Compiling MATLAB Executables

Abhimanyu Dubey, Virginia Tech

0 Introduction

This tutorial will provide instructions to compile MATLAB executables from MATLAB scripts.

The tutorial scripts have been tested on the object detection code from http://www.cs.berkeley.edu/~rbg/latent/.

1 MATLAB Compiler Runtime

The MATLAB Compiler Runtime (MCR) is a standalone set of shared libraries that enables the execution of compiled MATLAB applications or components on computers that do not have MATLAB installed. When used together, MATLAB, MATLAB Compiler, and the MCR enable you to create and distribute numerical applications or software components quickly and securely.

To run MATLAB applications, you would be required to have MCR installed on the target computer. Since as a developer you would also be testing your executables frequently before sending them to the target machine, it is a good idea to download the MCR and install it on your source computer as well. The download link is available at this link: http://www.mathworks.com/products/compiler/mcr/.

Tip: The easiest way to install is to run the install script in a GUI. If you are SSHing to a remote server and installing it there, use ssh –X to run the GUI.

2 Using MCC to Compile MATLAB Scripts

After you have completed installing the MCR, the next step is to write a wrapper script for the function/script you wish to call and compile that using the MATLAB compiler.

2.1 Writing a Wrapper Function

To compile your MATLAB script into an executable, you must keep some things in mind to avoid conflicts. The safest way to go about this would be to write a simple wrapper function to evaluate your code, keeping in mind a few points:

1. For a MATLAB function that returns a vector/matrix, the only way to interact with the outputted data is to store it in a file. The executables **do not** return any value.
2. Since the MATLAB compiler adds paths during compile time, there can be path conflicts if there are `addpath` calls present in your scripts. To prevent this from happening, you should surround all the `addpath` calls with an `~isdeployed` condition. For example,

```
if(~isdeployed)

        addpath('path to be added');

end
```

   This would prevent conflicts between the two path calls.

3. You can pass parameters to the executable function, and however, to avoid parsing errors, it is better to pass all arguments as strings at execution time (enclosed in double commas) and parse them to numbers, floats etc. in the wrapper function and pass to the main function.
4. If your main function requires a matrix/vector set as input, you can save the matrix/vector set to a .mat file and load the .mat file in the wrapper function, with its location being a parameter you pass to the wrapper function.

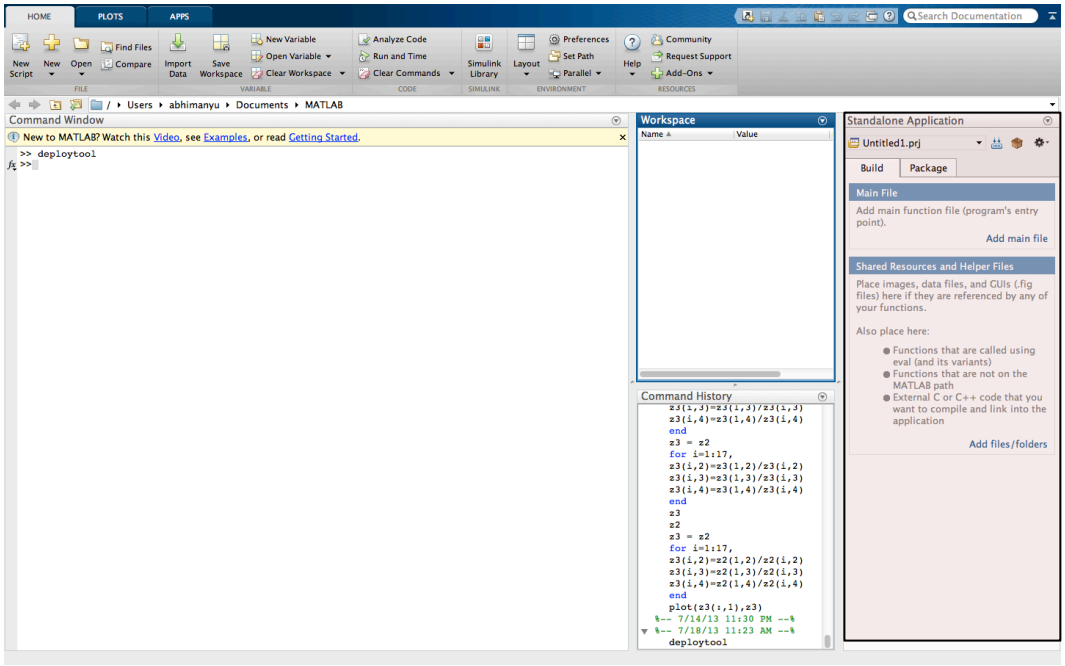
A sample wrapper function would look like this: (main function is main(x))

```
function wrapper (samplestring, sampleinteger) %no output type

        sampleinteger = str2num(sampleinteger); %converting string to number

        input = load(samplestring); %load a .mat matrix

        output = main(input); %computation step, main function returns a matrix

        save(filename, 'output'); % save results to .mat file

end
```

Save this wrapper function and proceed to the next part of the tutorial.

2.2 Using the MATLAB GUI for Deployment

Start MATLAB and type `deploytool` to open the deployment tool. Create a new project and select 'Standalone Application' under Type.



A sidebar opens up in the right. Under the 'Main File' section, you will be required to add the main file for the deployment. In this field you will point to the wrapper function that you have created in the previous section.

MATLAB automatically finds the requisite MATLAB dependencies (functions/scripts/mex files etc. provided they are in your MATLAB path) and links the files. You don't have to add each dependent function to the project.

Although if your project requires you to evaluate external code (C/C++/Java etc.) or some functions which are not present in your MATLAB path, you can do so by adding them under the 'Shared Resources' tab.

After you have added all the required dependencies, you can click the 'Build' button to compile and build your executable. It usually takes some time (~15-20 minutes) to build the executable.

After the build is completed, you can navigate to `<project_name>/distrib/` to view the executable.

2.3 Compiling using the Command Line Interface

To compile using MATLAB's command line interface, you can type

```
mcc –v <nameofwrapperfunction>
```

-v: verbose mode.

This command will compile wrapper.m along with all the MATLAB dependencies it can link to. To attach other files, add ' –a <path/to/file>' for each file in the mcc command. For other options, add each of the options you wish to after a –w.

As an example:

```
mcc -o Untitled1 -W main:Untitled1 -T link:exe -d
/Users/abhimanyu/Documents/MATLAB/Untitled1/src -w enable:specified_file_mismatch
-w enable:repeated_file -w enable:switch_ignored -w enable:missing_lib_sentinel -w
enable:demo_license -v /Volumes/Godel/sample.m –a /Volumes/Attach/Attachedfile.cpp
```

You can type `mcc help` in the MATLAB terminal for more details.

2.4 Running and Debugging Deployed Applications

To run a deployed application, navigate to `<project_name>/distrib/` and here you will find two files, one being the application itself and the other being a shell script beginning with `run_`. To run the application we must first provide the path to the MCR (as discussed in section 1) and this is done by the shell script mentioned above.

Thus, to execute your application type

```
./run_<applicationname>.sh <path-to-MCR> <variables, each separated by a space and
in double inverted commas, so that they are assumed strings>

Example, ./run_sample.sh ~/MCR/v81/ “these” “are” “variables” “1”
```

The deployed application takes a little while to load linked libraries, during which it does not prompt or display anything.

Debugging:

Debugging can be done via monitoring the terminal output, and the executable will store its temporary files in a folder known as the MCR Cache, located at `~/.mcrCache8.1` (latest version is 8.1, it may differ). All scripts and functions that are required by the executable are extracted here and then accessed, and temporary outputs are stored here.