

Lecture 7

Scientific Computing: Compiled MATLAB MEX Interface, MATLAB Coder

Matthew J. Zahr

CME 292
Advanced MATLAB for Scientific Computing
Stanford University

24th April 2014



1 Introduction

2 MEX Interface

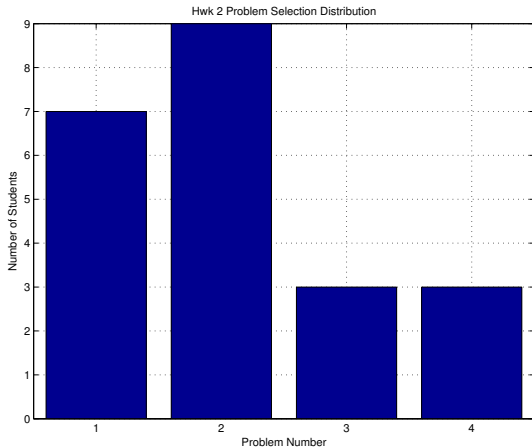
- Introduction
- Components of MEX-file
- C/C++ Matrix Library API
- Example: Bubble Sort

3 MATLAB Coder

- Background
- Demo: Bubble Sort



Homework 2 Problem Selection



Solutions posted!



Outline

- 1 Introduction
- 2 MEX Interface
 - Introduction
 - Components of MEX-file
 - C/C++ Matrix Library API
 - Example: Bubble Sort
- 3 MATLAB Coder
 - Background
 - Demo: Bubble Sort



Overview

- MATLAB Engine
 - The MATLAB engine library contains routines that allow you to call MATLAB software from programs written in other languages
 - Employs MATLAB as a computation engine
 - Possibility of shortened development time
- MATLAB executable, or MEX, files
 - Allows one to call a C/C++/Fortran program from within MATLAB (as if it were a MATLAB builtin function)
- MATLAB Coder
 - Generates standalone C/C++ code from MATLAB code



Outline

- 1 Introduction
- 2 MEX Interface
 - Introduction
 - Components of MEX-file
 - C/C++ Matrix Library API
 - Example: Bubble Sort
- 3 MATLAB Coder
 - Background
 - Demo: Bubble Sort



Motivation

MATLAB's MEX Interface allows

- Large existing C/C++/Fortran files to be called from within MATLAB without having to re-write them as M-files
- Speed up bottlenecks in a code (usually loop intensive computations) by writing them in low-level languages



Warning

The MEX interface also introduces complications to the highly-productive MATLAB environment

- Requires compilation
 - Setting up C/C++/Fortran compiler
 - Compiler must be compatible with version of MATLAB
- Portability between different systems may be an issue

The take-away message regarding MATLABs MEX-files

- Can be very useful for optimizing code or interfacing with large, existing packages
- Should only be used when necessary as they tend to decrease development productivity



MEX Files

- A MEX-file is the *interface* between MATLAB and the C/C++/Fortran program
 - Must know the inputs/outputs, data types, etc of *both* the MATLAB and C/C++/Fortran programs
- MEX-files are *dynamically-linked* subroutines that the MATLAB interpreter loads and executes
- A MEX-file contains only one function
- The name of the function in the MEX-file (as far as your MATLAB program is concerned) is the MEX-file name
- To call a MEX-file, use the name of the file, without the file extension



MEX Terminology

MEX Term	Definition
source MEX-file	C, C++, or Fortran source code file
binary MEX-file	Dynamically linked subroutine executed in the MATLAB environment
MEX function library	MATLAB C/C++ and Fortran API Reference library to perform operations in the MATLAB environment
mex build script	MATLAB function to create a binary file from a source file

http://www.mathworks.com/help/matlab/matlab_external/introducing-mex-files.html



What you will need

The remainder of this lecture will be assuming MEX-files are to be used to communicate with C/C++ code. To accomplish this, you will need:

- C/C++ source code
- A compiler supported by MATLAB
- C MEX Library
- mex build script



