A Compiler of MATLAB to MATX:
Compiling and Linking of m-files to an Executable Program

Masanobu Koga, **Mitsuji Sampei**, and **Katsuhisa Furuta**

Department of Control and System Engineering
Tokyo Institute of Technology
2-12-1, Oh-okayama, Meguro-ku, Tokyo 152
JAPAN

koga@ctrl.titech.ac.jp, sampei@ctrl.titech.ac.jp, and furuta@ctrl.titech.ac.jp

Abstract

In this paper we present a compiler of MATLAB to MaTX, a tool to automatically create mm-files of MaTX for any MATLAB m-files for using them in the interpreter (matx) and linking them to an executable program. The translator enables the user of MATLAB to benefit from the interactivity of the interpreter environment and the performance of the compiled and linked executable program.

Keyowrds: Environments for CACSD, compiler, interpreter, translator, MATLAB

1 Introduction

The appearance of the interactive linear-algebra workbench, MATLAB [5] was a factor decisively influencing the field of linear computer aided control system design (CACSD) tools. The powerful expressiveness of the MATLAB command interface allowing, for example, state-space control algorithms to be expressed through their matrix-operation primitives, and the reliability of the underlying algorithms from LINPACK and EISPACK, made it well suited for control purposes. Soon after its introduction, a number of control engineers became convinced of the advantages of general-purpose mathematical programs like MATLAB compared to special purpose control packages. Packages such as MATRIX-x, CTRL-C and MATLAB set a de facto standard for CACSD user interface design through their so called "matrix environments." [6] It has found widespread acceptance in application areas in which linear algebra plays an important role, such as signal processing, process identification and control system design. For these applications, special toolboxes are available, which are collections of small procedures (m-files) written in the MATLAB language. The "toolbox" concept, which allows the user to add arbitrary numerical algorithms to the package, started the trend toward open CACSD environments. [1]

However there are at least two areas where we should expect improvements. One is concerned with limited data type. MATLAB has only one data type (complex matrix), thus the other kinds of data are supposed to be coded by using matrices. Such limited data structure doesn't allow a natural representation of mathematical data and makes the problem description hard to understand. Therefore the extensions [6] to the linear matrix environments to handle more data types than the complex matrix (such as polynomial and rational function representations enabling the implementation of additional classes of control algorithms.) is indispensable. The other is concerned with speed of computation. M-file implementation of an algorithm is quite often not efficient enough, since m-files are interpreted rather than compiled.

We have developed a high-performance programming language MATX[2, 3, 4] for general scientific and engineering computation. It has inherited any of important ideas of matrix operation from the "matrix environments" of MATLAB, MATRIX_x, and CTRL-C. It is a type-oriented language and is equipped to recognize several data types such as integer, real number, complex number, string, polynomial, rational function, matrix, array, index, and list. It has inherited many features of C language such as declarations, control-flow, and style of function. The parser was developed by using the

well-known parser generator **yacc**, and all the other parts are written in C language.

MATXprovides 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 codes. The user can extend the functionality of a program by realizing algorithms as functions in "mm-files" which correspond to "m-files" of MATLAB. In the MATXprogram environment, the user ordinarily makes several mm-files by using a favorite editor according to the requirements. The interpreter (matx) enables us to test out the mm-files interactively file-by-file or line-by-line. If the mm-file implementation of an algorithm is not efficient enough, the examined mm-files are compiled into portable C language files with the compiler (matc). Then those C files are compiled and linked up with the run-time library (**MATX-Lib**) [4] with a C compiler to generate the desired executable program. It is possible to call your own C functions from MATXfunctions by linking C language routines to the executable program.

The interpreter (matx) and the compiler (matc) are built upon MaTX-Lib which is a collection of useful class libraries such as complex number, polynomial, rational function, matrix, and list. Most of the primitive matrix manipulation functions provided by MATLAB are included in the library. In order that the user of MATLAB can enjoy the type-oriented MATXenvironment and benefit from the interpreter and compiler facilities, we have developed the translator (**m2mm**) which translates m-files to mm-files. The translator is written in Perl [7] (Practical Extraction and Report Language), which is a language for easily manipulating text, files, and processes, and provides a more concise and readable way to do many jobs that were formerly accomplished by programming in the C language or one of the shells.

