Using external libraries with MATLAB: MEX-files
Introduction

• Although MATLAB is a complete environment, it is often required to interact with external data or modify the way MATLAB does it
• MATLAB provides an interface to external programs written in C and Fortran via MEX-files
• MEX-files are dynamically linked subroutines that the MATLAB interpreter can automatically load and execute at runtime
• MEX-files have several applications:
  – Large pre-existing C and Fortran programs can be called from MATLAB without having to be rewritten as M-files
  – Bottleneck computations, e.g., for-loops, can be recoded in C or Fortran for efficiency
MEX-files

- are subroutines produced from C or Fortran source code
- behave just like M-files and built-in functions
- while M-files have a platform-independent extension, .m, MEX-files extensions are platform-specific, e.g.
  - Alpha \(\Rightarrow\) mexaxp
  - Linux \(\Rightarrow\) mexglx
  - Solaris \(\Rightarrow\) mексol
  - Windows \(\Rightarrow\) dll
- are called exactly as any M-functions, e.g.,
  - to invoke conv2 function, MATLAB will look for conv2.mex on Windows platform
mx and mex Prefixes

• mx Routines
  – Routines in the API prefixed with mx create/access/manipulate/destroy mxArrays
    • the array access and creation library provides a set of array access and creation routines for manipulating MATLAB arrays, e.g.,
      #include "matrix.h"
      double *mxGetPr(const mxArray *array_ptr);
      array_ptr – pointer to an mxArray.

• mex Routines
  – Routines prefixed with mex perform operations in the MATLAB environment
    • mex routines are only available in MEX-functions, e.g.,
      #include "mex.h"
      int mexEvalString(const char *command);
      command – a string containing the MATLAB command to execute
MATLAB Data

- MATLAB works with only a single object type: the MATLAB array
- All MATLAB variables, e.g., scalars, vectors, matrices, strings, cell arrays, etc., are stored as MATLAB arrays
- In C, the MATLAB array is declared to be of type mxArray
- The mxArray structure contains, among other things:
  - Its type
  - Its dimensions
  - The data associated with this array
  - If numeric, whether the variable is real or complex
  - If sparse, its indices and nonzero maximum elements
  - If a structure or object, the number of fields and field names
Data Storage

• All data is stored columnwise (Fortran convention)
  – Example:

```matlab
>> a=['house'; 'floor'; 'porch']
a =
house
floor
porch

>> size(a)
ans =
    3     5

and its data is stored as: h f p o l o u o r s o c e r h
Some Data Types

• Complex Double-Precision Matrices
  – The most common data type in MATLAB is the complex double-precision, nonsparse matrix. These matrices are of type double and have dimensions m-by-n, where m is the number of rows and n is the number of columns
  – The data is stored as two vectors (referred to by pointers) of double-precision numbers
    • pr – pointer to real data
    • pi – pointer to imaginary data
  – A real-only, double-precision matrix is one with pi=NULL

• Numeric Matrices
  – MATLAB also supports other types: single-precision floating-point and 8-, 16-, and 32-bit integers, both signed and unsigned
  – The data is stored in two vectors in the same manner as double-precision matrices

• MATLAB Strings
  – MATLAB strings are of type char
  – Stored the same way as unsigned 16-bit integers w/o imaginary data, each character in 16-bit ASCII Unicode, unlike C not null terminated
More Data Types

• Sparse Matrices
  – Have a different storage convention than full matrices
  – pr and pi are still arrays of double-precision numbers, but there are three additional parameters, nzmax, ir, and jc:
    • integer nzmax contains the length of ir, pr, and pi – the maximum possible number of nonzero elements in the sparse matrix
    • ir points to an integer array of length nzmax containing the row indices of the corresponding elements in pr and pi
    • jc points to an integer array of length N+1 that contains column index information
    • for j, in the range 0 \( j \) N-1, \( jc[j] \) is the index in ir and pr (and pi) of the first nonzero entry in the \( j^{th} \) column and \( jc[j+1] - 1 \) index of the last nonzero entry; \( jc[N] = nnz \)
    • if \( nnz < nzmax \), then more nonzero entries can be inserted in the array without allocating additional storage
Using Data Types

- The six fundamental data types in MATLAB are double, char, sparse, uint8, cell, and struct.
- You can write MEX-files, MAT-file applications, and engine applications in C that accept any data type supported by MATLAB.
- In Fortran, only the creation of double-precision n-by-m arrays and strings are supported.
  - Try:
    ```
    cd([matlabroot '/extern/examples/mex']);
    x = 2;
    explore(x);
    explore([1 2 3 4 5])
    explore(1 2 3 4 5)
    explore({1 2 3 4 5})
    explore(int8([1 2 3 4 5]))
    explore ({1 2 3 4 5})
    explore(sparse(eye(5)))
    explore(struct('name', 'Joe Jones', 'ext', 7332))
    explore(1, 2, 3, 4, 5)
    ```
Building MEX-Files

- MATLAB contains tools for API – lcc, but no Fortran
- If use your C compiler, must be ANSI C
  - In Windows, must be able to create DLLs
- MATLAB supports some compilers, provides preconfigured files, called options files, e.g.,
  - Windows:
    - Lcc C Compiler (bundled) ⇒ lccopts.bat
    - Microsoft Visual C++, V7.1 ⇒ msvc71opts.bat
    - Compaq Visual Fortran ⇒ df66engmatopts.bat
  - Unix/Linux:
    - System ANSI Compiler ⇒ mexopts.sh
    - GCC ⇒ gccopts.sh
Specifying an Options File

• You can use the -f option to specify an options file on either UNIX or Windows, e.g., use:
  `mex filename -f <optionsfile>`

• To configure, run `mex –setup`

• You may attempt to use your own compiler; a snapshot of `<optionsfile>` (from mexopts.bat or .sh on Linux)

