MATX: A HIGH-PERFORMANCE INTERACTIVE SOFTWARE PACKAGE FOR SCIENTIFIC AND ENGINEERING COMPUTATION

Masanobu Koga and Katsuhisa Furuta

1 INTRODUCTION

The use of computer–aids for engineering has a history as long as that of computing itself. The earliest attempts at computer–aided control systems design (CACSD) were run in batch mode. The user submitted a Fortran program on cards or paper tape and, provided that the code was syntactically correct, he or she received the results some hours later. Major milestones were the development of specialized subroutine libraries — which encouraged the reuse of tried and tested methods — and packages such as simulation language translators — which accepted problem descriptions and data rather than programs. Some time later, interactive computing began to emerge and the engineer could then communicate on–line and in real–time with a main–frame computer. CACSD packages began to become much more sophisticated; they provided more features and were driven by question and answer dialogue (Munro,1988). An evolutionary step later saw the development of commandline interpreter driven CACSD packages such as CLADP(Maciejowski,1984) and MATLAB (Moler,1980;Moler,1987). We have developed a high-performance interactive software package **MaTX** which provides command-driven *interpreter* **matx** whose interfaces are similar to the use of MATLAB and *compiler* **matc** which has the same syntax as that of matx. The users can extend the functionality of a program by adding interactively functions and algorithms using matx and make independent applications from the same source code using matc.

MATLAB has only one data type, the complex matrix, and other kinds of data such as polynomials and transfer functions have to be coded as matrices. It is really an inefficient coding when we make a program with such limited data structures. The programming in MaTX is object oriented. An object-oriented program is based on objects, which contain both data and the code to manipulate these data(Ravn,1989). A class is the basic data structure definition for an object, along with the functions used for manipulating instances of that object. A call of a member function can depend on the actual type of the object. MaTX has several kinds of objects, such as Matrix, Polynomial, Rational-function, and Complex-number. And structurally different data is stored in separate data structures.

Despite the importance of a good user interface of the acceptance and success of a CACSD package, user interface aspects have often played a secondary role in the design of CACSD

packages. Moreover, even in CACSD software developments where much item and effort was devoted to User Interface (UI) design and implementation, the emphasis was often package flexibility and package extendability rather than user friendliness. MaTX User Interface has been designed and implemented based on the concept of **EDITOR**. A suitable editor is invoked when the user inputs or revises each object such as Matrices, Polynomials, and Rational-functions.

The disposition of the paper is as follows: The architecture of MaTX is given in Section2. The programming in MaTX is described in Section 3. A graphical drawing environment **XPLOT** is presented in Section 4. Section 5 deals with some practical example programs with MaTX. Section 6 concludes with some perspective remarks.

2 THE ARCHITECTURE OF MATX

The architecture of MaTX is shown in Fig. 1. MaTX has a hierarchical structure which consists of three levels, user-interface-level, MaTX-library-level, and object-level. User-interface-level provides three tools, that is, interpreter matx, compiler matc, and Matrix-Editor mated. User-interface-level is built upon MaTX-lib which is the library set of **Matrix**, **Polynomial**, **Rational-function**, and **Complex-number**.

2.1 matx

matx is an interpreter whose basic data elements are Matrices, Polynomials, Rational-functions, Complex-numbers, Strings, and Real-numbers. Matrices, Polynomials and Rational-functions data don't require dimensioning. This allows the users to solve many numerical problems in a fraction of the time it would take to write a program in a language such as Fortran, Basic, or C. Furthermore, problem solutions are expressed in matx almost exactly as they are written mathematically. Matrices and Polynomials can be introduced into matx in easy-to-use way by using Matrix-editor **mated** and Polynomial-editor **polyed**.

2.2 matc

matc is a compiler which recognizes the same source code as matx interprets, and outputs portable C-programs with matc code as comments. The users can make executable applications using these C-programs. All the C-functions, supported by the C-compilers the users use, are available in matc code. Furthermore, once an expression is described with some variables, this expression is reusable with other parameters just like a function because matc holds not only its value but also its definition in the effective scope.