The disposition of the paper is as follows: The tasks of the translator is presented in detail in Section 2. Section 3 describes how to translate m-files to mm-files and link them to an executable program, and Section 4 deals with an illustrative example. Finally, in Section 5 we conclude with some perspective remarks.

2 Tasks of the Translator

The design objective was to create a compiler that automatically creates an mm-file from any m-file to be used in MATX. The compiler has to perform the following steps:

1. In order to crate an mm-file the compiler should be aware of the data type, since in the function of the mm-file the data types of variables should be declared for compilation.

2. Analyze the source code of an m-file in order to translate the following expressions and statements of the m-file to that of the mm-file.

   - The style of some data types
   - The name and usage of some builtin functions
   - The style of control flow and conditions
   - The definition of function
   - The style of comments and continuing line

2.1 Data type declaration

There is no data type declaration statements in an m-file, since the interpreter knows everything about all variables at run time. Compilers, however, can't get the information of variables and functions at compile time, unless there are declaration statements.

The compiler searches the data type of a variable by pattern matching which can be done quite well by Perl. The compiler uses the following pattern matching:

Pattern	Expected data type
`var` = constant integer	Integer
`var` == constant integer	Integer
`var` = constant real number	Real
`var` == constant real number	Real
`var` = '.....'	String
`var` = matrix(integer)	Real
`var` = sqrt(–1)	Complex
`var` = matrix(integer:integer)	Matrix
`var` = [`var1 var2 ... varn`]	Matrix
`var` = [`var1; var2; ... varn`]	Matrix
`var` = matrix'	Matrix
`var` = matrix .* matrix	Matrix
`var` = matrix ./ matrix	Matrix
`var` = matrix .\ matrix	Matrix
for `var` = [integer:integer]	Integer
[`var1:var2:var3`]	Integer
`var` = [integer:integer:integer]	Index

The translator utilizes the information about the return values of some functions which often appear in many m-files as follows.

Pattern	Expected data type
var = rank(matrix)	Integer
var = max(size(matrix))	Integer
var = zeros(integer,integer)	Matrix
var = ones(integer,integer)	Matrix
var = eye(integer, integer)	Matrix
var = imag(matrix)	Matrix
var = real(matrix)	Matrix
var = inv(matrix)	Matrix
var = diag(matrix)	Matrix
var = norm(matrix)	Real
var = sum(matrix)	Real
var = length(matrix)	Integer
var = find(condition)	Index

We can simply add the information of a function to the database of the compiler. At the moment the translator can search the data type of almost all variables, especially for m-files of control toolbox of MATLAB over 90 percent data type of variables can be determined. If the data type of some variables can not be determined, the translator lists up those variable names at the beginning of the function. Then the user has to specify the data type of those variables after the compilation. The more information about the functions of m-files are registered in the database, the more accurately the data type of variables can be determined, but the more time it takes for the compilation.

2.2 Expression and statement

2.2.1 Data Types

Matrices

In m-files the easiest method of entering small matrices is to use an explicit list. The explicit list of elements is separated by blanks or commas, is surrounded by brackets, [and], and uses the semicolon ; to indicate the ends of the rows. In mm-files, the easiest method of entering a small matrix is to use an explicit list whose elements are separated by spaces, tabs, carriage returns, or commas, and surround it by square brackets, [and]. Brackets are also used for indicating the beginning and end of the rows. For example,

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

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

is translated into

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

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

Strings

Text strings are entered into MATLAB surrounded by single quotes. In MATXA 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. For example,

```
A = 'Hello'
```

is translated into

```
A = "Hello"
```

2.2.2 Relational and Logical Operators

The relational operators of MATXare >, >=, <, <=, ==, and !=. The difference of the relational operators between MATLAB and MATXis ~= and !=. The logical operators of MATXare &&, ||, and ! which correspond to &, |, and ~ of MATLAB, respectively.