Hello World

hello.c (source MEX-file)

```
// From "Writing MATLAB C/MEX Code (Getreuer)
#include "mex.h" /* Always include this */

void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
    mexPrintf("Hello world!\n");
    return;
}
```

```
>> mex hello.c
>> hello
Hello world!
```



mexFunction Gateway Routine

- Entry point to a MEX-file
- Takes place of main function in source code
- The name of the file containing `mexFunction` is the name of the function to be called in MATLAB
 - source MEX-file
- File extension of binary MEX-file is platform-dependent
 - `mexext` to return MEX-file extension on current platform



mexFunction signature

In source MEX-file

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

Parameter	Description
prhs	Array of right-side input arguments
plhs	Array of left-side output arguments
nrhs	Number of right-side arguments, or the size of the prhs array
nlhs	Number of left-side arguments, or the size of the plhs array



mexFunction signature: Example

- Suppose we have a source MEX-file named `mymex.c`
- `[X, Y, Z] = mymex(a, b)`
 - `nlhs = 3`
 - `nrhs = 2`
 - `plhs[0]`, `plhs[1]`, `plhs[2]` - pointers to X, Y, Z
 - `prhs[0]`, `prhs[1]` - pointers to a, b



Computational Routine

- Code in low-level language implementing desired subroutine
- Can be located in source MEX-file or separate file
- When computational routine(s) separate from source MEX-file, be sure the source MEX-file is the first argument to `mex`
 - More on this later



MATLAB Array (`mxArray`)

MATLAB works with a single object: **MATLAB Array**

- All variables stored as MATLAB Arrays
- In C/C++, the MATLAB Array is declared as `mxArray`
- `mxArray` contains the following information about the array:
 - its type, dimensions, and data
 - if numeric, whether the data is real or complex
 - if sparse, nonzero elements and `nzmax`
 - if structure or object, the number of fields and field names



Data Flow

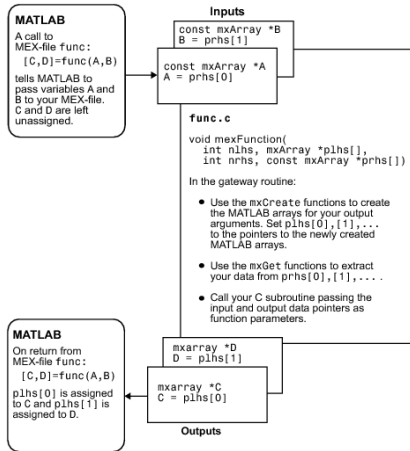


Figure : http://www.mathworks.com/help/matlab/matlab_external/data-flow-in-mex-files.html



Data Types

Command	Description
<code>mxArray</code>	Type for MATLAB
<code>mwSize</code>	Type for size values
<code>mwIndex</code>	Type for index values
<code>mwSignedIndex</code>	Signed integer type for size values
<code>mxChar</code>	Type for string array
<code>mxLogical</code>	Type for logical array
<code>mxClassID</code>	Enumerated value identifying class of array
<code>mxComplexity</code>	Flag specifying whether array has imaginary components

http://www.mathworks.com/help/matlab/data-types_bt12zvw-1.html



Create/Delete Array

Command	Description
<code>mxCreateDoubleMatrix</code>	2d, double array
<code>mxCreateDoubleScalar</code>	Scalar double
<code>mxCreateNumericMatrix</code>	2d numeric matrix
<code>mxCreateNumericArray</code>	Nd numeric matrix
<code>mxCreateString</code>	1-N array
<code>mxCreateCharArray</code>	Nd string array

<http://www.mathworks.com/help/matlab/create-or-delete-array.html>



Create/Delete Array

Command	Description
<code>mxCreateLogicalScalar</code>	Scalar, logical array
<code>mxCreateLogicalMatrix</code>	2d logical array
<code>mxCreateLogicalArray</code>	Nd logical array
<code>mxCreateSparse</code>	2d sparse array
<code>mxCreateStructMatrix</code>	2d structure array
<code>mxCreateStructArray</code>	Nd structure array