2.3 mated

Figure 2 shows screen-dump of the Matrix editor mated. mated has the following functions.

- Handles any size of Matrices.
- Changes size of Matrices.

- Automatically scrolls if necessary.
- Transposes Matrices if necessary.
- Reads and Writes Matrices from and to files.
- Cut and paste.
- Cursor moves just like VI or Emacs editors.

2.4 MaTX-lib

MaTX-lib is the library set of **Matrix**, **Polynomial**, **Rational-function**, and **Complex-number**. Some objects have an attribute of real and complex. Of course, care need not be taken to distinguish the attribute in programming.

2.4.1 Matrix.

Matlib has higher-level library and lower-level library. Lower-level library provides fast access to the objects, but does little to make programming easier. It handles the operation of the members of the objects and includes some optimizations that encourage efficient memory usage.

Higher-level library is built upon lower-level library. The purpose of higher-level library is to provide an object-oriented layer that supports the user-interface abstraction. It handles the memory allocation of the objects and examines some attributes of the objects and calls suitable lower-level functions.

[Attributes of Matrix]
Matrix class has the following attributes.

Name	:	Any number of letters and digits.
Size	:	Number of columns and Number of Rows.
Type	:	Temporary instance, Permanent instance.
Class	:	Complex Matrix, Real Matrix, Rational Matrix.
Elem	:	Element of the Matrix.

The following list captures the basic functions of **Matrix** class.

- Any size of instances are available.
- Memory allocation is done automatically.
- Three type of objects are treated equally.
- Instances can be referred by their names.

2.4.2 Polynomial.

MaTX handles polynomials of one variable, with real and complex coefficients. The coefficients of a polynomial area represented in MaTX as a row vector containing the coefficients ordered by ascending powers. **Polylib** handles the memory allocation of the objects and examines some attributes of the objects and treats it suitably.

[Attributes of Polynomial]
Polynomial class has the following attributes.

Name	:	Any number of letters and digits.
Type	:	Temporary instance, Permanent instance.
Class	:	Complex Polynomial, Real Polynomial.
Coef	:	Coefficient of the Polynomial.
Order	:	Any order of polynomials are handled.

The following list captures the basic functions of **Polynomial** class.

- Any order of instances are available.
- Memory allocation is done automatically.
- Real and complex instances are treated equally.
- Instances can be referred by their names.

2.4.3 Rational-function.

MaTX handles rational-functions of one variable, with real and complex coefficients. A Rational-function is represented in MaTX as a numerator polynomial and a denominator polynomial. **Ratlib** handles the memory allocation of the objects and examines some attributes of the objects and treats it suitably.

[Attributes of Rational-function]
Rational-function class has the following attributes.

Name	:	Any number of letters and digits.
Type	:	Temporary instance, Permanent instance.
Class	:	Complex Rational, Real Rational.
Nume	:	Numerator of the rational-function.
Deno	:	Denominator of the rational-function.
NOrder	:	Any order of polynomial.
DOrder	:	Any order of polynomial.

The following list captures the basic functions of **Rational-function** class.

- Any order of instances are available.
- Memory allocation is done automatically.
- Real and complex instances are treated equally.
- Instances can be referred by their names.

3 THE PROGRAMMING IN MATX

The MaTX language contains no dimension statements. Type declarations are required only in matc for compilation. Storage is allocated automatically, up to the amount available on any particular computers.

3.1 Entering Simple Matrices

- Entered by an explicit list of elements,
- Generated by built-in statements and functions,
- Created by Matrix Editor mated,
- Loaded from external data files.

The easiest method of entering small matrices is to use explicit lists. The explicit list of elements separated by blanks, tabs, commas, or carriage returns, is surrounded by brackets, "[" and "]". Brackets are also used for indicating the begins and ends of the rows.

For example, entering the statement

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

or

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

results in the output

```
=== [A] : (  3,  3) ===
           (  1)           (  2)           (  3)
(  1)  1.00000000E+00  2.00000000E+00  3.00000000E+00
(  2)  4.00000000E+00  5.00000000E+00  6.00000000E+00
(  3)  7.00000000E+00  8.00000000E+00  9.00000000E+00
```

The matrix **A** is saved for later use.

