MGC : a tool for automatic linking of Fortran routines to MATLAB

Citation for published version (APA):

Document status and date:
Published: 01/01/1990

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 18. Sep. 2019
MGC - A TOOL FOR AUTOMATIC LINKING OF FORTRAN ROUTINES TO MATLAB

Version 2.2
For PC-MATLAB, MATLAB-386 and PRO-MATLAB (VAX/VMS)

W.A. Renes, December 31, 1990


WGS - report 90-2
Chapter 1 : Introduction

The MATLAB MEX facility makes it possible to call user written FORTRAN routines from the MATLAB environment (ref. 1). To do this, the user must write a small interface routine, called a gateway, which passes and converts arguments from MATLAB to FORTRAN and vice versa.

![Gateway Diagram]

Writing an efficient gateway is a time consuming and difficult task which requires a high-degree of FORTRAN programming expertise. This is caused by the following factors:

1) MATLAB matrices are passed to FORTRAN as pointers to dynamically allocated matrices in the MATLAB workspace. These matrices cannot be accessed using standard ANSI FORTRAN-77. On some platforms (VAX-VMS, IBM PC MS FORTRAN), it can be done by using non-standard compiler extensions; on other platforms only machine code can do the job.

2) All MATLAB data is in DOUBLE PRECISION format, even character strings; the gateway must therefore convert the data to or from the data type required by the FORTRAN routine. This conversion is compiler dependent; furthermore, intermediate FORTRAN arrays are needed to hold the converted data. This uses up valuable disk and memory space and also makes it necessary to make compile-time assumptions about the maximum dimensions of the matrices involved.

Since the Benelux Working Group on Software wanted to make its SLICOT library (ref. 2) available to MATLAB users, it was decided to develop a tool to facilitate the task of writing a gateway. The result of this development is the MATLAB Gateway Compiler (MGC). MGC analyses the FORTRAN source code and then generates an appropriate gateway which efficiently handles all required pointer manipulations and data conversions.
This WGS Report describes version 2.2 (December 1990) of MGC for the following platforms:

* PC-MATLAB on an IBM PC XT/AT or compatible with MicroSoft FORTRAN
* MATLAB-386 for 80386 based systems using NDP FORTRAN-386
* PRO-MATLAB for VAX/VMS using VAX FORTRAN.

Finally, I gratefully acknowledge the help of Mr. R. Kool in the preparation of this report.
Chapter 2 : Brief Usage Summary

Using MGC is very simple. Basically, if one wants to use a FORTRAN routine (a SUBROUTINE or a FUNCTION) in MATLAB, the following three steps must be taken:

1) Insert a **gateway instruction** (specifying the MATLAB header that one wants to give to the routine) in the source code of the FORTRAN routine, just above the first statement (the header) of the routine:

<table>
<thead>
<tr>
<th>FILE INCR.FOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

2) Run MGC specifying this FORTRAN source file as its input; this creates a FORTRAN gateway (INCRG.FOR in this example):

```
MGC INCR.FOR INCRG.FOR
```

3) Compile and link the FORTRAN routine and the gateway to a MATLAB mex-file:

```
MGCFORT INCR
MGCFORT INCRG
MGCLINK INCR INCRG (PC and 80386 version)
MGCLINK INCR,INCRG (VAX/VMS version)
```

The result will be a file with extension .MEX (PC), .MX3 (80386), .MEXD (VAX/VMS D-float) or .MEGX (VAX/VMS G-float) in the current directory. If one puts this file somewhere in the MATLAB path, the function is now ready to be used in MATLAB:

```
>> y = incr(100)

y = 101
```

>>
Let's now look in more detail to the gateway instruction in the above example:

<table>
<thead>
<tr>
<th>CM</th>
<th>MATLAB GATEWAY [Y]=INCR(X)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Column 7: gateway instruction  
Column 2: MGC flag character  
Column 1: FORTRAN comment character

Every line containing a gateway instruction begins with a 'C' in column 1 and an 'M' in column 2. The 'C' tells FORTRAN to regard this line as a comment. The 'M' is a special flag character used by MGC: every line flagged with this 'M' is interpreted by MGC as a gateway instruction. The gateway instruction itself starts in column 7 (or after the first TAB, if VAX FORTRAN TAB formatting is used) just as a normal FORTRAN statement.

The gateway instruction in this example, 'MATLAB GATEWAY', instructs MGC to generate a MATLAB gateway. It is directly followed by the specification of the MATLAB header one wants to give to the routine. The argument names used here (Y and X in the example) must exactly correspond to the argument names of the FORTRAN routine.

MGC uses the specification of the MATLAB header to determine which FORTRAN arguments are inputs and which are outputs. For example, X in the above example is clearly an input argument, while Y is an output argument. MGC obtains all other information required to generate a gateway (data types, array dimensions, etc.) by analyzing the FORTRAN header of the routine. Sometimes, in more complex cases involving parameters that are arrays of varying dimensions or functions, MGC needs additional information from the user. In that case extra gateway instructions must be inserted. See Chapter 4 for details.
Chapter 3: Compiling and Linking MGC Functions

3.1 The MGC command

The MGC command runs the gateway compiler. The possible forms of the command are:

\[ \text{MGC <input> <output> <list>} \]

or \[ \text{MGC <input> <output>} \]

or \[ \text{MGC <input>} \]

In this syntax diagrams, \textit{<input>} denotes the input file, a FORTRAN source file containing the FORTRAN code and one or more gateway instructions. The default extension for this file is .FOR.

