Matlab – External interfaces

Stefan Johansson

Department of Computing Science
Umeå University

Outline

- External interfaces
  - □ MEX-files
- Some about
  - □ Java
  - □ Component Object Model (COM)
  - □ Matlab Compiler

External interfaces

- Possible to call and use external software and libraries in Matlab
- Programs/routines written in
  - □ C
  - □ C++
  - □ Fortran
  - □ Java (built-in support)

$\left.\begin{matrix}\text{C}\\ \text{C++}\\ \text{Fortran}\end{matrix}\right\}$ Must be compiled into a binary MEX-file

Why call external routines

- Faster execution
- Can use and exchange information with existing software and libraries
  - □ Do not need to rewrite them as m-files
- Can use external GUIs, e.g. written in Java or Visual Basic

External interfaces

- Shared libraries
  - .dll (Windows)
  - .so (UNIX and LINUX)
- Only Windows
  - COM (Component Object Model)
  - Microsoft .NET Framework

Can be accessed directly from Matlab (without any need of a gateway routine)

MEX-files

- External C and Fortran routines compiled for Matlab are called *MEX-files*
- MEX-file extensions
  - MS Windows
    - .mexw32 / .mexw64 (from Matlab v. 7.1)
    - .dll (until Matlab v. 7.0.4)
  - LINUX
    - .mexglx / .mexa64
  - Sun Solaris
    - .mexsol / .mexs64

MEX-files

- C compiler for Windows included (*LCC*)
  - Matlab has support for several external compilers (e.g., *MS Visual Studio*)
- External C compiler required for LINUX and UNIX (e.g., *gcc*)
- External Fortran compiler required (e.g., *gfortran* and *Intel Fortran*)

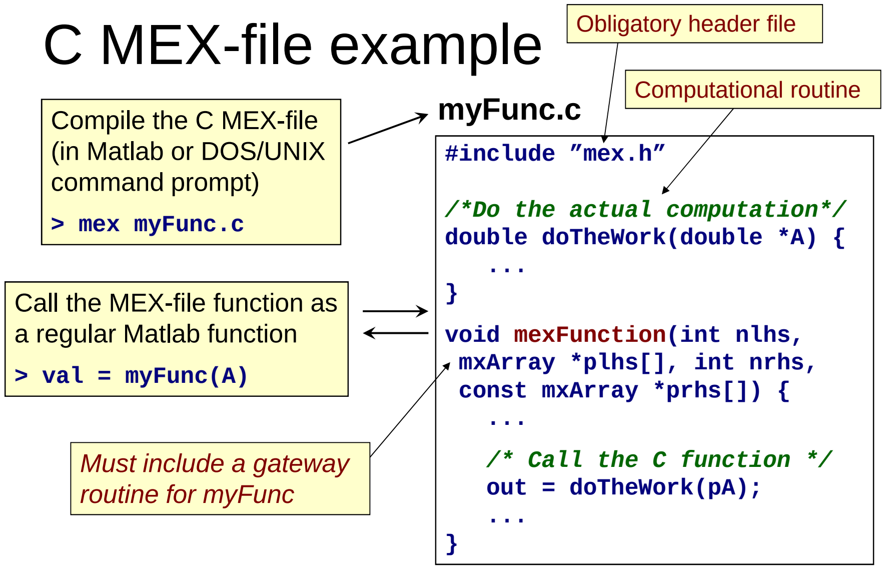
Integrating C files with Matlab

- A *gateway routine* is needed as an interface between the *computational routine(s)* and Matlab
- The gateway routine must be called *mexFunction* (compare with the *main* function)
- Use the *mex* command to compile the computational routine(s) together with the gateway routine into a binary MEX-file
  - mex can both be called within Matlab and from the Windows/Linux command prompt

The MEX-file

- Computational and gateway routines can be in separate files
- The first file supplied to the *mex* command must include the gateway routine
- The name of the binary MEX-file is the name of the function called from Matlab

C MEX-file example



Obligatory header file

Computational routine

Compile the C MEX-file (in Matlab or DOS/UNIX command prompt)

```
> mex myFunc.c
```

Call the MEX-file function as a regular Matlab function

```
> val = myFunc(A)
```