```
rem ******************************************************
rem Compiler parameters
rem ******************************************************
set COMPILER=lcc
set COMPFLAGS=-c -Zp8 -l"%MATLAB%\sys\cc\include" -DMATLAB_MEX_FILE noregistrylookup
set OPTIMFLAGS=-DNDEBUG
...

rem ******************************************************
rem Linker parameters
rem ******************************************************
set LIBLOC=%MATLAB%\extern\lib\win32\cc
set LINKER=lcclnk
```
Testing Your Configuration

• By doing

```matlab
>> matlabroot
>> ans =
    C:\Program Files\MATLAB71
>> cd C:/Prog’ram Files’/MATLAB71/extern/examples/mex
>> mex yprime.c
>> yprime(1,1:4)
```
Creating C Language MEX-Files

void mexFunction(
    int nlhs, mxArray *plhs[],
    int nrhs, const mxArray *prhs[])
{
    /* more C code ... */
}

where
    nlhs – number of arguments on left-hand side,
    plhs – first element in the array of mxArray* on the left-hand side,
    similarly for nrhs/prhs
The Components of a C MEX-File

The source code for a MEX-file consists of two distinct (logical) parts:

- A computational routine that contains the code for performing the computations (numerical computations as well as input/output)

- A gateway routine that interfaces the computational routine with MATLAB by the entry point mexFunction and its parameters prhs, nrhs, plhs, nlhs

  - In the gateway routine, you can access the data in the mxArray structure and then manipulate this data in your C computational subroutine, e.g., the expression

    ```c
    mxArray *prhs[0];
    ...
    mxArray *result;
    ...
    result = mxCreateDoubleMatrix(3, 3, mxREAL);
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            result = mxSetPr(matrix, i * 3 + j);
            *result = i * j;
        }
    }
    ```

    returns a pointer of type double * to the real data in the mxArray pointed to by prhs[0]
Examples of C MEX-Files: Passing a Scalar

- **Computational module**
  ```c
  #include <math.h>
  void timestwo(double y[], double x[])
  {
    y[0] = 2.0*x[0];
    return;
  }
  ```

- **MEX-file**
  ```c
  #include "mex.h"
  void timestwo(double y[], double x[])
  {
    y[0] = 2.0*x[0];
  }
  ```

  ```c
  void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
  {
    double *x, *y;
    int mrows, ncols;
  }
  ```
/* Check for proper number of arguments. */
if (nrhs != 1) {
    mexErrMsgTxt("One input required.");
} else if (nlhs > 1) {
    mexErrMsgTxt("Too many output arguments");
}

/* The input must be a noncomplex scalar double. */
mrows = mxGetM(prhs[0]);
ncols = mxGetN(prhs[0]);
if (!mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
    !(mrows == 1 && ncols == 1)) {
    mexErrMsgTxt("Input must be a noncomplex scalar double.");
}

/* Create matrix for the return argument. */
plhs[0] = mxCreateDoubleMatrix(mrows, ncols, mxREAL);

/* Assign pointers to each input and output. */
x = mxGetPr(prhs[0]);
y = mxGetPr(plhs[0]);

/* Call the timestwo subroutine. */
timestwo(y, x);
Examples of C MEX-Files: Passing a Matrix