Create/Delete Array

Command	Description
<code>mxCreateCellMatrix</code>	2d cell array
<code>mxCreateCellArray</code>	Nd cell array
<code>mxDestroyArray</code>	Free dynamic memory
<code>mxDuplicateArray</code>	Make deep copy of array
<code>mxCalloc</code>	Allocate dynamic memory for array
<code>mxMalloc</code>	Allocate uninitialized dynamic memory
<code>mxRealloc</code>	Reallocate dynamic memory
<code>mxFree</code>	Free dynamic memory



Validate Data

Command	Description
<code>mxIsDouble</code>	Determine whether data is a double
<code>mxIsComplex</code>	Determine whether data is complex
<code>mxIsNumeric</code>	Determine whether input is numeric
<code>mxIsChar</code>	Determine whether input is string
<code>mxIsLogical</code>	Determine whether array is logical
<code>mxIsStruct</code>	Determine whether input is structure
<code>mxIsCell</code>	Determine whether input is cell
<code>mxIsInf</code>	Determine whether input is infinite
<code>mxIsSparse</code>	Determine whether input is sparse

[http:](http://www.mathworks.com/help/matlab/validate-data.html)

[//www.mathworks.com/help/matlab/validate-data.html](http://www.mathworks.com/help/matlab/validate-data.html)



Access Data - Matrices

Command	Description
mxGetDimensions	Number of dimensions in array
mxSetDimensions	Modify number of dimensions and size
mxGetM	Number of rows in array
mxSetM	Set number of rows in array
mxGetN	Number of columns in array
mxSetN	Set number of columns in array

<http://www.mathworks.com/help/matlab/access-data.html>



Access Data - Pointers

Command	Description
mxGetScalar	Real component of first data element
mxGetPr	Real data in double array
mxSetPr	Set real data in double array
mxGetPi	Imaginary data in double array
mxSetPi	Set imaginary data in double array
mxGetData	Pointer to real data in array
mxSetData	Set pointer to real data in array
mxGetImagData	Pointer to imaginary data in array
mxSetImagData	Set pointer to imaginary data in array



Access Data - Sparse matrices

Command	Description
<code>mxGetNzmax</code>	Number of elements in triplet
<code>mxSetNzmax</code>	Set storage space for nonzero elements
<code>mxGetIr</code>	<code>irow</code> of sparse matrix triplet
<code>mxSetIr</code>	Set <code>irow</code> of sparse matrix triplet
<code>mxGetJc</code>	<code>jcol</code> of sparse matrix triplet
<code>mxSetJc</code>	Set <code>jcol</code> of sparse matrix triplet



Bubble Sort

- You are given a source MEX-file, `bubble_sort_mex.c`, implementing bubble sort
- Use the code below to compile and test `bubble_sort_mex`

```
>> mex bubble_sort_mex.c  
>> v = rand(1000,1);  
>> tic; s= bubble_sort_mex(v); toc  
>> tic; sm = ...  
    bubble_sort(v,struct('order','ascend')); toc
```



Bubble Sort

```
#include "mex.h"
#include "matrix.h"

void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[]) {
    int i, M;
    bool swapped;
    double *v, *s, tmp;

    // Get size of vector (assume column)
    M = mxGetM(prhs[0]);
    // Get pointer to data of input
    v = mxGetPr(prhs[0]);

    // Get output and pointer to its data of output
    plhs[0] = mxCreateDoubleMatrix(M, 1, mxREAL);
    s = mxGetPr(plhs[0]);
```



Bubble Sort

```
// Copy input array to output array
for (i = 0; i < M; ++i) {s[i] = v[i];}

// Bubble sort algorithm
while ( true ) {
    swapped = false;
    for (i = 1; i < M; ++i) {
        if (s[i-1] > s[i]) {
            tmp = s[i-1];
            s[i-1] = s[i];
            s[i] = tmp;
            swapped = true;
        }
    }
    if (!swapped) { return; }
}
}
```