\textbf{NOTE}

It is not strictly necessary to put the gateway instructions in the FORTRAN source file; one may also put them in a separate file. This is handy if one does not want to modify the FORTRAN source code, or if the source code is not available. See Chapter 6 for details.

\textit{<output>} denotes the output file with the generated FORTRAN gateway. If one omits the output file specification, the name of the input file will be used, with a 'G' appended to the file name.

\textit{<list>} denotes the list file. All error messages and a symbol table for every routine will be written to this file. If the list file specification is omitted, MGC will use the name of the output file, but with file extension .LIS.

\textbf{Some examples:}

\begin{itemize}
  \item \textbf{Command: MGC INCR}
    \begin{itemize}
      \item \textbf{Input file:} INCR.FOR
      \item \textbf{Output file:} INCRG.FOR
      \item \textbf{List file:} INCRG.LIS
    \end{itemize}
  \item \textbf{Command: MGC INCR.F TEST ERROR.TXT}
    \begin{itemize}
      \item \textbf{Input file:} INCR.F
      \item \textbf{Output file:} TEST.FOR
      \item \textbf{List file:} ERROR.TXT
    \end{itemize}
\end{itemize}

During compilation, the names of all processed modules appear on the screen. Consult the list file for details if an error has been detected.
3.2 The MGCFORT command

MGCFORT runs the FORTRAN compiler with all compiler options set as needed for MGC:

\[ \text{MGCFORT <input file>} \]

The default file extension is '.FOR'.

3.3 The MGCLINK command

The MGCLINK command links the compiled gateway to the FORTRAN objects, the MGC run-time library and the MATLAB MEX interface routines. The exact form of the command is version dependent:

3.3.1 PC and 80386 versions

The syntax of the MGCLINK command is:

\[ \text{MGCLINK <obj 1> <obj 2> ... <obj n>} \]

The result is a mex-file in the current directory named "<obj 1>.MEX" (PC) or "<obj 1>.MX3" (80386).

Example:

Suppose file TESTLIB.OBJ contains the compiled gateway for the FORTRAN routines in files TEST1.OBJ and TEST2.OBJ. To link these files to a mex-file, the required MGCLINK commands would be:

\[ \text{MGCLINK TESTLIB TEST1 TEST2} \]

The result of these operations is a mex-file named TESTLIB.MEX (PC version) or TESTLIB.MX3 (80386 version)

3.3.2 VAX/VMS version

For VAX/VMS the syntax of the MGCLINK command is as follows:

\[ \text{MGCLINK <obj 1>,<obj 2>,... <com 1>,<com 2>},... \]

The result is a mex-file named "<obj 1>.MEXD". The arguments "<com 1>"", "<com 2>" etc. must be used to specify the names of FORTRAN COMMONs and MATLAB ALIAS functions. Every COMMON block declared in the FORTRAN code must be named here. Also every MATLAB ALIAS must be named here as "ALIAS_fun", where "fun" is the name of the alias (see Section 4.4.6).
Example:

Suppose file TESTLIB.OBJ contains the compiled gateways for the FORTRAN routines in files TEST1.OBJ and TEST2.OBJ. The FORTRAN routines declare two COMMON blocks, named TESTDATA and TESTCHAR, and one MATLAB ALIAS function named "FX". The correct MGCLINK command in this case would be:

```
MGCLINK TESTLIB,TEST1,TEST2 TESTDATA,TESTCHAR,ALIAS_FX
```

The result of this operation is a mex-file named "TESTLIB.MEXD" or "TESTLIB.MEXG", depending on the floating point format used by the version of MATLAB. A mex-file with extension .MEXD uses the VAX D-floating format; a mex-file with extension .MEXG uses G-floating format. MGC automatically selects the "best" format for the system. See the READ.ME file on the MGC VAX/VMS distribution media for more details.
Chapter 4 : Passing Data between MATLAB and FORTRAN

4.1 Introduction

Passing function arguments from the MATLAB environment to FORTRAN and vice versa is the single most important task of a gateway generated by MGC. This Chapter gives some information on how a gateway performs its tasks, and explains in which cases additional gateway instructions are needed to handle arrays with unspecified dimensions.

NOTE

Please read this Chapter carefully. Especially understanding the sections on passing variable dimension arrays is necessary for every serious application of MGC.

4.2 The tasks of a gateway

The gateway generated for a FORTRAN routine to be called from MATLAB performs the following tasks:

0) For libraries only: get the name of the routine to be called from MATLAB argument 1, remove argument 1 from the list of input arguments and call the gateway of the selected routine.
1) Obtain the value of FORTRAN variables specified in a MATLAB SIZE gateway instruction (see Section 4.4.4).
2) Generate temporary FORTRAN arrays in the MATLAB workspace to be used as arguments of the FORTRAN subroutine.
3) Convert the elements of all MATLAB input matrices to FORTRAN format and store them in temporary arrays created in step 2).
4) Create MATLAB matrices for passing results back to MATLAB.
5) Call the FORTRAN routine, passing the temporary arrays created in step 2) as arguments.
6) Convert the elements of all FORTRAN output arrays to MATLAB format and store them in the MATLAB matrices created in step 4).
7) Delete all temporary FORTRAN matrices and return to MATLAB.

