Science & Technology Facilities Council

Technical Report
RAL-TR-2011-026

Guidelines for the development of MATLAB interfaces for HSL packages Revised for MATLAB 2011a

M Arioli, IS Duff, JD Hogg, HS Thorne

December 2011

RAL Library
STFC Rutherford Appleton Laboratory
R61
Harwell Oxford
Didcot
OX11 0QX

Tel: +44(0)1235 445384
Fax: +44(0)1235 446403
email: libraryral@stfc.ac.uk

Science and Technology Facilities Council reports are available online at: http://epubs.stfc.ac.uk

ISSN 1358- 6254

Guidelines for the development of MATLAB interfaces for HSL packages (revised for MATLAB 2011a)

M. Arioli, I. S. Duff, J. D. Hogg, and H. S. Thorne

ABSTRACT

In this report, we describe the modus operandi for providing MATLAB interfaces for HSL and GALAHAD codes. We discuss the file structure for the MATLAB interface and how the file should be constructed. We also provide details of the `HSL_MATLAB` package, the user documentation of which complements this report, and discuss how the user can install the resulting software.

This report supersedes the previous guidelines for MATLAB interfaces in report RAL-TR-2010-013.

Keywords: MATLAB interfaces, HSL, Fortran

Current reports are available from `http://www.cse.stfc.ac.uk/nag/reports.shtml`
The work was supported by EPSRC grant EP/F006335/1

Computational Science and Engineering Department
STFC Rutherford Appleton Laboratory
Oxfordshire OX11 0QX

December 6, 2011

Contents

1 Introduction

These guidelines are a simple indication of a safe and consistent approach to the design of interfaces between HSL or GALAHAD Fortran codes and the MATLAB environment. Although developers can maintain reasonable freedom, we strongly encourage them to follow these guidelines when designing and coding the interface.

MATLAB interfaces provide a mechanism for using HSL or GALAHAD packages within the MATLAB environment. MATLAB is widely used so the provision of such an interface can be of great benefit to current HSL or GALAHAD users and should also increase the number of users of this software.

A MATLAB interface is distributed with selected HSL or GALAHAD packages and consists of the following, in addition to the standard components of a package:

- a `mex` file for the HSL or GALAHAD package;
- a `mex` file containing the `hsl_matlab` or `galahad_matlab` package;
- an installation file: `<packagename>_install.m`;
- a file `README` containing general information about using the interface;
- a file `INSTALL` containing installation instructions for the interface;
- for some HSL codes, additional MATLAB `mex` files may be present if multiple interfaces are required. For example, for the HSL packages `MA57` and `ME57`, there are interfaces that return the factors rather than just solve a set of equations.
- for each `mex` file, a corresponding `.m` file containing online documentation. If the command is invoked without compiling the `mex` file first, the contents of these files will be executed. Hence they should print a MATLAB error message to the effect that the interface needs to be compiled before use.
- optionally additional `.m` files providing a simplified wrapper around the `mex` file.

We will describe these components in the following sections as well as discussing the directory structure, documentation, and testing. We focus in what follows on HSL codes: essentially identical ideas are used within GALAHAD.

The reader may find it useful to consult the documentation for the associated `HSL_MATLAB` package, and to consult a pre-existing MATLAB interface. We recommend the one for `HSL_MA48` as it covers most common situations likely to be encountered in interfacing of HSL and GALAHAD codes.

2 Fortran codes for the HSL package

When we develop a MATLAB interface, our starting point will be the HSL Fortran package. **As the interface will be distributed with the Fortran package as part of the library, any changes required to the main HSL Fortran package must be reflected in and tested with the MATLAB interface and vice-versa.**

There may be occasions when minor changes need to be carried out within the Fortran code to enable the provision of a MATLAB interface. Such changes should be made in line with the normal procedure for updating the Fortran package and should be passed to the HSL Manager at the same time as the MATLAB interface. For example, in the Fortran 95 HSL codes, it is

common practice to use a derived data type that contains components that the user must not alter. To ensure this, the components are declared as `private`. However, within our MATLAB interface, if we need access to these components, we will need to remove the `private` declaration. Similarly, if any of the dependencies need modification, the associated HSL packages should be formally updated.

3 The `mex` file

The main component of the MATLAB interface is its `mex` file: `mex` stands for *MATLAB executable*. We have chosen to compile the `mex` file from Fortran 95 source. The `mex` file is a way to call Fortran (or C) routines from MATLAB as if they were built-in MATLAB functions. A major complication with these files is that the interface between Fortran and MATLAB is far from straightforward. In particular, MATLAB pointers are quite different from those used in Fortran and there is no direct equivalent of the derived data types used in Fortran. Normally, the name of the `mex` file determines the name of the MATLAB function it provides. For example, the file `factorlu.mex` will produce a MATLAB function that can be called as follows

```
[output_arguments] = factorlu(input_arguments)
```

Each source file defining a HSL MATLAB interface should include two components:

- the gateway subroutine (see Section 3.1)
- the statement: `use hsl_matlab` (see Section 3.2)

In Listing 1, we provide the skeleton of a MATLAB interface. We will then describe the components of the file in more detail.

