

Introduction to Mex with Matlab

Stefan Karlsson

November 2012

1 Introduction

This introductory exercise will start off the ph.d. course in ‘Efficient Matlab coding with visualization’. The topics covered here will aim at getting us started with some basic setup procedures for our future Matlab development

1.1 Matlab and parallel computing

Matlab is growing and improving¹ all the time. The two aspects of modern desktop computing that are most important for efficiency are multi-threading² and GPU computing³. Most Matlab functions now make full use of the multi-threading capabilities of the host machine, but support from the GPU is still very much in development.

The newest Matlab version is 8.0, and from an efficiency perspective, the biggest new addition is the [parallel toolbox](#).

Halmstad university does not have this toolbox or the latest version of Matlab yet. It has built in support for using clusters, multi-threading and GPUs. A small video introduction is found [here](#).

1.2 Different versions of Matlab

There are two matlab version for supported OSes: **Unix** and **Windows**. Strange to some, the windows version of Matlab is by far the better version of the two. As an example of this, look at the [matlab documentation for writing avi-files](#). See the option ‘compression’ on that page, and what options are available. For the Unix version, you can evidently not save video to anything else than a raw dump of the frames⁴. The documentation has quite a few of these little surprising differences, and I have never seen a case where Unix version has the upper hand on the windows version.

¹‘growing’ and ‘improving’ are not at all the same thing. There is now a jungle of toolboxes, and it can be a bit overwhelming for the newcomer to orient one-self

²most computers from laptops to stationary have at least dual core these days

³GPU stands for Graphical Processing Unit. Its the very powerful hardware on a computers graphics card that makes 3D renderings possible without overloading the CPU

⁴You can save movies to mpeg under unix, using [videoWriter](#)

There is NO NATIVE support for Mac, which means that running Matlab under Mac is especially nasty. One is forced to run the Unix version of matlab in the Mac implementation of x11 environment. This means a bad version of Matlab, on an emulator environment on the Mac. As a result, if you have Matlab on a Mac, expect extremely slow execution, problems with saving your results, and (in my experience) regular crashes.

There are also some practical pros and cons between the 32 bit and 64 bit versions of Matlab which we will come to later when dealing with large data sets and mex interfacing.

2 Setting Up Matlab

This will describe how to setup for a windows machine. Linux and Mac have the same command line arguments so its straightforward to make the analogous constructs to the windows shortcuts mentioned below. Regarding compilers, for Linux you will need the gcc compiler and for windows you need visual studio 2010.

2.1 Setting up Mex

For Mex to work stable and fast, you should get the same compiler used to compile the Matlab application. For windows thats visual studio and for Unix its gcc. Once you have started Matlab, type

```
mex -setup
```

The installed compiler should be listed among the available compilers, choose it (visual studio for win, gcc for unix). To test that it works, put the provided 'sillyFirstMex.cpp' into the current folder, and build it into an executable by typing:

```
mex sillyFirstMex.cpp
```

which should execute without error. For more help on a command, as always, type:

```
doc mex
```

after you compiled sillyFirstMex.cpp, you should now have an executable in your current folder. Depending your system, this will be called differently. For my system, its called 'sillyFirstMex.mexw64'⁵. After a function is compiled like this, it can be called like any other function: from the matlab command line or in a function or script. You can look inside of sillyFirstMex.cpp, by typing

```
edit sillyFirstMex.cpp
```

⁵for a win64 system, there is also unix OS support and 32 bit support making 4 different kinds of possible builds

notice that the matlab editor correctly highlights the syntax for this file, because the .c or .cpp ending. While its possible to develop mex files this way in Matlab, it is a nightmare to debug. Therefore, we are going to make use of visual studio for developing future mex files.

2.2 Visual Studio

Now, open sillyFirstMex.cpp in VS. It is possible to set up VS so that it will correctly build mex files, which you then still need matlab in order to run. We wont do that, but will for now just view the c-files here, and compile them from matlab like we did previously. The problem with matlab development is that we have no way of debugging the executable for runtime errors. This is the benefit with VS.

In VS, modify your function to contain the lines:

```
size_t m,n;
mexPrintf("\nHiya World!\n");

m = mxGetM(prhs[0]);
n = mxGetN(prhs[0]);
mexPrintf("size of input is %i by %i\n",m,n);
```

This new version is designed to take the input of the user, and display its width and height. Go back to the matlab prompt, build the project as before and call it with some matrix input:

```
mex sillyFirstMex.cpp
sillyFirstMex(ones(4,8));
```

Voila! As we expect. Now, lets try that again, but without sending an argument to the mex function

```
sillyFirstMex();
```

Disaster, Matlab crashes. No helpful information on where or what kind of error. This cartoon example is just to illustrate how easily critical errors occur in mex programming. There is no way of debugging source files of this kind in Matlab.

2.2.1 debugging in visual studio

- Build the mex file using the -g switch as

```
mex -g sillyFirstMex.cpp
```

- Install the 'MatlabDebug.vb' macro, with menu-bar button (instructions are found inside the file)

- In VS, enter a break point on line 26⁶ (if you get warnings, ignore).
- Start Matlab monitoring through VS by clicking the new menu button you installed
- Switch to Matlab, run the mex executable:

```
sillyFirstMex(ones(4,8));
```

At this point, you will find yourself in VS debug mode. Matlab main application is not responsive (as it should be) and you have the power to step through the execution, line by line and inspect the variables. Run again 'sillyFirstMex' without arguments (where Matlab crashes). You will now get an error message first in VS, and you will find out where the execution fails. In more realistic projects then this one, this will be very useful.