Must include a gateway routine for myFunc

myFunc.c

```
#include ”mex.h”

/*Do the actual computation*/
double doTheWork(double *A) {
   ...
}
void mexFunction(int nlhs,
 mxArray *plhs[], int nrhs,
 const mxArray *prhs[]) {
   ...

   /* Call the C function */
   out = doTheWork(pA);
   ...
}
```

mexFunction() – Purpose

- Validate the input and output, e.g., check:
  - number, type, and size of arguments
  - range of values and length of strings
- Allocate memory for output
- Preprocess the data
  - Type conversion
- Call the computational routine(s)
  - The computational routines can be part of the gateway routine (not recommended)
- Postprocess the data

mexFunction()

mexFunction(int nlhs, mxArray *plhs[],
             int nrhs, const mxArray *prhs[])

nlhs – The number of left-hand arguments
plhs – An array of left-hand output arguments
nrhs – The number of right-hand arguments
prhs – An array of right-hand input arguments

Example 1 – meanvec.c

```
/*
  m = meanvec(V) computes the mean m of the values in the vector V.
*/
#include ”mex.h”

#define MAX(A, B) ((A) > (B) ? (A) : (B))
#define MIN(A, B) ((A) < (B) ? (A) : (B))

/* Compute the mean of the values in a vector */
double vmean(double *vec, int length) {
   int i;
   double sum = 0;

   for(i = 0; i < length; i++) {
        sum += vec[i];
   }
   return sum/length;
}
```

meanvec.c (cont.)

```
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[]){
   double *V;
   double mean;
   size_t m,n;

   /* Check number of arguments */
   if (nrhs > 1) {
        mexErrMsgTxt("Too many input arguments.");
   } else if (nrhs < 1) {
        mexErrMsgTxt("Not enough input arguments.");
   } else if (nlhs > 1) {
        mexErrMsgTxt(”Too many output arguments”);
   }

   m = mxGetM(prhs[0]); /* Get number of rows */
   n = mxGetN(prhs[0]); /* Get number of columns */
```

meanvec.c (cont.)

```
   /* Check dimension and type of input */
   if (!mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
            MIN(m,n) != 1){
        mexErrMsgTxt(”V must be a non-empty real vector”); }

   V = mxGetPr(prhs[0]); /* Get pointer to start-address */

   /* Call computational routine */
   mean = vmean(V,MAX(m,n));

   /* Allocate memory for output argument and assign
      the mean value to it */
   plhs[0] = mxCreateDoubleScalar(mean);

   return;
} /* End of mexFunction */
```

```
>> mex meanvec.c

>> m = meanvec([1 2 3 4])
m =
    2.5000

>> meanvec([1 2 3 4])
ans =
    2.5000

>> [m,n] = meanvec([1 2 3 4])
??? Too many output arguments

>> m = meanvec([1 2 3 4],4)
??? Too many input arguments.

>> m = meanvec([1 2 3 4; 5 6 7 8])
??? Input argument must be a non-empty real vector

>>
```

Data types and macros

- mxArray (data type)
  - Matlab array
- mwSize
  - Array dimensions and number of elements
- mwIndex
  - Index values

mwSize and *mwIndex* are preprocessor macros included for cross-platform flexibility

Some specific mex functions

- mxCalloc
  - In a mex-file call mxCalloc to allocate memory (do not use calloc/malloc)
- mxFree
  - Free memory allocated with mxCalloc
- mexPrintf
  - ...as printf

Error handling

- mexWarnMsgTxt issue a Matlab warning

- mexErrMsgTxt issue a Matlab error and return to Matlab. Allocated memory is freed before termination

Matlab matrices in C

- Matlab stores (complex) matrices *column-vise* (C stores arrays row-vise) in two vectors: one contains the real data and one contains the imaginary data
- Pointer to each vector is obtained with *mxGetPr* and *mxGetPi*, respectively

... matrices

- mxCreateDoubleMatrix
  - □ Creates a 2-D matrix of type mxArray
- mxCreateSparse
  - □ Create a 2-D sparse matrix
- mxCreateNumericalArray
  - □ Creates an N-D matrix
- ... etc

... matrices

- mxGetPr
  - □ Get pointer to first real element in an array
- mxGetPi
  - □ Get pointer to first complex element in an array
- mxGetM / mxGetN
  - □ Get number of rows/columns of an array
- mxDestroyArray
  - □ Free memory allocated with mxCreate*

Example 2 – lapackLU.c

- This example will use the routine *dgetrf* in the LAPACK library to compute the LU-factorization of a real matrix
- For syntax of the computational Fortran routine in LAPACK, see

  http://www.netlib.org/lapack/double/dgetrf.f

lapackLU.c (cont.)