Outline

- 1 Introduction
- 2 MEX Interface
 - Introduction
 - Components of MEX-file
 - C/C++ Matrix Library API
 - Example: Bubble Sort
- 3 MATLAB Coder
 - Background
 - Demo: Bubble Sort



Introduction

- MATLAB Coder
 - Generate standalone C/C++ code from MATLAB code
 - Generate MEX functions from MATLAB code
- Available from
 - command line (codegen)
 - GUI (Project Interface)
- Requires MATLAB and host C compiler
- Works with Simulink and Embedded Coder



Considerations

When converting MATLAB code to C/C++ code, the following must be considered

- C/C++ use static variable types (MATLAB does not)
 - Before conversion can take place, all variables must be assigned a type
- Variable-sized arrays/matrices supported for code generation
- Memory
 - Static memory - declare size of all arrays at compile time
 - Faster
 - Usually requires more memory (must allocate for worst case)
 - Dynamic memory - size determined at runtime



Designing for Code Generation

- Only generate C/C++ code for functions (not scripts)
- Remove unsupported constructs from MATLAB code
 - List of supported functions [here](#)
 - Unsupported features
 - anonymous functions (functions without a file)
 - cell arrays
 - nested function
 - recursion
 - sparse matrices
 - `try/catch`
 - Complete list of supported/unsupported features [here](#)
- MATLAB Code Generation Readiness Tool
 - `coder.screener('filename')`



MATLAB Code Generation Readiness Tool

Code Generation Readiness - run_heatflow.m etc.

Summary Code Structure

Code Generation Readiness Score: 4

Requires some minor changes

Code generation tools may fail unless the issues listed below are fixed.

Unsupported MATLAB function calls - 23 invocations

View by: Calling function MATLAB function

Click on a function at left to see more details

Anonymous function handles - 7 occurrences

Anonymous function handles are currently unsupported for code generation. The following uses were found.

- create_mesh.m 6
- run_heatflow.m 1

Cell arrays - 4 occurrences

Cell arrays, used in `simpplot.m`, are currently unsupported for code generation.

Use of scripts - 2 occurrences

Only functions are supported for code generation. This code uses the following scripts.

- dsegment.mexmaci64 1
- run_heatflow.m 1

Workspace

Name	Value

Command History

```

size(T)
close all
clc
h = subplot(msh.p,msh.t);
set(h,'Facecolor','interp');
Tf = zeros(size(bcs.dbc_loc));
Tf(bcs.dbc_loc) = bcs.dbc_loc;
for i = 1:size(T,2)
    Tf(i,bcs.dbc_loc) = T(i,i);
end;
set(h,'Facevertcdata',Tf);
run_heatflow
profile viewer
clc
help coder
clc
coder.screener('build_mat...');
coder.screener('run_heatt...');
    
```

Geometry File defining the geometry of a NACA0015.

naca.m (MATLAB Function)

naca(bs,s)



MATLAB Code Generation Readiness Tool

The screenshot shows the MATLAB R2012b interface with the Code Generation Readiness tool open for the project 'run_heatflow.m'. The tool provides a 'Summary' and 'Code Structure' view. The 'Code Distribution' pie chart shows the following data:

File	Code Generation Readiness	Lines
run_heatflow.m	4	19
simpplot.m	4	60
surftri.m	5	17
heat_ode_der.m	4	3
get_temp_fem.m	4	32
tri_quassquad.m	5	15
tri_shapefun.m	5	3
time_integrate.m	6	21

The 'Call Tree' table below shows the same data:

File	Code Generation Readiness	Lines
run_heatflow.m	4	19
simpplot.m	4	60
surftri.m	5	17
heat_ode_der.m	4	3
get_temp_fem.m	4	32
tri_quassquad.m	5	15
tri_shapefun.m	5	3
time_integrate.m	6	21

The 'Code Distribution' pie chart legend indicates:

- Requires very extensive changes (Red)
- Requires some minor changes (Green)
- Suitable now or with minimal changes (Dark Green)