Since the data type MATLAB uses corresponds to FORTRAN DOUBLE PRECISION, no temporary arrays will be created and no conversions will be done for DOUBLE PRECISION arguments.

For scalar variables (arrays of dimensions 1x1) and CHARACTER variables, the temporary variables are not allocated in the MATLAB workspace, but declared directly (and therefore statically) in the gateway code itself. This has as important consequence that the dimensions of CHARACTER ARRAYS must be known at compilation time.
4.3 Supported data type conversions

The following FORTRAN data types are recognized by MGC and automatically converted from MATLAB to FORTRAN format when needed:

- LOGICAL, LOGICAL*2, LOGICAL*4 (default is LOGICAL*4)
- INTEGER, INTEGER*2, INTEGER*4 (default is INTEGER*4)
- REAL, REAL*4, REAL*8 (default is REAL*4)
- DOUBLE PRECISION
- COMPLEX, COMPLEX*8, COMPLEX*16 (default is COMPLEX*8)
- DOUBLE COMPLEX
- CHARACTER, CHARACTER*n

FORTRAN arguments of data type "DOUBLE PRECISION" do not require any conversion and are therefore handled more efficiently, both in processing time (no conversion must be done) and memory (no temporary arrays for storing the converted data are needed).

4.4 Passing array arguments

When passing arrays between MATLAB and FORTRAN, MGC needs to know the actual dimensions of the array. In most cases the required dimensions can be determined automatically by the gateway at runtime. In several cases however, MGC needs a little help from the user in the form of additional gateway instructions.

4.4.1 Explicitly dimensioned arrays

If an array is dimensioned explicitly in the FORTRAN routine, i.e. if its dimensions are indicated by constants, constant expressions, or variable expressions (containing only variables passed as input to the routine via its parameter list), the correct code to handle it can be generated by MGC without additional gateway instructions.

Example:

```
CM MATLAB FUNCTION [B]=XXX(N,M,A)
SUBROUTINE XXX(N,M,A,B)
INTEGER N,M
REAL A(N,0:M+1), B(M,100)
```

Since N and M are passed to XXX via its parameter list, their value is known to the gateway. Therefore, MGC needs no special instruction from the user for this simple case.
4.4.2 Arrays with unspecified last dimension

It is possible in FORTRAN-77 to specify the last dimension of any array with an asterisk ('*').

This is no problem for input and input/output arrays. In this case the gateway simply looks up the dimension of the corresponding MATLAB matrix, and no special instructions from the user are necessary.

For output and work arrays though, no corresponding MATLAB matrix exists yet, so looking up the MATLAB dimension is not possible. Therefore in this case extra information from the user is needed in the form of an "MATLAB ALLOCATE" gateway instruction, as in the following example:

```
CM MATLAB GATEWAY [B]=YYY(N,A)
CM MATLAB ALLOCATE B(N),WORK(2*N+10)
SUBROUTINE YYY(N,A,B,WORK)
INTEGER N
REAL A(*),B(*)
DOUBLE PRECISION WORK(*)
```

The MATLAB ALLOCATE instruction in the second line in this example specifies that B must have dimension N and WORK a dimension of 2*N+10. For input array A no extra instruction is needed, because its dimension can be obtained from the corresponding MATLAB matrix.

If the dimension of an array is explicitly declared both in an MATLAB ALLOCATE instruction and in FORTRAN, the FORTRAN dimension takes priority. In all other cases the dimension specified in the MATLAB ALLOCATE statement will be used.

The syntax of the MATLAB ALLOCATE instruction is identical to the FORTRAN DIMENSION statement. However, the rules for specifying the dimensions are somewhat relaxed: one may use any FORTRAN expression which uses constants, subroutine arguments and intrinsic FORTRAN functions, like for example:

```
CM MATLAB GATEWAY []=ZZZ(N,M)
CM MATLAB ALLOCATE WORK(2*N+10*MAX(N,M))
SUBROUTINE YYY(N,N,WORK)
INTEGER N,M
DOUBLE PRECISION WORK(*)
```
IMPORTANT

Many FORTRAN programmers use a dimension of "1" for arrays of unspecified length. This is a violation of standard ANSI FORTRAN-77 and not recognized by MGC, i.e. MGC simply assumes that the array has length 1, which may lead to unexpected results or memory protection errors.

4.4.3 Workspaces

Many FORTRAN routines need workspaces, i.e. arrays that must be passed to the routine via its argument list, which are used only internally for storing intermediate results.

MGC assumes that every argument of a FORTRAN routine that is not mentioned in its MATLAB header is a workspace and will therefore allocate it temporarily just before the routine is invoked. This temporary array will not be initialized, and its contents are lost upon return to MATLAB.

Example:

```
CM MATLAB GATEWAY [Y]=ZZZ(N,X)
SUBROUTINE ZZZ(N,X,W)
INTEGER N
DOUBLE PRECISION X(N),Y(N)
COMPLEX W(3*N+4)
```

In this example, argument W of routine ZZZ is not mentioned in the MATLAB GATEWAY instruction and therefore automatically recognized as a workspace variable.

Workspace dimensions must always be dimensioned explicitly. If the FORTRAN declaration uses an asterisk, one must use a MATLAB ALLOCATE command specifying the dimension explicitly.