3.2 Entering Polynomials

Polynomials are entered by using a function **poly()**. For example,

```
A = poly("4 * s^2 + 5 * s + 6");
```

or

```
A = poly("4s^2+5s+6");
```

results in the output

```
A = 4 s^2 + 5 s + 6
```

3.3 Entering Strings

Strings are entered by surrounding strings by double quotations " and ". For example,

```
s = "Hello world."
```

3.4 Matrix Elements

Matrix elements can be any MaTX expressions; for example,

```
x = [-1.3 , sqrt(3), (1+2+3)*4/5 ];
```

results in

```
=== [x] : (  1,  3) ===
           (  1)           (  2)           (  3)
(  1) -1.30000000E+00  1.73205081E+00  4.80000000E+00
```

Individual matrix elements can be referenced with indices inside parentheses, "(" and ")". Big matrices can be constructed using little matrices as elements. For example, we could attach another row to our matrix **A** with

```
r = [10 11 12];
A = [[A][r]];
```

which result in

```
=== [A] : (  4,  3) ===
           (  1)           (  2)           (  3)
(  1)  1.00000000E+00  2.00000000E+00  3.00000000E+00
(  2)  4.00000000E+00  5.00000000E+00  6.00000000E+00
(  3)  7.00000000E+00  8.00000000E+00  9.00000000E+00
(  4)  1.00000000E+01  1.10000000E+01  1.20000000E+01
```

Any block matrices can be extracted from big matrices specifying left-top and right-bottom locations. For example,

```
A = A(1,1,3,3);
```

takes the 3×3 matrix to give us back the original **A**.

3.5 Statements and Variables

MaTX statements are frequently of the form

variable = *expression*;

Expressions are composed from operators, other special characters, functions, and variable names. Variable and function names are formed by a letter, followed by any number of letters and digits (or underscores). All the characters of a name are remembered. MaTX is case-sensitive; it normally distinguishes between upper- and lower- case letters.

3.6 Complex Numbers and Matrices

Complex numbers are allowed in all operations and functions in MaTX. Complex numbers are entered as a couple of real numbers surrounded by angle brackets "<"and ">". For example, entering the statement

```
a = <3,4>;
```

results in the output

```
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.

3.7 Matrix Operations

Matrix operations are fundamental to MaTX; wherever possible they are indicated the way they would be in a textbook or on paper, subject only to the character set limitations of the computer. The operation of real matrices and complex matrices results in a complex matrix.

[Inverse] The special character tilde "$\sim$" denotes the inverse of a matrix.

[Transpose] The special character apostrophe "'" denotes the transpose of a matrix.

[Conjugate Transpose] The special character sharp "#" denotes the complex conjugate transpose of a complex matrix, and the transpose of real matrix.

[Addition and subtraction] Addition and subtraction of matrices are denoted by "+" and "−". The operations are defined whenever the matrices have the same dimensions.

[Matrix multiplication] Multiplication of matrices is denoted by "∗". The operation is defined whenever the inner dimensions of the two operands are the same.

[Matrix division] There are two matrix division symbols in MaTX, "/" and "\" just like MATLAB. If **A** is a nonsingular square matrix, then "**A****B**" and "**B**/**A**" correspond formally to left and right multiplication of **B** by the inverse of **A**.

[Matrix powers] The expression "**A**^p" raises **A** to the p-th power and is defined if **A** is a square matrix and p is an integer greater than zero.

[Determinant and Absolute value] Determinants of matrices and absolute values of complex and real numbers are represented by surrounding them by "|" and "|".

[Reuse of expressions] Once an expression is described with some variables, this expression is reusable with other parameters just like a function. For example,

```
x = 1.0;
A = [x 2.0];
```

results in

```
=== [A] : (  1,  2) ===
           (  1)           (  2)
(  1)  1.00000000E+00  2.00000000E+00
```

and