```c
void mexFunction(int nlhs, mxArray *plhs[], int nrhs,
    const mxArray *prhs[])
{
    int m,n,i;
    double *A, *B;

    /* check for arguments here, omitted */

    m = mxGetM(prhs[0]);
    n = mxGetN(prhs[0]);

    A = mxGetPr(prhs[0]);

    /* allocate the answer */
    plhs[0] = mxCreateDoubleMatrix(n, m, mxREAL);
    B = mxGetPr(plhs[0]);
    for(i=0;i<m*n;i++) *(B+i) = *(A+i)*2.0;
}
```
More Features

- You can also
  - link multiple files
    
    ```
    >> mex circle.c square.obj rectangle.c shapes.lib
    ```
  - call MATLAB functions, e.g.,
    ```
    mexCallMATLAB(1, lhs, 1, rhs, "sin");
    mexCallMATLAB(0, NULL, 1, lhs, "plot");
    ```
  - clean up allocated memory
    ```
    mxDestroyArray(rhs[0]);
    mxDestroyArray(lhs[0]);
    ```
  - and so on, see documentation
Caveats

• On sparse matrix handling
  – in MATLABR Release 2006b, new types are introduced to handle sparse matrices to accommodate *large* matrices

• Argument passing
  – when returning *large* LHS array, you pay the price for copying/memory allocation; might be better off by working with pointers on the RHS instead
    • in particular, this is where you might run into memory problems with MATLAB; especially, avoid nested .m functions, the argument are passed by value, not by reference
Using LAPACK and BLAS Functions

• Specifying the Function Name
  – When calling an LAPACK or BLAS function, some platforms require an underscore character following the function name in the call statement.
    • On the PC and HP platforms, use the function name alone, with no trailing underscore. For example, to call the LAPACK dgemm function, use
      
      $\text{dgemm}(\text{arg1, arg2, ..., argn})$;

      • On the LINUX, Solaris, and Macintosh platforms, add the underscore after the function name, e.g.,
      
      $\text{dgemm}_{}(\text{arg1, arg2, ..., argn})$;
#include "mex.h"

doctype{void mexFunction(int nlhs, mxArray *plhs[], int nrhs, mxArray *prhs[])
{
    double *A, *B, *C, one = 1.0, zero = 0.0;
    int m,n,p;
    char *chn = "N";

    A = mxGetPr(prhs[0]);
    B = mxGetPr(prhs[1]);
    m = mxGetM(prhs[0]);
    p = mxGetN(prhs[0]);
    n = mxGetN(prhs[1]);
    if (p != mxGetM(prhs[1])) {
        mexErrMsgTxt("Inner dimensions of matrix multiply do not match");
    }
    plhs[0] = mxCreateDoubleMatrix(m, n, mxREAL);
    C = mxGetPr(plhs[0]);
    /* Pass all arguments by reference */
    dgemm (chn, chn, &m, &n, &p, &one, A, &m, B, &p, &zero, C, &m);
}
Building the C MEX-File w. BLAS/LAPACK

• Building on the PC (Windows)
  – for C MEX-file on a PC platform, you need to explicitly specify a library file to link with, e.g.,
    • for Lcc compiler that ships with MATLAB:
      \[ \text{mex myCmexFile.c <matlab>/extern/lib/win32/lcc/libmwlapack.lib} \]
    • for Microsoft Visual C++:
      \[ \text{mex myCmexFile.c <matlab>/extern/lib/win32/microsoft/libmwlapack.lib} \]

• Building on Other Platforms
  – build MEX-file as you would any other C MEX-file
    \[ \text{mex myCmexFile.c} \]
Calling MATLAB from C and Fortran Programs

• You can reciprocate:
  – MATLAB engine library is a set of routines that allows you to call MATLAB from your own programs, employing MATLAB as a computation engine
  – MATLAB engine programs are C/Fortran programs that communicate with a separate MATLAB process via pipes, on UNIX, and through a Component Object Model (COM) interface, on Windows