Listing 1: Skeleton of mex file

```
! The gateway routine
! (NB name of MATLAB function created is taken from
!  filename)
SUBROUTINE mexFunction(nlhs, plhs, nrhs, prhs)
use hsl_matlab
! Other 'use' statements

implicit none

! mex declarations
integer*4 :: nlhs, nrhs
integer(mwp_) :: plhs(*), prhs(*)

! Other variable declarations

! Body of the subroutine (in Fortran)

END SUBROUTINE mexFunction
```

3.1 The gateway routine

The *gateway routine* is the entry point to the `mex` interface. It is through this routine that MATLAB accesses the subroutines from the HSL package. The name of the gateway routine must be `mexFunction` and it must contain the parameters `nlhs`, `plhs`, `nrhs` and `prhs`, where `nrhs` and `nlhs` are the number of input and output arguments, respectively, and `prhs` and `plhs` are arrays of MATLAB pointers to the input and output arguments, respectively.

It is good practice to give alias names to the input and output variables even if it is not necessary. In Listing 2, we provide an example.

Listing 2: Alias in mex file

```
! Expect calls of the form:
! [L, U] = factor(A)
SUBROUTINE mexFunction(nlhs, plhs, nrhs, prhs)
use hsl_matlab
...
implicit none

! Dummy arguments
integer*4 :: nlhs, nrhs
integer(mwp_) :: plhs(*), prhs(*)

! Use parameters to refer to position of each argument
! for code clarity and ease of maintenance
integer, parameter :: A_pos = 1
integer, parameter :: L_pos = 1
integer, parameter :: U_pos = 2

! prhs(A_pos) is then a pointer to A
! plhs(L_pos) is then a pointer to L
! plhs(U_pos) is then a pointer to U

! Body of the subroutine
! This should check that nlhs = 2, nrhs = 1, and that
! the argument A is in fact a square matrix before
! performing the factorization.
...

END SUBROUTINE mexFunction
```

3.2 `hsl_matlab.F90`

The `hsl_matlab.F90` package is a Fortran-based module that provides interfaces and kind definitions to a common subset of the full MATLAB API. It also provides typesafe and error checked wrappers around these routines. This enables portability of the interfaces between 32 and 64-bit machines and also allows for better maintainability of the code. For example, any changes to these underlying routines will only result in an alteration to the `hsl_matlab` package. If you find that you need to use a MATLAB API function that is not available via an interface

within hsl_matlab, then you should update the hsl_matlab package to include an interface to this function.

The hsl_matlab package is part of the HSL library, but is not distributed by itself to users. The documentation is available to HSL developers in the file:

/numerical/num/hsl2011/packages/hsl_matlab/hsl_matlab.pdf .

3.3 MATLAB specific data type

To enable cross-platform flexibility, the MATLAB Fortran API contains several preprocessor macros (alternatively, MATLAB data types) that are used by hsl_matlab functions. To avoid the need for use of the preprocessor, hsl_matlab.F90 declares equivalent kind parameters instead.

- integer(mws_) (equivalent to mwSize) is a data type for sizes, such as array dimensions and number of elements.
- integer(mwi_) (equivalent to mwIndex) is a data type for index values, needed to identify components of arrays.
- integer(mwp_) (equivalent to mwPointer) is a data type for a MATLAB pointer. MATLAB uses a special C data type, the mxArray. Because there is no way to create such a data type in Fortran, MATLAB passes an identifier of this type to the Fortran program. It can then be used in calls to the MATLAB API to access the data to which it refers.

The exact kinds of these types varies depending on whether a 32-bit or 64-bit platform is used, and on what compiler flags are passed to mex.

These MATLAB related types are only required for interaction with MATLAB. For all other purposes normal Fortran types may be used.

3.4 Body of the mex code

3.4.1 Initial Check

The first check is on the arguments nrhs and nlhs of the gateway function. These must be checked to ensure that the correct number of arguments have been passed to the function. In the following example, there must be between 1 and 3 input arguments and between 1 and 5 output arguments.

```
IF (nrhs < 1 .or. nrhs > 3) THEN
  CALL MATLAB_error( "Wrong number of input arguments" )
END IF
IF (nlhs < 1 .or. nlhs > 5) THEN
  CALL MATLAB_error( "Wrong number of output arguments" )
END IF
```

The subroutine MATLAB_error prints an error message and returns the user to the MATLAB prompt. If a warning is more appropriate, use MATLAB_warning instead.