- To compile the source code the LAPACK library need to be included

```
>> mex –largeArrayDims lapackLU.c -lmwlapack
```

Support of 64-bit API (64-bit integers)

The LAPACK library is included with Matlab

lapackLU test

```
>> A = rand(5);
>> [L,U,P] = lapackLU(A)
L =
    1.0000         0         0         0         0
    0.3098    1.0000         0         0         0
    0.5177    0.5538    1.0000         0         0
    0.1236    0.0475    0.0414    1.0000         0
    0.4995    0.9573    0.9230   -0.6548    1.0000
U =
    0.7803    0.1320    0.2348    0.1690    0.5470
         0    0.9153    0.7485    0.6794    0.5753
         0         0   -0.5207    0.1840   -0.4128
         0         0         0    0.3902    0.6089
         0         0         0         0    0.2522
P =
     1     0     0     0     0
     0     0     0     0     1
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
>> norm(A-P*L*U)
ans = 1.2413e-16
```

Mex configuration

- The option file for mex is *mexopts.sh* allocated in (search order):
  - □ Current directory
  - □ *Matlabroot*/bin
  - □ The local preferences directory (returned by *prefdir*). On Linux, usually in ’*Your home directory*’/.matlab/*Matlabversion*/
- To configure mex (e.g., to select compiler) type ’*mex -setup*’ in Matlab

Mex options

`-v`	Verbose
`-I`*<path>*	Search for header files in *<path>*
`-l`*<library>*	Link with library *<library>*
`-L`*<path>*	Search for libraries in *<path>*
`-D`*<name>*	Define the symbol *<name>*
`-output` *<name>*	Write output to <name>.mex*

Makefile example

```
# Makefile for lapackLU.c
#
# Targets:
#    * lapackLU: Create the mex file lapackLU.mex*
#    * debug: Compile mex-file with debug
#             information and trace outputs
#    * clean: Remove all object files.
#             Needed before compiling for a new
#             architecture.
```

```
MEX = mex
MEXOPT = -O -largeArrayDims
MEXLIBS = -lmwlapack
MEXDEBUG = -g -DDEBUG

RM = rm -f

# Rules for targets
default: all

all : lapackLU clean

lapackLU:
	@echo Compile MEX-file
	$(MEX) $(MEXOPT) -output $@ lapackLU.c $(MEXLIBS)
	@echo Done..

debug:
	@echo Compile MEX-file for debugging
	$(MEX) $(MEXDEBUG) -output lapackLU lapackLU.c $(MEXLIBS)
	@echo Done..

clean:
	$(RM) *.o
```

For more information

- Go to Matlab help
  - Matlab → External Interfaces
- Or visit: http://www.mathworks.se/access/helpdesk/help/techdoc/matlab_external/bp_kqh7.html

Calling Java from Matlab

- Intergrated in Matlab (since Matlab 6)
- Can call native (built-in) as well as external Java classes directly from the Matlab command prompt
- The Java classes do not need to be compiled explicitly for Matlab
- No gateway function needed in the Java class

The other way around...

- Possible to call Matlab functions from an external program
  - UNIX: Through pipes
  - Windows: through a COM interface

⇨ Use Matlab as computation engine

Component Object Model

- *Component Object Model* (COM) – Framework for integrating existing software into an application
  - Software can be written in any language that supports COM
  - The applications communicate through a COM object

Calling Matlab from Java

- Is officially *not* supported by Mathworks
- However...
  - □ *jmi.jar* included to support this functionality
  - □ Third-party Java-Matlab interfaces exist

Compiled Matlab code

- Compile your Matlab code to stand-alone applications or software
  - □ Matlab is not needed by the end-user
- Requirements
  - □ *Matlab Compiler*
  - □ An ANSI C or C++ compiler
- Restrictions
  - □ Do not work with all toolboxes (or parts of)
  - □ Most pre-built GUIs can not be included