2.2.3 Builtin functions

The names and usage of some functions in m-files are translated into that of mm-files as follows:

m-file	mm-file
B = real(A)	B = Re(A)
B = imag(A)	B = Im(A)
B = eye(A)	B = I(A)
B = zeros(A)	B = Z(A)
B = ones(A)	B = ONE(A)
B = int2str(A)	B = String(A)
B = sqrt(-1)	B = (0,1)
B = sign(A)	B = sgn(A)
B = expm(A)	B = exp(A)
B = logm(A)	B = log(A)
B = inv(A)	B = A
B = size(A)	B = [Cols(A) Rows(A)]
C = polyval(A,B)	C = eval(Polynomial(fliplr(A)), B)
B = roots(A)	B = roots(Polynomial(fliplr(A)))
B = eig(A)	B = eigval(A)

2.2.4 Control Flow

MATXhas control flow statements which are almost same as that of C language. It provides the fundamental flow-control constructions required for well-structured programs: statement grouping, decision making (**if**); looping while the termination test (**while**, **for**) such as **for** statement, **while** statement, and **if-then-else** statement. The **break** statement provides an early exit from **for** and **while**. They cause the innermost enclosing loop to be exited immediately. The **continue** statement

causes the next iteration of the enclosing loop to begin. The condition of control flow is described as an expression inside parentheses. For example,

```
if (number < low)
  msg = 'Not enough input arguments.';
elseif (number > high)
  msg = 'Too many input arguments.';
end

for i=1:n-1
  v = A*v;
  T = [T,v];
end
```

is translated into

```
if (number < low) {
  msg = "Not enough input arguments.";
} else if (number > high) {
  msg = "Too many input arguments.";
}

for (i = 1; i <= n-1; i++) {
  v = A*v;
  T = [T,v];
}
```

2.2.5 Definition of a function

In the m-file of MATLAB, functions have the general form:

```
[a, b, c, ...] = function_name(d, e, f, ...)
```

where the ... denotes ellipsis. The a, b, c, ... are "left-hand-side" arguments and the d, e, f, ... are "right-hand-side" arguments. In other words functions use multiple input arguments and return multiple output arguments. In the syntax of traditional languages such as Fortran, Pascal, and C, functions use multiple input arguments and return only one value. In order to implement the multiple output functions, MATXintroduce the "List Class" which is a collection of elements not necessarily of the same type. Functions return multiple output values as a list which contains those multiple values. For example,

```
function [P,result] = ric(A,R,Q)
```

is translated into

```
Func List ric(A,R,Q)
    Matrix A,R,Q;
{
    Matrix P;
    Integer result;
    ......................
    return {P,result};
}
```

Functions of m-files can take the variable number of arguments and the number of input arguments is set in a variable (**nargin**). In mm-files a function with the variable number of arguments is described by using ... at the end of the argument list and the number of input arguments is set in a variable (**nargs**). For example,

```
function X = lyap(A, B, C)
    if nargin == 2
        C = B;
        B = A';
    end
```

is translated into

```
Func Matrix lyap(A, B, C, ...)
    CLASS A, B, C;
{
    Matrix X;

    if (nargs == 2) {
        C = B;
        B = A';
    }
    ......................
```

2.2.6 Comments and continuing line

MATLAB uses % to introduce comments, and any characters to the end of the line are ignored. In mm-files, any characters between "/*" and "*/" are ignored. In m-files, if the expression is so complicated that the statement will not fit on one line, an ellipsis consisting of three or more periods, ..., followed by the carriage return, can be used to indicate that the statement continues on the next line. In mm-files, an arithmetic or logical operator or a comma "," at the end of a line means that the expression continues to the next line.

```
s = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 - ...
          1/8 + 1/9 - 1/10 + 1/11 -1/12;
```

is translated into

```
s = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 -
        1/8 + 1/9 - 1/10 + 1/11 -1/12;
```