3.4.2 MATLAB structures

In the Fortran 95 HSL codes, it is common practice to define data types to hold the control parameters, information details, and data that needs to be stored from one call to the next. In MATLAB, we can use *structures* to carry out the same functionality. Unfortunately, we have discovered that there is currently a restriction on the use of array components within structures. Arrays can be defined and assigned to within a MATLAB structure but they cannot be read back from the structure within the interface. Hence, arrays need to be passed separately.

3.4.3 Allocation and deallocation

Having declared the local variables, storage must be allocated for each array to be used in the `mex` file. The command

```
ALLOCATE( name( n+1 ), STAT = err  )
```

will allocate the array `name` of dimension `n+1`. If the allocation is successful, `err` will be equal to zero. Otherwise, an error has occurred (normally that there is insufficient memory available). Thus, it is necessary to check if the allocation was successful:

```
IF (err .ne. 0) THEN
  CALL MATLAB_error( " Failure when attempting to allocate memory " )
END IF
```

Before the final return to MATLAB, it is necessary to deallocate all the arrays that have been used. Input or output variables allocated using MATLAB calls must not be deallocated.

4 Installation files

4.1 Directory structure

The MATLAB interface is treated as part of the library package and sits in its own `matlab` subdirectory of the main package directory.

The files in this directory must all be prefixed with the package name, except for the `README` and `INSTALL` files. This ensures that multiple HSL MATLAB interfaces can be stored in the same directory.

Typical files inside the `matlab` subdirectory:

`README` user documentation on how to use the interface

`INSTALL` guide on how to compile the matlab interface

`<packagename>_install.m` installation script

`<packagename>_test.m` test script

`<packagename>_test_data1.mat` MATLAB data for test, file 1

`<packagename>_test_data2.mat` MATLAB data for test, file 2

`<packagename>_full_test.m` comprehensive test of interface

`<filename>.output` sample output from running the test with the matching filename.

The `matlab` directory should not contain any subdirectories except an optional `examples` subdirectory containing several MATLAB files and data files that give examples of how to use the interface (supplementary to the `<packagename>_test.m` file).

It is our convention to use the extension `.output` for example output files, and `.log` for output files generated when the user runs the code. This prevents example outputs being overwritten accidentally.

4.2 The `<packagename>_install.m` file

Installation is performed using the MATLAB file `<packagename>_install.m`. It can have several input arguments passed using the `varargin` parameter:

`<packagename>_install()` installs `<packagename>` and its MATLAB Interface. It is assumed that the BLAS and LAPACK routines provided by MATLAB are used, and mex is configured to use your preferred compiler. The test example is not run.

`<packagename>_install(TEST)` installs `<packagename>` and its MATLAB interface and optionally runs the test example. It is assumed that the BLAS and LAPACK routines provided with the interface are used, and mex is configured to use the preferred compiler. If `TEST` ≤ 0, the test example is not run; if `TEST` > 0, the test example is run.

`<packagename>_install(TEST,LIBS)` installs `<packagename>` and its MATLAB interface and optionally runs the test example. It is assumed that mex is configured to use the preferred compiler.
If `LIBS` has the value `'matlab'` this is equivalent to the setting `LIBS='-mwlapack -mwblas'`. Use of this option on a 64-bit platform will force use of 64-bit default integers and may degrade the performance of the HSL code.
Otherwise `LIBS` should be set to specify which BLAS to link against, and may optionally specify the location of other libraries (eg libf95.a libgcc.a if they are not on a default search path, libmetis.a if required). Typically this will take the form of `LIBS='-L/path/to/blas -lblas'`.

`<packagename>_install(TEST,LIBS,MEXFLAGS)` installs `<packagename>` and its MATLAB interface and optionally runs the test example. The contents of the variable `MEXFLAGS` is passed to mex as follows.
```
mex $(MEXFLAGS) -c file.F90
mex $(MEXFLAGS) $(LIBS) -output foo.mex file.F90
```
If `MEXFLAGS` is not supplied it assumes the default value of `'-largeArrayDims'` on a 64-bit platform and is empty on a 32-bit platform.
If `MEXFLAGS` is supplied and `LIBS='matlab'` on a 64-bit machine, the user must ensure that the relevant flag to force 64-bit default integers is passed to the compiler by explicitly setting `FFLAGS='-i8 -fPIC'` (g95) or `'-fdefault-real-8 -fPIC'` (gfortran). (The `-fPIC` is needed as the mex default value of `FFLAGS` uses `-fPIC` to create a shared library).

The `<packagename>_install.m` file has five parts:

1. Initially, it checks whether it is on a supported operating system and, if so, whether the version of MATLAB is recent enough for the installation of the interface. It also determines whether it is on a 32 or 64-bit machine.