```
x = 2.0;
B = @A;
```

results in

```
=== [B] : (  1,  2) ===
           (  1)           (  2)
(  1)  2.00000000E+00  2.00000000E+00
```

3.8 Output to Standard-output

All objects can be wrote to standard-output by **print** command. Multiple objects can be specified separating by commas. For example,

```
print a, A, 1, <2,3>;
```

3.9 Input from Standard-input

All objects can be read from standard-input by **read** command. Multiple objects can be specified separating by commas. If Matrices and Polynomials are specified, the corresponding editor is invoked to read them. For example,

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

3.10 File Operation

To input and output matrices from and to files, **read** and **print** commands are used with **redirect** symbol < and > just like UNIX shell. For example,

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

3.11 Control Flow

MaTX has control flow statements just like C language such as **for** statement, **while** statement, and **if then else** statement.

4 XPLOT

Scientific and engineering data, which was created in MaTX, are examined graphically in **XPLOT**. XPLOT is a very useful graphical drawing environment which runs under **X Window System**. X Window System, which enables a wide variety of bitmapped graphics devices to support window, icon, mouse, and pointer (WIMP) interfaces across a network, was developed as part of Project Athena (Balkovich,1985) In the X Window System, applications can be built upon useful tools called X Toolkit (Nye,1990). It consists of the following:

- A set of intrinsic mechanisms for building widgets.
- A consistent interface set (widget set) for programming.

The XPLOT graphical drawing environment allows the user to perform all controls-related operations in a very consistent manner over mouse-operated menus and forms. A menu hierarchy is used to group related operations together into domains. At the bottom of the menu tree, selection and action forms are used to give a highly interactive execution of most operations.

XPLOT is built upon some Widgets classes:

[Canvas Widget Class]
The size of this widget class can be changed, and it redraws the objects automatically whenever hidden parts appear.

[Data Information Widget Class]
This class displays various information and manages all the data in XPLOT. The user can select the data using a scroll-bar and change the state of drawing with a toggle-button.

Any data in XPLOT are created by putting together some 1-dimensional data as the user likes. Figure 3 and Fig. 4 show screen-dumps of the XPLOT graphical drawing environment. The 1D-Data Information Widget and 2D-Data Information Widget are displayed at the right side on the screen. When a toggle-button in a Data Information Widget is clicked upon, the corresponding line is drawn. The 1D-Canvas Widget and 2D-Canvas Widget are displayed at the left side on the screen. The data of 2D-Canvas Widget was made by putting together two 1-dimensional data of 1D-Canvas Widget.

5 EXAMPLE

The MaTX Package has been used to create an illustration of the capabilities and features described above. The demonstration scenario involves obtaining the solutions of the Riccati equations of continuous and discreet linear systems.

5.1 Discreet Riccati Equation

The solution of the Discreet Riccati Equation of a linear system is obtained as in List 1. The following statements are explanations of List 1.

- On the 1st and 16th line, declaration of functions.

- On the 3rd and 19th line, declaration of variables.
- On the 6th to 10th, 21st, and 22nd line, entering matrices.
- On the 12th line, call function DRiccati().
- On the 13th line, print matrix P.
- On the 25th to 27th line, calculation of matrices.
- On the 29th line, examine the value of Frobenius norm.

5.2 Continuous Riccati Equation

The solution of the Continuous Riccati Equation of a linear system is obtained as in List 2. The following statements are explanations of List 2.

- On the 1st and 11th line, declaration of functions.