The 'Call Tree' table includes a checkbox for 'Show MATLAB Functions'. The Command History window shows the following code:

```

size(T)
close all
clc
h = simplot(msh.p, msh.t);
set(h, 'BackgroundColor', 'interp');
Tf = zeros(size(bes.dbc_loc) + 1, 1);
for i = 1:size(T, 2)
    Tf(i-bes.dbc_loc) = T{:, i};
end
run_heatflow
profile viewer
clc
help coder
clc
coder.screener('build_mat...')
coder.screener('run_heatt...')
    
```



Bubble Sort

- Bubble sort is a simple sorting algorithm that repeatedly steps through a list, comparing adjacent items and swapping them if in the wrong order
 - Best case performance: $\mathcal{O}(N)$
 - Average case performance: $\mathcal{O}(N^2)$
 - Worst case performance: $\mathcal{O}(N^2)$
- Involves repeatedly looping through array and swapping (or not) adjacent items
 - Difficult to vectorize

We are going to use MATLAB Coder to convert `bubble_sort.m` to C/C++



Example: Code Generation Readiness Tool

```
coder.screener('bubble_sort')
```

The screenshot displays the MATLAB R2012b environment. The Command Window shows the execution of `coder.screener('bubble_sort')`. A dialog box titled "Code Generation Readiness - bubble_sort.m etc." is open, showing a green progress bar and the following text:

Code Generation Readiness Score: ██████████ 100
Suitable now or with minimal changes

This code is ready to use with code generation tools. A preliminary check found no issues. Some minor problems may be found if code generation is attempted, but if so, they should be easy to fix.

The Workspace window on the right shows the following variables and their values:

Name	Value
Y	<253x149 double>
TO	<253x1 double>
TF	<263x1 double>
bcs	<1x1 struct>
h	@0,0heat_ode_v...
func	174.6375
l	149
jac	<253x253 spars...
msh	<1x1 struct>
nstep	100
p	<263x2 double>
phys	<1x1 struct>
spars	<1x1 struct>
t	<431x3 double>
time	<149x1 double>
tspan	[0,1]

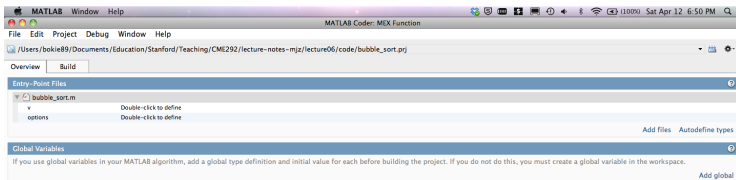
The Command History window shows the following commands:

```
IN2 = ID('');  
norm(LM-IN2)  
dbsql  
run_heatflow  
help axis  
run_heatflow  
save_str = [save_str,'h_',  
run_heatflow  
od ...  
od CME292/lecture-notes-04  
od lecture08/  
od code/  
od ...  
od ../lecture06/  
od code/  
od ...  
od code.screener('bubble_sort')
```



Example: MATLAB Coder - Initialize

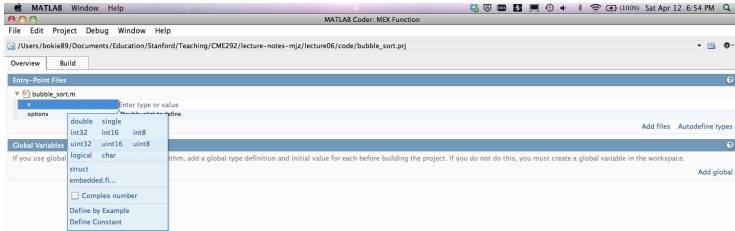
- `coder -new bubble_sort_coder.prj`
- Add `bubble_sort.m` to Entry-Point Files



