arr)     = Complex Number * Real-Poly(Mat|Arr)
Comp-Poly(Mat|Arr)     = Real-Poly(Mat|Arr) * Complex Number
Comp-Poly(Mat|Arr)     = Complex Number * Comp-Poly(Mat|Arr)
Comp-Poly(Mat|Arr)     = Comp-Poly(Mat|Arr) * Complex Number

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

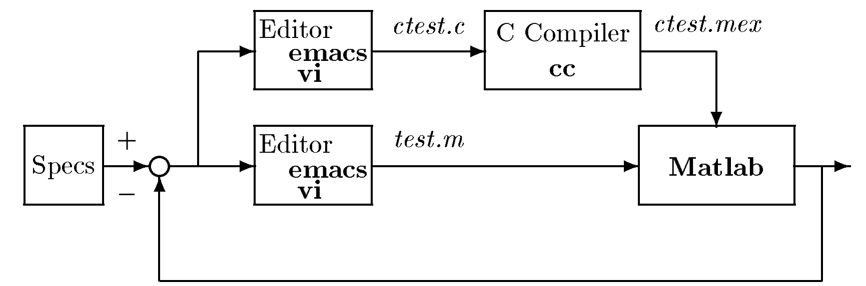

Figure 1.4: Program Developing Process in Matlab

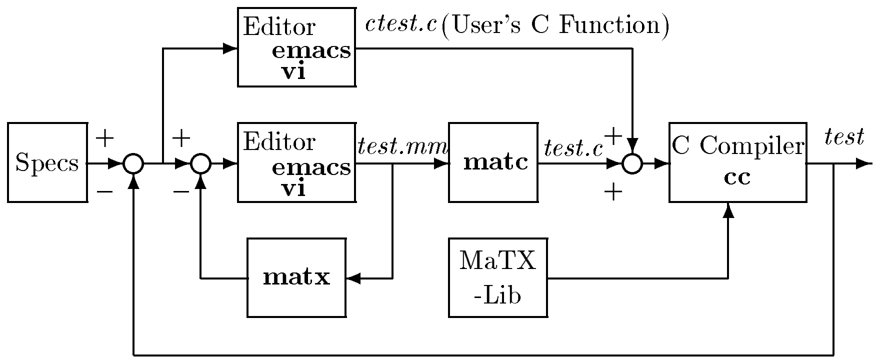

Figure 1.5: Program Developing Process in MATX

satisfied or not.

1.4 MATX Running Environment

At the moment, MATX is checked to run on the following machines under the operating system.

CRAY C916/12256 Cray UNICOS
NEWS5000X NEWS-OS 4.2.1R


```
Complex Matrix          = Complex Array ^ Real Array
Real-PolyMatrix         = Real-PolyArray ^ Real Array
Comp-PolyMatrix         = Comp-PolyArray ^ Real Array
Real-Rat-FuncMatrix     = Real-Rat-FuncArray ^ Real Array
Comp-Rat-FuncMatrix     = Comp-Rat-FuncArray ^ Real Array

Real Matrix             = Real Matrix .^ Integer
Complex Matrix          = Complex Matrix .^ Integer
Real-PolyMatrix         = Real-PolyMatrix .^ Integer
Comp-PolyMatrix         = Comp-PolyMatrix .^ Integer
Real-Rat-FuncMatrix     = Real-Rat-FuncMatrix .^ Integer
Comp-Rat-FuncMatrix     = Comp-Rat-FuncMatrix .^ Integer

Real Matrix             = Real Matrix .^ Real
Complex Matrix          = Complex Matrix .^ Real
Complex Matrix          = Complex Matrix .^ Complex

Real Matrix             = Real Matrix .^ Real Matrix
Complex Matrix          = Complex Matrix .^ Real Matrix
Real-PolyMatrix         = Real-PolyMatrix .^ Real Matrix
Comp-PolyMatrix         = Comp-PolyMatrix .^ Real Matrix
Real-Rat-FuncMatrix     = Real-Rat-FuncMatrix .^ Real Matrix
Comp-Rat-FuncMatrix     = Comp-Rat-FuncMatrix .^ Real Matrix

Real Matrix             = (Integer|Real Number) .^ Real Matrix
Complex Matrix          = Complex Number .^ Real Matrix
Real-PolyMatrix         = Real-Poly .^ Real Matrix
Comp-PolyMatrix         = Comp-Poly .^ Real Matrix
Real-Rat-FuncMatrix     = Real-Rat-Func .^ Real Matrix
Comp-Rat-FuncMatrix     = Comp-Rat-Func .^ Real Matrix
```

B.8 Transpose

Transpose of scalar values (Integer, Real Number, Polynomial, Rational Function) is the same as itself.

1.1 Programming Language MATX

Many of the important ideas of matrix operation of MATX stem from the matrix manipulation language Matlab developed by Cleve Moler [8], [9]. Although it shares several characteristic features with Matlab, MATX is in no sense a dialect of it. Matlab is "typeless" language: the only data type is the Matrix, and operation of other kinds of objects is by special operators or function calls. MATX is equipped to recognize several primitive data type such as Integer, Real-Number, Complex-Number, String, Polynomial, Rational-Function, Matrix, Array, and List. Each class object is stored in an appropriate data structure and treated suitably. Matrix, Array, Polynomial, Rational-Function, and List don't require dimension.

It deals with symbolic formulae as well as numbers. It can solve polynomial equations and evaluate derivatives and integrals symbolically. MATX has inherited the idea of C language such as declaration of variables, control-flow, style of function, and structure of a program [5], [14]. It is written all in C language and the parser has been developed by using the well known parser generator "**yacc**" [13], [15].

MATX provides not only command-line *interpreter* **matx** whose interfaces are similar to the use of Matlab but also *compiler* **matc** which accepts the same syntax as that of matx and outputs portable C language code. We denote the package by MATX and distinguish it from the interpreter (matx). The users can extend the functionality of a program by realizing algorithms as functions in "mm-files" which correspond to "m-files" of Matlab. It is very easy to call your own C functions from MATX functions by linking C language routines to the output of matc. This allows us to utilize huge pre-exist C routines and speed up the rate of computation and promote the efficiency of memory usage.

1.2 Architecture of MATX

The architecture of MATX is shown in Fig. 1.1. At the moment, MATX provides three useful user-interface tools: interpreter "matx", compiler "matc", and matrix-editor "mated". They are built upon MaTX-Lib which is a collection of useful Class libraries such as **Complex-Number**, **Polynomial**, **Rational-Function**, **Matrix**, and **List**[6].

The library set MaTX-Lib can be utilized to make programs in C lan-


```
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:
                              (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
String1 == String2            String1 is same as String2
String1 != String2            String1 is not same as String2
Poly1 == Poly2                Poly1 is equal to Poly2
Poly1 != Poly2                Poly1 is not equal to Poly2
```