- On the 3rd, 14th, and 15th line, declaration of variables.
- On the 6th line, reading matrices from standard input.
- On the 7th line, call function CRiccati().
- On the 8th line, print matrix P.
- On the 19th line, making a Hamilton matrix.
- On the 21th line, calculation of eigenvectors.
- On the 27th line, **Re** denotes Real part of the matrix.

6 CONCLUSIONS

We have developed a high-performance interactive software package **MaTX** which provides command-driven *interpreter* **matx** whose interfaces are similar to the use of MATLAB and *compiler* **matc** which has the same syntax as that of matx. The users can extend the functionality of a program by adding interactively functions and algorithms using matx and make independent applications from the same source code using matc. MaTX User Interface has been designed and implemented based on the concept of EDITOR. A suitable editor is invoked when the users input or revise each object such as Matrices, Polynomials, and Rational-functions.

Scientific and engineering data, which was created in MaTX, are examined graphically in **XPLOT**. XPLOT is a very useful graphical drawing environment which runs under X Window System. The XPLOT graphical drawing environment allows the user to perform all controls-related operations in a very consistent manner over mouse-operated menus and forms.

Further work is required to investigate and implement a higher performance for scientific and engineering computation. Work is also needed to explore and demonstrate its benefits for CACSD. The database management facilities will be needed to include version control, concurrency, and consistent data checking.

REFERENCES

Balkovich,E. ,S.Lerman, and R.P.Parmalee (1985). Computing in higher education: The athena experience. *CACM, 28*,1214–1224

Maciejowski,J.M. (1984).Data structure for control system design. *EUROCON'84, Brighton, U.K.*

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

Moler,C. (1980). *MATLAB – User's Guide.* Alberquerque, USA

Munro,N. (1988). Ecstasy – a control system cad environment. *Preprints of 4th IFAC Symposium on Computer Aided Design in Control Systems – CACSD '88, Beijing, P.R. China*, 76–80

Nye,A. ,and T.O'Reilly (1990). *X Toolkit Intrinsics Programming Manual, 4.* O'Reilly & Associates, Inc.

Ravn,O. (1989). On user–friendly interface construction for cacsd packages. *CACSD '89,*, 35–40

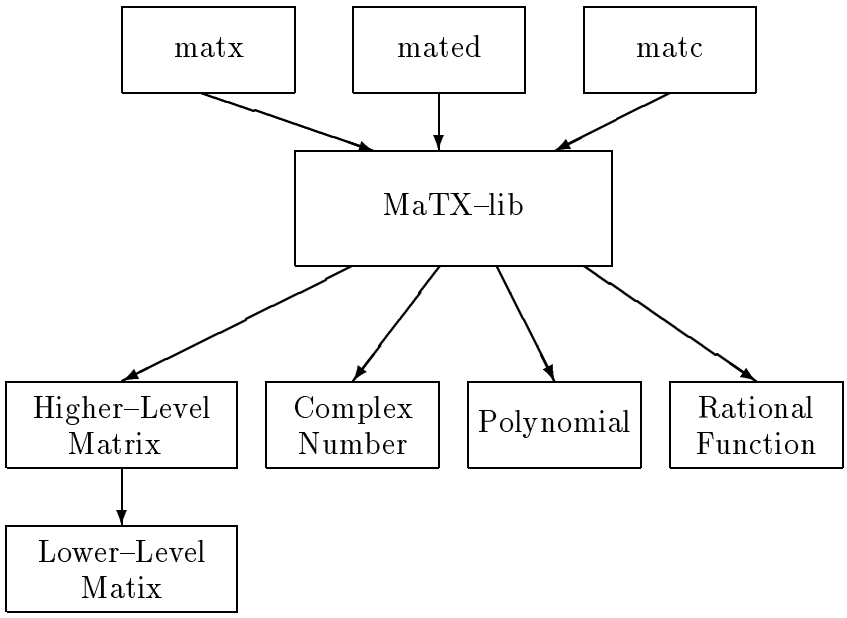

Fig. 1. The architecture of MaTX

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

           (  3)           (  4)           (  5)           (  6)
( 12)  -1.98208202E-01  8.37350158E-02 -1.98208202E-01  8.37350158E-02
( 13)  -1.97804416E-01  8.57034700E-02 -1.97804416E-01  8.57034700E-02
( 14)  -1.97400631E-01  8.76719243E-02 -1.97400631E-01  8.76719243E-02
( 15)  -1.96996845E-01  8.96403785E-02 -1.96996845E-01  8.96403785E-02
( 16)  -1.96593060E-01  9.16088328E-02 -1.96593060E-01  9.16088328E-02
( 17)  -1.96189274E-01  9.35772871E-02 -1.96189274E-01  9.35772871E-02
( 18)  -1.95785489E-01  9.55457413E-02 -1.95785489E-01  9.55457413E-02
( 19)  -1.95381703E-01  9.75141956E-02 -1.95381703E-01  9.75141956E-02
( 20)  -1.94977918E-01  9.94826498E-02 -1.94977918E-01  9.94826498E-02
( 21)  -1.94574132E-01  1.01451104E-01 -1.94574132E-01  1.01451104E-01
( 22)  -2.08454259E-01  8.37854890E-02 -2.08454259E-01  8.37854890E-02
( 23)  -2.08050473E-01  8.57539432E-02 -2.08050473E-01  8.57539432E-02
( 24)  -2.07646688E-01  8.77223975E-02 -2.07646688E-01  8.77223975E-02
( 25)  -2.07242902E-01  8.96908517E-02 -2.07242902E-01  8.96908517E-02
( 26)  -2.06839117E-01  9.16593060E-02 -2.06839117E-01  9.16593060E-02
( 27)  -2.06435331E-01  9.36277603E-02 -2.06435331E-01  9.36277603E-02
( 28)  -2.06031546E-01  9.55962145E-02 -2.06031546E-01  9.55962145E-02
( 29)  -2.05627760E-01  9.75646688E-02 -2.05627760E-01  9.75646688E-02
( 30)  -2.05223975E-01  9.95331230E-02 -2.05223975E-01  9.95331230E-02
( 31)  -2.04820189E-01  1.01501577E-01 -2.04820189E-01  1.01501577E-01
( 32)  -2.04416404E-01  1.03470032E-01 -2.04416404E-01  1.03470032E-01
```