However, even now, Matlab crashes! This is a major drawback, and we have to model our work around this. Next session will be useful for this.

2.3 Command line shortcuts

For some hints on the non windows systems, see [here](#), otherwise this will be all windows version here.

Now, find your windows shortcut to start the matlab application. If you dont have one, simply make one from the executable. Copy this shortcut twice in the same folder, and name the copys 'Matlab nodesktop' and 'Matlab runscript'. Next right click the 'nodesktop' shortcut, properties, change target field by adding two command line options '-nosplash' and '-nodesktop'. On my system this will look like this:

```
"C:\R2011b\bin\matlab.exe" -nodesktop -nosplash
```

Running by 'Matlab nodesktop' will now start a thinner client. You will have access to all the Matlab functionality, except for the interactive desktop. This desktop takes huge amounts of resources, and a long time to start up.

Now, change the 'Matlab runscript' target to look like this:

```
"C:\R2011b\bin\matlab.exe" -nojvm -nosplash -r "callMyMex"
```

'-nojvm' will skip loading the java virtual machine. This will make matlab startup much faster(even thinner than nodesktop), but you will loose all but the most basic functionality (including editing m-files). The java machine handles many sub graphical subroutines, so this is not always a good idea to do. The reason we want a shortcut like this, is to be able to quickly run scripts that we suspect will crash matlab. The '-r' switch tells Matlab to run a script file, in this case it will look for a file called 'callMyMex.m' in the starting folder⁷. Make such a file and put the following in it:

⁶A quick way get to a line is CTR+G, to put a break point click the left margin of the editor

⁷If you dont know the starting folder, just start up Matlab, and see which folder it starts in

```
%content of callMyMex.m:
disp('Hello java-free world!');
```

```
disp('Hit any key to quit');
pause; exit;
```

This will make matlab start super thin, say hello and then quit. Usually, you will want your script file to look something more like this:

```
%Change to folder(where I have sillyFirstMex.cpp)
cd('C:\share\matlab\c-code');
```

```
%compile the mex
mex -g sillyFirstMex.cpp;
```

```
%now ask to be attached, wait for keypress
disp('click the Matlab button in Visual studio, then hit any key to continue');
pause;
```

```
% generate or load some proper args input:
inArg = ones(3,4);
```

```
%run the executable
sillyFirstMex(inArg);
```

```
% if you have output from the mex function, check
% that it is consistent by plotting it or likewise
```

```
disp('Hit any key to quit');
pause; exit;
```

The flow of your work will then be as such:

- Work on your c-code in VS
- Run your 'Matlab Runscript' when you want build and run your function.
- In VS you do the debugging with break points etc.

3 Task

As a simple task, change the function 'sillyFirstMex.cpp' so that it doesnt crash. It should output the dimensions of ALL the input args.

A good script file to test and debug (called in 'Matlab Runscript') is the following.

```
cd('C:\share\matlab\matlab course\source');
mex -g sillyFirstMex.cpp;

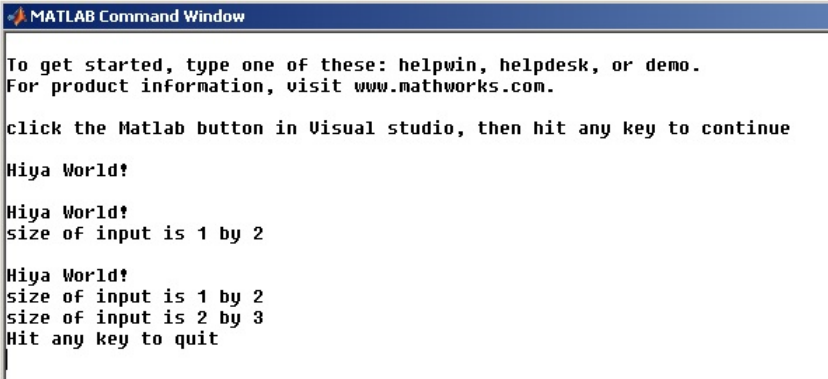
disp('click the Matlab button in Visual studio, then hit any key to continue');
pause;

inArg1 = ones(1,2);
inArg2 = ones(2,3);

sillyFirstMex()
pause(0.01);
sillyFirstMex(inArg1)
pause(0.01);
sillyFirstMex(inArg1,inArg2)
pause(0.01);

disp('Hit any key to quit');
pause;exit;
```

Output in the command window should look like figure 1

A screenshot of the MATLAB Command Window. The window title is "MATLAB Command Window". The text inside the window is as follows:

```
To get started, type one of these: helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.

click the Matlab button in Visual studio, then hit any key to continue

Hiya World!

Hiya World!
size of input is 1 by 2

Hiya World!
size of input is 1 by 2
size of input is 2 by 3
Hit any key to quit
```

Figure 1: correct output