4.4.4 Variables derived from input array dimensions.

To reduce the number of arguments a user must specify, many MATLAB functions use the "size" function to obtain the dimensions of an input array, as in the following example:

```
function [c]=add(a,b)
[n m]=size(a); c=zeros(n,m);
for i=1:n,
    for j=1:m,
        c(i,j)=a(i,j)+b(i,j);
    end
end
```
Since in this example n and m can be obtained from a, there is no need to pass them via the argument list.

The same technique may be used for FORTRAN routines by means of the MATLAB SIZE gateway instruction, as in the following example. This example is equivalent to the MATLAB example given above:

```
CM MATLAB GATEWAY [C]=ADD(A,B)
CM MATLAB SIZE N=ROWS(A), M=COLS(A)

SUBROUTINE ADD(N,M,A,B,C)
INTEGER N,M
REAL A(N,M),B(N,M),C(N,M)

INTEGER I,J

DO 20, I=1,N
   DO 10, J=1,M
      C(I,J) = A(I,J) + B(I,J)
   10 CONTINUE
20 CONTINUE

RETURN
END
```

Note that N and M are no longer mentioned in the MATLAB header specified in line 1. "ROWS" and "COLS" are special functions in MGC which return the number of rows and the number of columns of a matrix. These functions may only be used in a MATLAB SIZE instruction. Of course, sizes can only be derived from input and input/output matrices.

The following more complicated example shows a very common technique used by FORTRAN libraries to handle arrays with varying dimensions. In these techniques, the leading dimension (the declared dimension of the array in the calling program) and the actual dimension (the part of the array that will be used in the routine) are passed via the argument list via separate arguments:
MUL computes the product \( C \) (dimension \( N \times P \)) of the \((N \times M)\) matrix \( A \) with the \((M \times N)\) matrix \( B \). In this example, the actual dimensions \( N, M, \) and \( P \) are obtained via the SIZE command from the dimensions of MATLAB matrices \( A \) and \( B \). The leading dimensions of \( A \) and \( B \) are obtained in exactly the same manner (this example assumes that leading and actual dimensions are identical, i.e. the complete matrix passed to FORTRAN will be used). Note that the value of \( LDC \) cannot be obtained by asking the dimensions of matrix \( C \), since \( C \) does not exist yet: it will be created later as output by the routine MUL! We therefore obtain \( LDC \) from matrix \( A \), since we know that both must always have the same number of rows (because the MATLAB ALLOCATE instruction for \( C \) specifies that \( C \) will be created with \( N \) rows).

4.4.5 Input/output parameters.

In FORTRAN, it is possible to use the same argument both for input and output, e.g:

```
SUBROUTINE ADDONE(X)
INTEGER X
X=X+1
RETURN
```
The value returned for X is equal to the original value plus one. To use this function from MATLAB, X must be specified at both sides of the MATLAB header specification, i.e. both as input and as output argument:

```
CM MATLAB GATEWAY [X]=ADDONE(X)
SUBROUTINE ADDONE(X)
INTEGER X
X=X+1
RETURN
END
```

4.4.6 FUNCTION and SUBROUTINE parameters.

In FORTRAN it is possible to pass a FUNCTION or SUBROUTINE as argument to a routine. For instance, it is possible to pass a user supplied function to a subroutine which finds one or more roots of that function.

MGC supports passing MATLAB functions as arguments to a FORTRAN routine. To do this, the user must define a MATLAB ALIAS, which informs MGC about the number and type of arguments of the MATLAB function to be passed. A MATLAB ALIAS definition has the following structure:

```
CM MATLAB ALIAS <matlab header definition>
CM <FUNCTION or SUBROUTINE statement>
CM <FORTRAN declarations of arguments>
CM END
```

The definition follows the same pattern as MATLAB GATEWAY definitions: first specify the MATLAB header as wanted, then specify the corresponding FORTRAN header and the declaration of the argument types and dimensions. An 'END' statement is required to conclude the definition. Since an ALIAS definition has no meaning to the FORTRAN compiler, all lines must be preceded by the characters 'CM' in columns 1 and 2.

For each function argument of a routine, one must add a MATLAB EXTERNAL instruction to the GATEWAY definition:

```
CM MATLAB EXTERNAL FUNC:XYZ
```

Function arg.: FUNC
Separating colon
Alias identifier: XYZ
Consider the following example, which interfaces a root solver routine SOLVE to MATLAB. SOLVE\((A,B,\text{FUNC},X)\) computes a root of a FORTRAN function \text{FUNC} in the interval \([A,B]\). The root is returned in \(X\). \text{FUNC} can be any user defined \text{REAL FUNCTION} with one \text{REAL} argument. The following code is needed to use SOLVE in MATLAB as a function which can find roots of any user supplied MATLAB function.

<table>
<thead>
<tr>
<th>FILE SOLVE.FOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM MATLAB GATEWAY ([X]=\text{SOLVE}(A,B,\text{FUNC}))</td>
</tr>
<tr>
<td>CM MATLAB EXTERNAL FUNC:XYZ</td>
</tr>
<tr>
<td>SUBROUTINE SOLVE((A,B,\text{FUNC},X))</td>
</tr>
<tr>
<td>REAL (A,B,\text{FUNC},X)</td>
</tr>
<tr>
<td>EXTERNAL \text{FUNC}</td>
</tr>
<tr>
<td>RETURN</td>
</tr>
<tr>
<td>END</td>
</tr>
<tr>
<td>CM MATLAB ALIAS ([Y]=\text{XYZ}(X))</td>
</tr>
<tr>
<td>CM FUNCTION XYZ((X))</td>
</tr>
<tr>
<td>CM REAL XYZ</td>
</tr>
<tr>
<td>CM REAL (X)</td>
</tr>
<tr>
<td>CM END</td>
</tr>
</tbody>
</table>

In the above example, we have defined \text{XYZ} as a function which has one input argument and one output argument, both of type \text{REAL}. To use SOLVE in MATLAB to find a root of function 'myfunc' in the interval \([0,\pi]\), enter the following command:

```
>> x = solve\((0,\pi,'myfunc')\)
```

'myfunc' can be any MATLAB m-file or mex-file. MATLAB built-in functions may not be used directly. If one wants to use a built-in function, put it in a small m-file with the same parameters.