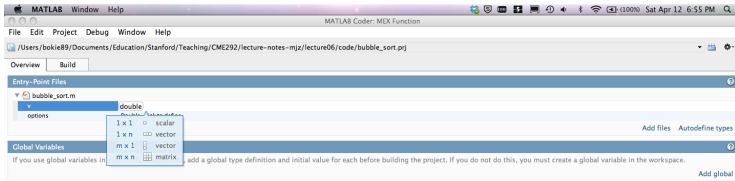
- Define both input arguments
 - `v` - double array of unknown length
 - `options` - structure with one field (`order`) that can be a string



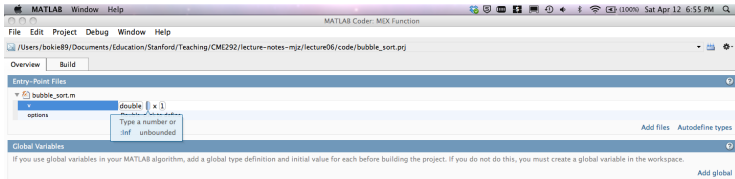
Example: MATLAB Coder - Define Inputs



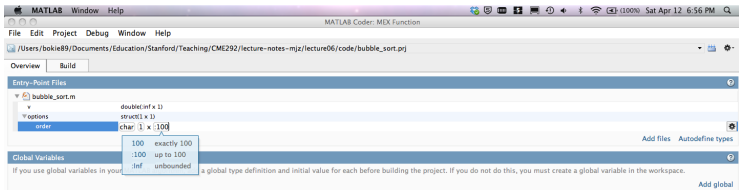
Example: MATLAB Coder - Define Inputs



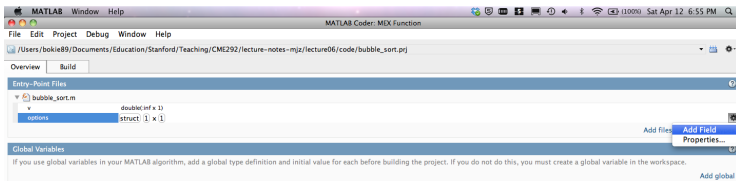
Example: MATLAB Coder - Define Inputs



Example: MATLAB Coder - Define Inputs

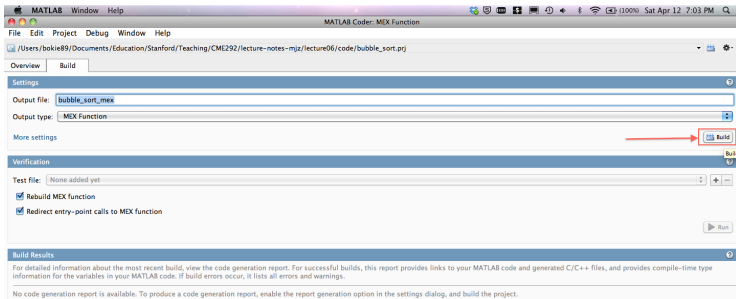


Example: MATLAB Coder - Define Inputs



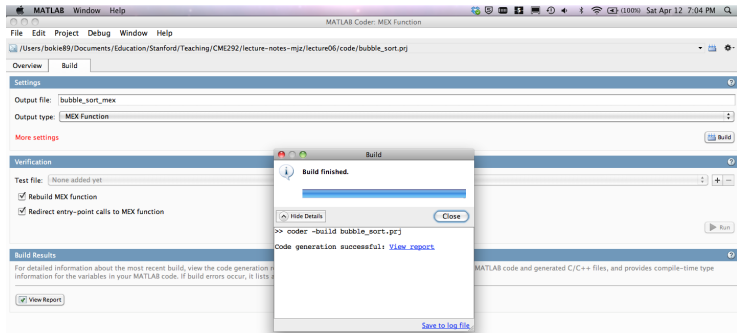
Example: MATLAB Coder - Build

• `coder -build bubble_sort.prj`



Example: MATLAB Coder - Build

- `coder -build bubble_sort.prj`



Test it!

- Run `bubble_sort_coder_test.m`
 - Verify outputs of MATLAB code and generated C/C++ code match
 - How do the timings compare?