Fig. 2. Matrix Editor — mated

Fig. 3. 1D–Canvas Widget and 2D–Canvas Widget

Fig. 4. SemiLog–Canvas Widget

```
01:func void main()
02:{
03:  Matrix A, B, C, Q, R, P, PP;
04:  Matrix DRiccati();
05:
06:  A = [[0 1][-3, 4]];
07:  B = [0 1]';
08:  C = [-2, 1];
09:  Q = C'*C;
10:  R = I(1);
11:
12:  P = DRiccati(A, B, Q, R);
13:  print P;
14:}
15:
16:func Matrix DRiccati(A, B, Q, R)
17:  Matrix A, B, Q, R;
18:{
19:  Matrix P, PP;
20:
21:  P  = Z(2);
22:  PP = Z(2);
23:
24:  while (1) {
25:    P = Q + A'*P*A
26:        - A'*P*B*(R + B'*P*B)~ * B'*P*A;
27:    P = (P' + P)/2.0;
28:
29:    if (||P - PP|| < 10E-7) {
30:      break;
31:    }
32:    PP = P;
33:  }
34:
35:  return (P);
36:}
```

List 1 The programm of Discreet Riccati Equation

```
01:func void main()
02:{
03:  Matrix A, B, Q, R, P;
04:  Matrix CRiccati();
05:
06:  read A, B, Q, R;
07:  P = CRiccati(A, B, Q, R);
08:  print P
09:}
10:
11:func Matrix CRiccati(A, B, Q, R)
12:  Matrix A, B, Q, R;
13:{
14:  Matrix H, U, V, vec, P;
15:  int size;
16:
17:  size = RSize(A);
18:
19:  H = [[ A  , -B*R~*B#]
20:       [-Q  , -A#     ]];
21:  vec = eigvec(H);
22:  U = vec(     1, size+1,   size, 2*size);
23:  V = vec(size+1, size+1, 2*size, 2*size);
24:
25:  if (isreal(A) && isreal(B)
26:              && isreal(Q) && isreal(R)) {
27:    P = Re(V*U~);
28:  }
29:  else {
30:    P = V*U~;
31:  }
32:
33:  return (P);
34:}
```

List 2 The programm of Continous Riccati Equation