\text{NOTE 1:}
It is possible to use the same ALIAS definition for more than one function argument in a MGC library, provided these are not arguments of the same routine. If for instance one has a library containing several root solving functions which all find roots of a real function with one real parameter, they may all share the same alias definition. However, if one has a routine with two function arguments with the same types of parameters, one must use separate alias definitions.

\text{NOTE 2:}
A FORTRAN COMMON block with the name ALIAS\_name, where 'name' is the name of the alias, is created for each ALIAS definition. In VAX/VMS all COMMON blocks used in MGC must be specified at link-time, see Chapter 3. To link the example SOLVE given above, VMS users need the following MGCLINK command:

```
MGCLINK SOLVE,SOLVEG ALIAS\_F
```
Chapter 5: Calling MATLAB Functions from FORTRAN

MGC was originally developed for calling FORTRAN functions from MATLAB. However it can also be used to do the opposite, i.e. calling MATLAB functions (m-files) from FORTRAN. The only requirement is that the FORTRAN code from which the MATLAB function is called is itself part of an MGC-based mex-file.

To call a MATLAB function from FORTRAN, MGC must know its MATLAB header and how one wants to call it from FORTRAN. This is accomplished by means of a MATLAB FUNCTION definition block, which has the following structure:

\[
\begin{array}{l}
\text{CM MATLAB FUNCTION <mfile header definition>}
\text{CM \textless FUNCTION or SUBROUTINE statement\textgreater}
\text{CM \textless FORTRAN declarations of arguments\textgreater}
\text{CM END}
\end{array}
\]

As can be seen, a MATLAB FUNCTION definition follows the same pattern as MATLAB GATEWAY or ALIAS definitions: first specify the MATLAB header, then specify how one wants the FORTRAN header and the FORTRAN argument types and dimensions. An 'END' statement is required to conclude the definition. Since a MATLAB FUNCTION definition has no meaning to the FORTRAN compiler, all lines must be preceded by the characters 'CM' in columns 1 and 2.

The following example uses the well-known MATLAB routine ROOTS in a FORTRAN routine TEST:

\[
\begin{array}{l}
\text{CM MATLAB GATEWAY []=TEST}
\text{SUBROUTINE TEST}
\text{REAL P(0:2)}
\text{COMPLEX Q(2)}
\text{...}
\text{CALL ROOTS(2,P,Q)}
\text{...}
\text{RETURN}
\text{END}
\end{array}
\]

\[
\begin{array}{l}
\text{CM MATLAB FUNCTION [Y]=ROOTS(POLY)}
\text{CM SUBROUTINE ROOTS(N,POLY,Y)}
\text{CM INTEGER N}
\text{CM REAL POLY(0:N)}
\text{CM COMPLEX Y(N)}
\text{CM END}
\end{array}
\]

18
Chapter 6: MGC Libraries

Often it is advantageous to combine several FORTRAN routines into one single mex-file, because:

- a combined mex-file uses less disk and memory space since the routines share the MGC and MEX run-time libraries.
- the combined routines may share data, subroutines or call each other directly.
- routine activation time is less: once the first library function has been called, the entire library is immediately available without further delay caused by loading from disk.
- one can ship all the routines as a single file to end users.

A combined MEX file is called a MGC Library. Every MGC Library must start with a MATLAB LIBRARY command:

```
CM MATLAB LIBRARY TESTLIB  
library name  
Column 7: library instruction  
Column 2: MGC flag character  
Column 1: FORTRAN comment character  
```

For example, consider a MGC library TESTLIB containing two functions INCR and INCR2:

```
FILE TESTLIB.FOR  
CM MATLAB LIBRARY TESTLIB  
CM MATLAB GATEWAY [Y]=INCR(X)  
SUBROUTINE INCR(X,Y)  
INTEGER X,Y  
Y = X+1  
RETURN  
END  
CM MATLAB GATEWAY [Y1,Y2]=INCR2(X1,X2)  
SUBROUTINE INCR2(X1,Y1,X2,Y2)  
INTEGER Y1,Y2,X1,X2  
Y1 = X1+1  
Y2 = X2+1  
RETURN  
END  
```
To link the library, use the MGCLINK command as usual:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MGC TESTLIB TESTLIBG</td>
<td>(create gateway)</td>
</tr>
<tr>
<td>MGCFORT TESTLIB</td>
<td>(compile source)</td>
</tr>
<tr>
<td>MGCFORT TESTLIBG</td>
<td>(compile gateway)</td>
</tr>
<tr>
<td>MGCLINK TESTLIB TESTLIBG</td>
<td>(on PC or 80386 PC)</td>
</tr>
<tr>
<td>MGCLINK TESTLIB,TESTLIBG</td>
<td>(on VAX)</td>
</tr>
</tbody>
</table>