3 Compiling and Linking of m-files to an executable program

In order to use the function **ric()** in (**ric.m**) of MATLAB, the user must run the m-file/mm-file compiler (m2mm) to create the mm-file (**ric.mm**), edit the mm-file (ric.mm) for specifying the data type of arguments of the function as shown in the next section. Then compile the mm-files (ric.mm) and (main.mm) which calls the function ric() with the compiler (matc) to create the C code files (ric.c and main.c), compile the C code files (ric.c and main.c) with a C compiler, and finally link the object files (ric.o, and main.o) and the run-time library

```
%LQR    Linear quadratic regulator design for continuous-time systems.
%  [K,S] = LQR(A,B,Q,R) calculates the optimal feedback gain matrix K
%  such that the feedback law  u = -Kx  minimizes the cost function:

[d,index] = sort(real(d));   % sort on real part of eigenvalues
```

is translated into

```
/*
 * LQR  Linear quadratic regulator design for continuous-time systems.
 * {K,S} = LQR(A,B,Q,R)  calculates the optimal feedback gain matrix K;
 * such that the feedback law  u = -Kx  minimizes the cost function:
 */

{d,index} = sort(Re(d));     /* sort on real part of eigenvalues */
```

(libMaTX.a) together to an executable file (ric). For UNIX users this would involve typing in the following sequence of commands:

```
% m2mm ric.m
% edit ric.mm main.mm
% matc ric.mm main.mm
% matcc ric.c main.c
```

Figure 1 shows the procedure of compiling and linking of m-files to an executable program.

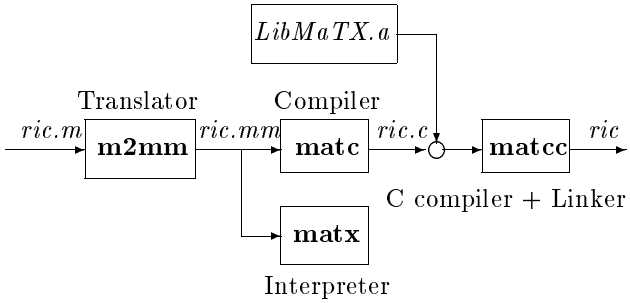

Fig. 1 Compiling and linking of m-files

The example presented above creates an mm-file for a single m-file. It is also possible to create mm-files for an entire m-files, which the user has made to use in MATLAB environment, compile those mm-files with matc to create C code files, and make everything together to libraries such as LibCtrl.a, LibRobust.a, and LibSignal.a as shown in Fig. 2.

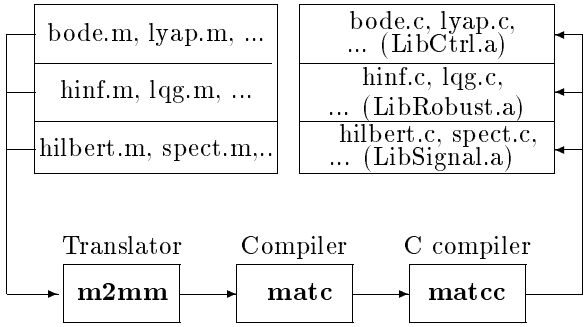

Fig. 2 Making run-time libraries from m-files

4 Example

In this section we illustrate the benefit of the m-file/mm-file compiler by an m-file from the control toolbox of MATLAB. We use the the function **ric()** in (**ric.m**) of MATLAB shown in List 1, the user must run the m-file/mm-file compiler (m2mm) to create the mm-file (**ric.mm**) shown in List 2, edit the mm-file (ric.mm) for specifying the data type of arguments of the function. In this case at the 2nd line:

```
CLASS A,R,Q;  ------->   Matrix A,R,Q;
```

Then compile the files (ric.mm) and (main.mm):

```
Func void main()
{
    Matrix A, Q, R, P;
    Integer result;
    List ric();

    read A, Q, R;
    {P, result} = ric(A, R, Q);
    print P;
}
```

with the compiler (matc) to create the C code files (ric.c and main.c), compile the C code files (ric.c and main.c) with a C compiler, and finally link the object files (ric.o, and main.o) and the run-time library (libMaTX.a) together to an executable file (ric).