2. It sets the variables `LIBS` and `MEXFLAGS` correctly for the supplied options.

3. All Fortran files are compiled with the `mex` MATLAB command.

4. It adds the directory in which the resulting mex file resides to the current MATLAB and Java paths.

5. It runs `<packagename>_test.m` if required.

5 MATLAB codes to enable simple call of subroutines

While individual `mex` files can be prepared for each function of a package, we recommend against this approach if the package features user-opaque data objects (such as the `keep` variable in many recent packages).

Instead, different functions can be specified by using a text string as the first argument, for example `'factor'` and `'solve'`. The user-opaque data object can be stored using a Fortran pointer in a module level global array with the `save` attribute. An integer handle is then passed back to the MATLAB user through which they may refer to the data object. Having all routines offered by a single `mex` file reduces the risks of data corruption.

Small `.m` files can be written that wrap this monolithic function using more user-friendly names. For example, the `HSL_MA48` interface uses a single `mex` function `hsl_ma48_expert`. The routines `hsl_ma48_factor`, `hsl_ma48_solve`, `hsl_ma48_backslash` and `hsl_ma48_destroy` are implemented as simple MATLAB functions.

6 Interface testing

As described in Section 4.1, the developer must supply a `<packagename>_test.m` file that can be used either to check the installation and/or to supply useful examples of execution with the different input and output arguments.

In addition, an automated exhaustive testing of the interface should be performed by the script `<packagename>_full_test.m`. To test errors such as the wrong number of arguments, a try-catch structure should be used. An example that tests for no arguments is shown in Listing 3. This mechanism enables all the interface tests to be included in a single script with each test similar to that shown in Listing 3.

Listing 3: Catching exceptions

```
% No inputs
try
    x = hsl_mi20();
catch
    errstr = lasterror;
    str=strtrim(errstr.message);
    str1=strtrim(['hsl_mi20 requires at least 1' ...
        'input argument']);
    if (size(strfind(str,str1),2)==0)
        error('Failure at error test 1')
    end
end
```

The newer version of the `try-catch` construct that replaces `catch; errstr = lasterror` with the simpler `catch errstr` version **must not** be used as this is incompatible with older versions of MATLAB. To test any warnings, `lastwarn` may be used instead of `lasterror`.

Before an update is accepted to a library code this test and the Fortran comprehensive test must both complete successfully. The MATLAB code must work with all versions of MATLAB released in the last 3 years on both 32-bit and 64-bit Linux.

7 Pitfalls

For those used to Fortran programming and the good diagnostics available from most Fortran compilers, the MATLAB environment can come as a shock, in particular the debugging of mex files is not always a pleasurable business. In this section, we highlight some of the problems we have already encountered and request that any other issues are reported to the HSL Manager so that this section can be expanded accordingly.

- One of the main issues is that MATLAB structures and pointers do not really have anything in common with Fortran derived data types or pointers.
- Variables in MATLAB are case sensitive so that variable `jimmy` is quite different from `Jimmy`, for example.
- `g95` gets confused about the type of default integer when using its `-i8` option. This can often be resolved using an integer kind of `selected_integer_kind(5)` (only do this in those cases where it complains).
- If an array is allocated both within a mexfile and in a Fortran subroutine called by the mexfile, no warning or error flag is raised but the program will fail badly, usually with a segmentation fault.
- There are very few helpful diagnostic messages when a MATLAB program fails, a return is always made back to the MATLAB calling environment and very often the MATLAB window is closed. Running in the command window without using the JVM (i.e. running as `matlab -nojvm`) can help by allowing debug messages to be printed with the standard Fortran `print` command.
- **All** files that call MATLAB functions, directly or indirectly, must be compiled with `mex`. Failure to do so can result in strange errors. In particular, this applies to the `hsl_matlab.F90` module.

8 Documentation

8.1 The `README` file

The `README` file contains details about the installation requirements and the use of the MATLAB interface. Some of this information will be repeated in the online documentation (Section 8.2). The `README` file should contain the following sections:

1. A short introduction that describes what the package is used for, any assumptions made, and references.
2. The requirements for installing the interface
   - recommended version(s) of MATLAB and Fortran compilers;
   - the MATLAB environment variables point to the correct place.

3. The directory structure with a description of the files and subdirectories.

4. Reference to installation instructions contained in the `INSTALL` file.

5. A description of the MATLAB interface `<packagename>.m` and of the test examples. Some MATLAB interfaces will be only a way to give on-line information. Other interfaces will be more sophisticated and will allow the user to use more options.

6. A description of any structure containing control parameters.

7. A description of any structure containing information components.

8.2 On-line documentation

The documentation for the code is included in the `<packagename>`.m file so that it can be viewed using the MATLAB `help <packagename>` command. Analogously, with the command `help hsl_install` installation documentation can be seen on-line.