The result will be a file TESTLIB with extension .MEX (PC), .MX3 (80386), .MEXD (VAX/VMS D-float) or .MEXG (VAX/VMS G-float) in the current directory. If one puts this file somewhere in the MATLAB path, the library is now ready to be used in MATLAB:

To call a routine from the library, specify the library name, followed by a parameter list with the name of the selected function in position 1, and then the parameters of the function. For instance:

\[
Y = \text{TESTLIB('INCR',X)} \\
[Y_1,Y_2] = \text{TESTLIB('INCR2',X_1,X_2)}
\]

Sometimes it is not desirable or impossible to put gateway instructions directly in the FORTRAN source code. In that case one should put the instructions together with the definition of the FORTRAN headers in a separate file, as in the following example:

<table>
<thead>
<tr>
<th>FILE HEADERS.FOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM MATLAB LIBRARY TESTLIB</td>
</tr>
<tr>
<td>CM MATLAB GATEWAY ([Y]=\text{INCR}(X))</td>
</tr>
<tr>
<td>SUBROUTINE \text{INCR}(X,Y)</td>
</tr>
<tr>
<td>INTEGER X,Y</td>
</tr>
<tr>
<td>END</td>
</tr>
<tr>
<td>CM MATLAB GATEWAY ([Y_1,Y_2]=\text{INCR2}(X_1,X_2))</td>
</tr>
<tr>
<td>SUBROUTINE \text{INCR2}(X_1,Y_1,X_2,Y_2)</td>
</tr>
<tr>
<td>INTEGER Y_1,Y_2,X_1,X_2</td>
</tr>
<tr>
<td>END</td>
</tr>
</tbody>
</table>

To compile and link library TESTLIB with a separate headers file, use the following commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MGC HEADERS.FOR TESTLIBG.FOR</td>
<td>(compile gateway)</td>
</tr>
<tr>
<td>MGCFORT TESTLIBG</td>
<td>(PC and 80386 version)</td>
</tr>
<tr>
<td>MGCLINK TESTLIB TESTLIBG</td>
<td>(VAX/VMS version)</td>
</tr>
</tbody>
</table>
Chapter 7 : MGC Version 2.2 Implementation Notes

A number of restrictions and limitations apply to the 2.2 release of MGC for MS-DOS and VAX/VMS. They will most likely be removed in future versions of the product.

a) MGC only recognizes FORTRAN FUNCTIONS which begin with the word FUNCTION. If FUNCTION is preceded by a data type, it is not recognized.

So this is NOT accepted:

```
CM MATLAB GATEWAY [Y]=INCR(X)
INTEGER FUNCTION INCR(X)
INTEGER X
```

While this will be accepted:

```
CM MATLAB GATEWAY [Y]=INCR(X)
FUNCTION INCR(X)
INTEGER INCR
INTEGER X
```

b) The DOUBLE COMPLEX data type in MicroSoft FORTRAN is not correctly supported. COMPLEX*16 however works without any problem.

c) In Release 2.2 CHARACTER strings are statically declared in the gateway and must therefore have a constant length. If the FORTRAN routine specifies a variable length, use the MATLAB ALLOCATE instruction to override this with a constant length.

d) The maximum number of array dimensions is two; the maximum number of dimensions of a CHARACTER array is 1. These are limitations imposed by MATLAB.

e) FORTRAN one-dimensional arrays correspond with column vectors in MATLAB. Beware if one wants to pass row vectors from MATLAB to FORTRAN or vice versa: one should use (1*n) two-dimensional arrays in FORTRAN. Otherwise, on input only the first element will be passed correctly, and all other elements will be equal to zero. On output, one-dimensional FORTRAN output arrays will always be converted to MATLAB column vectors.
Chapter 8 : References


Appendix: Overview MGC Commands

A.1. Introduction

This reference section gives a formal description of all the instructions recognized by the MATLAB Gateway Compiler. These instructions can be divided into two groups:

1. Gateway Instructions
2. Command Line Instructions

Gateway instructions are included in the input file processed by MGC (normally a FORTRAN source file) and tell MGC how to interface a given FORTRAN routine to MATLAB or vice versa.

Command Line Instructions are specified in the command line by which one invokes MGC, or by which one compiles and links routines in order to create a mex-file that can be executed by MATLAB.

Throughout this Appendix, the following notation is used to describe the syntax of an instruction or command:

<description> : refers to a syntax element defined elsewhere
::= : defines the syntax element at the left by the definition at the right
| : separates two alternative syntax elements
/.../ : used to enclose an optional syntax element
{} : used to enclose syntax elements that may be repeated zero, one or more times

Line formatting characters (see Section A.2.1) will not be shown in the syntax diagrams.

A.2 Gateway Instructions

A.2.1 Format

The general format of a gateway instruction follows the FORTRAN standard, with one extension: the flag character 'M' in position 2 of a source line. A complete instruction consists of line formatting characters (such as the comment character, the MGC flag character, spaces, tabs and carriage returns) and a statement part, which starts in column 7:

```
CM <statement>
```

- Column 7: Statement, starts here and may extend to column 72
- Column 6: Reserved for continuation character
- Column 2: MGC flag character
- Column 1: FORTRAN comment character
Due to the comment character in column 1, a gateway instruction will be ignored by the FORTRAN compiler. MGC also ignores all comment lines, except those that also have the flag character 'M' in column 2. Such lines are interpreted by MGC as ordinary source lines and will normally contain MGC gateway instructions.