5 Concluding Remarks

The m-file/mm-file compiler was presented and its features were described. Its main strength is its simplicity: the user only has to supply the data type of arguments of a function. The m-file/mm-file compiler then automatically generates the mm-file from an m-file. The generated mm-files can be used in the interpreter matx, and compiled with the compiler (matc) and linked with the run-time library to an executable program.

The concepts described here are equally well suited for interfacing MATXwith other matrix manipulation packages like MATRIX_x or CONTROL-C. The flexibility and strength of the m-file/mm-file

compiler is easily seen to be important for designers who want to combine the interactivity of the interpreter environment with the performance of the compiled and linked executable program.

References

[1] H. Anthony Barker, Min Chen, Philip W. Grant, Chris P. Jobling, and Peter Townsend. Open architecture for computer-aided control engineering. *IEEE Control Systems*, 13(2):17–27, 1993.

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

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

[4] Masanobu Koga and Katsuhisa Furuta. Programming language MaTX for scientific and engineering computation. In Derek A. Linkens, editor, *CAD for Control Systems*, chapter 12, pages 287–317. Marcel Dekker, Inc., July 1993.

[5] Cleve Moler. *MATLAB – User's Guide*. Albequerque, USA, 1980.

[6] C. Magnus Rimvall. Computer-aided control system design. *IEEE Control Systems*, 13(2):14–16, 1993.

[7] Larry Wall and Randal L. Schwartz. *Programming Perl*. O'Reilly & Associates, Inc., 1990.


```
01:function [P,result] = ric(A,R,Q)
   % To solve Riccati equation using
   % Potter and Arimoto's method
   % PA + A'P + PRP = Q

05:  [n,m] = size(A);
     H = [A,R;Q,-A'];
     [v,d] = eig(H);
     cnt = 0;
     for i= 1:1:2*n
10:    if real(d(i,i)) < -eps
       cnt = cnt + 1;
       Z = [Z,v(:,i:i)];
       end
     end
15:  if cnt < n
       result = 0;
       fprintf('Number of stable eigen-
                -values is too small !\n');
       return;
     % error('Number of stable eigen-
              -values is too small !\n')
20:end
     X = Z(1:n,:);
     Y = Z(n+1:2*n,:);
     P = Y/X;
     P = real(P+P')/2;
25:  result = 1;
```

List 1 m-file (**ric.m**) for solving Riccati equation

```
01:Func List ric(A,R,Q)
     CLASS A,R,Q;
   {
     Integer result;
05:  Matrix P;

     Integer cnt,i,m,n;
     Matrix H,X,Y,Z_Z,d,v;
   /*
10: * To solve Riccati equation using
    * Potter and Arimoto's method
    * PA + A'P + PRP = Q
    */
     {n,m} = size(A);
15:  H = [[A,R][Q,-A#]];
     {d, v} = eig(H);
     cnt = 0;
     for (i = 1; i <= 2*n; i = i + (1)) {
       if (Re(d(i,i)) < -EPS) {
20:      cnt = cnt + 1;
         Z_Z = [Z_Z,v(:,i:i)];
       }
     }
     if (cnt < n) {
25:    result = 0;
       printf("Number of stable eigenvalues
               is too small !\n");
       return {P,result};
   /*
    *  error("Number of stable eigenvalues
              is too small !\n\n");
30: */
     }
     X = Z_Z(1:n,:);
     Y = Z_Z(n+1:2*n,:);
     P = Y/X;
35:  P = Re(P+P#)/2;
     result = 1;

     return { P,result };
   }
```

List 2 **ric.mm** generated from the **ric.m**