NOTE

If a FORTRAN source file contains ordinary comment lines which have an 'M' in column 2, one should remove or adapt them. Otherwise they will be interpreted by MGC as a gateway instruction line.

If a statement is so long that it would extend beyond column 72, it must be divided into several source lines. In this MGC follows the standard FORTRAN rules: a source line may be extended by the next source line by putting a continuation character in column 6 of this next line. The continuation character can be any valid FORTRAN character, although '+' or '*' are commonly used for this purpose. Remember that continuation lines for gateway instructions should always begin with a 'C' in column 1 and an 'M' in column 2, e.g.:

```
CM MATLAB GATEWAY [Y]=VERYLONGFUNCTIONEXAMPLE
CM + (A,B,C,D,E,F,G,H,I,J)
```

Apart from the official format given above, MGC also supports VAX FORTRAN TAB formatting, in which a TAB character (ASCII code 9) may be used instead of spaces to separate the flag characters 'C' and 'M' from the remainder of the instruction, for example:

```
CM<TAB character>MATLAB GATEWAY [Y]=F(X)
```

A.2.2 Basic Syntactic Elements

The following basic syntactic elements are used in the syntax diagrams given in the subsequent paragraphs:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;alfa&gt;</td>
<td>::= &lt;any alfabetic character&gt;</td>
</tr>
<tr>
<td>&lt;digit&gt;</td>
<td>::= &lt;any decimal digit&gt;</td>
</tr>
<tr>
<td>&lt;sign&gt;</td>
<td>::= +</td>
</tr>
<tr>
<td>&lt;integer&gt;</td>
<td>::= /&lt;sign&gt;/&lt;digit&gt;/{&lt;digit&gt;}</td>
</tr>
<tr>
<td>&lt;identifier&gt;</td>
<td>::= &lt;alfa&gt;{&lt;alfa}</td>
</tr>
</tbody>
</table>

1234, -12 are valid examples of <integer>
TEST1, R2D2 are valid examples of <name>.
123HELLO is an invalid <name>, since it starts with <digit>

A.2.3 FORTRAN syntax

The following diagrams give a summary of the FORTRAN syntax recognized by MGC:
\textbf{<expr> ::= <any scalar FORTRAN expression>}
\textbf{<var expr> ::= <any scalar FORTRAN expression involving only constants or scalar formal routine parameters (arguments)>}
\textbf{<const expr> ::= <any scalar FORTRAN expression involving constants only>}

Note that by definition any <const expr> is also a <var expr> and an <expr>. Likewise, any <var expr> is also an <expr>. Example:

1+\text{MIN}(10,20) is a valid <expr>, <var expr> and <const expr>. It is a valid <var expr> only if \text{N} is defined as a scalar input argument of the routine. It is a valid <const expr> only if \text{N} is defined as a constant by means of the FORTRAN PARAMETER statement.

\textbf{<dim expr> ::= <var expr>/:<var expr>/}
\textbf{<dimension> ::= (/<dim expr>,/<dim expr>*)}

\textbf{<variable> ::= <identifier>}
\textbf{<type name> ::= \text{INTEGER}|\text{REAL}|\text{DOUBLE PRECISION}|
\\hspace{1em} |\text{LOGICAL}|\text{DOUBLE}|\text{COMPLEX}|\text{CHARACTER}}
\textbf{<type length> ::= (/\langle\text{digit}\rangle\langle\text{digit}\rangle|/)}
\textbf{<data type> ::= <type name>*<type length>/}
\textbf{<data decl> ::= <variable>*<type length>//<dimension>/}
\textbf{<FORTRAN decl> ::= <data type><data decl>{,<data decl}}

\textbf{<routine> ::= <identifier>}
\textbf{<arg list> ::= (<variable>{,<variable>})}

\textbf{<FORTRAN sbr> ::= \text{SUBROUTINE} <routine name>/<arg list>/}
\textbf{<FORTRAN fun> ::= \text{FUNCTION} <routine name>/<arg list>/}

\textbf{<FORTRAN hdr> ::= <FORTRAN sbr>|<FORTRAN fun>}

\textbf{A.2.4 MATLAB syntax}

The following MATLAB syntax is recognized by MGC:

\textbf{<MATLAB outp>::= /<variable>{,<variable}>/}
\textbf{<MATLAB fun> ::= <identifier>}
\textbf{<MATLAB inp> ::= <variable>{,<variable}>}
\textbf{<MATLAB hdr> ::= [<MATLAB outp]=<MATLAB fun>/(<MATLAB inp)>/}

The name of each <variable> should exactly match the name of the corresponding argument of the FORTRAN routine. Examples of <MATLAB hdr>'s are:

\text{[Y1,Y2]=XYZ(A,B)} header of a function with 2 outputs (Y1 and Y2) and two inputs (A and B).
\text{[]=TEST(X)} header of a function with no outputs and 1 input
\text{[Y]=TEST} header of a function with no inputs and 1 output
A.2.5 Definition of a MATLAB GATEWAY

A MATLAB GATEWAY definition defines how a FORTRAN routine or function is to be called from MATLAB. It consists of a MATLAB GATEWAY statement, optionally followed by MATLAB SIZE, MATLAB EXTERNAL and/or MATLAB ALLOCATE statements. The definition must immediately precede the <FORTRAN hdr> of the corresponding FORTRAN routine. It is defined by the following syntax diagrams:

<alias> ::= <identifier>
<ext def> ::= <variable>:<alias>
ALLOC def> ::= <variable><dimension>
<size expr> ::= ROWS(<variable>)|COLS(<variable>)
<size def> ::= <variable>=<size expr>

<gateway instr> ::= MATLAB GATEWAY <MATLAB hdr>
{MATLAB ALLOCATE <ALLOC def>{,<ALLOC def>}}
{MATLAB SIZE <SIZE def>{,<SIZE def>}}
{MATLAB EXTERNAL <EXTERNAL def>{,<EXTERNAL def>}}

The <alias> defined in the MATLAB EXTERNAL statement must be defined elsewhere in the same input file by means of the MATLAB ALIAS statement.

The MATLAB ALLOCATE statement is functionally equivalent to the FORTRAN DIMENSION statement. Dimensions specified here override unspecified dimensions (by means of an '*'') in the FORTRAN source.

The functions ROWS(x) and COLS(x) may only be used in the MATLAB SIZE statement. They deliver the actual dimensions of the corresponding MATLAB matrix supplied as input to the routine.

A.2.6 Definition of a MATLAB FUNCTION

A MATLAB FUNCTION definition defines how a MATLAB function can be called from FORTRAN. It consists of a MATLAB FUNCTION statement, immediately followed by the definition of a dummy FORTRAN subroutine or function header, the definition of the data types and dimensions of the arguments, and finally an END statement. Since it is a dummy definition, the entire FORTRAN header must specified with the FORTRAN comment indicator 'C' in column 1 and the MGC flag character 'M' in column 2 (this is not shown in the syntax diagram).

<gateway instr> ::= MATLAB FUNCTION <MATLAB hdr>
<FORTRAN hdr>
{<FORTRAN decl>}
END

When the input file is compiled by MGC, a FORTRAN routine with the specified name will be generated, which can be called by other FORTRAN routines in the same mex-file.
A.2.7 Definition of a MATLAB ALIAS

A MATLAB ALIAS definition defines how a MATLAB function should be passed as an external function to a FORTRAN routine. It consists of a MATLAB alias statement, immediately followed by the definition of a dummy FORTRAN subroutine or function header, the definition of the data types and dimensions of the arguments, and finally an END statement. Since it is a dummy, the entire FORTRAN header must be specified with the FORTRAN comment indicator 'C' in column 1 and the MGC flag character 'M' in column 2 (this is not shown in the syntax diagram).

\[
\text{<gateway instr> ::= MATLAB ALIAS <MATLAB hdr>}
\text{<FORTRAN hdr>}
\text{\{<FORTRAN decl>\}}
\text{END}
\]

To use this alias in a gateway definition, the MATLAB EXTERNAL statement must be used (see Section A.2.5).

A.2.8 Definition of a MATLAB LIBRARY

A MATLAB LIBRARY is an input file which starts with a MATLAB LIBRARY command, followed by a number of MATLAB GATEWAY, MATLAB FUNCTION and MATLAB ALIAS definitions. It has the following syntax:

\[
\text{<library name> ::= <identifier>}
\text{<library instr> ::= MATLAB LIBRARY <library name>}
\]

Compiling and linking a MATLAB LIBRARY will result in a single mex-file.

A.3 Command Line Instructions

A.3.1 The MGC command

The MGC runs the MGC compiler. It should be invoked from the command line interpreter from the operating system (or from the MATLAB interpreter using the shell operator "!")) and has the following syntax:

\[
\text{<MGC command> ::= MGC <inp file>/<outp file>/<list file>://}
\]

<input file>, <outp file> and <list file> are specifications of files and must obey the filenaming conventions of the operating system. The default extension of the input file is ".FOR". The default name of the output file is the name of the input file, with a "G" appended. The default name of the list file is the name of the output file, but with file extension ".LIS".

27
A.3.2 The MGCFORT command

The MGCFORT command runs the FORTRAN compiler with all compiler options selected as required for compiling mex-files. It has the following syntax:

<MGCFORT command> ::= MGCFORT <input file>

<input file> must be a valid file specification for the operating system. Due to limitations in MSDOS batch file processing, in MSDOS this file must be specified WITHOUT extension. The default extension used is ".FOR".

A.3.3 The MGCLINK command

The MGCLINK command links the compiled gateway to the FORTRAN objects files, the MGC run-time library MGC.LIB and the MATLAB MEX interface library MEX.LIB. The exact form of the command is version dependent:

A.3.3.1 PC and 80386 versions

The syntax of the MGCLINK command is:

<MGCLINK command> ::= MGCLINK <object file>{<object file>}

The result is a mex-file in the current directory with the same name as the first object file supplied, but with extension ".MEX" (PC) or ".MX3" (80386).

A.3.3.2 VAX/VMS version

For VAX/VMS the syntax of the MGCLINK command is as follows:

<obj file list> ::= <object file>,<object file>
<common list> ::= <identifier>,<identifier>

<MGCLINK command> ::= MGCLINK <obj file list>{<common list>}

The result is a mex-file named with the same name as the first object file, but with extension ".MEXD" (VAX D-Floating) or ".MEXD" (VAX G-Floating). The common list is a list of the names of all the FORTRAN COMMON BLOCKS referenced by the mex-file, as explained in Chapter 3.