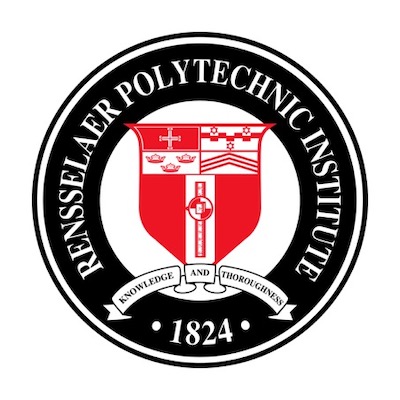
Merging Autolev (C) and MATLAB

Guide by David Hoddinott  
Project for Kurt Anderson  
December 2020



Contents

[Overview 3](#_Toc58575795)

[Requirements 3](#_Toc58575796)

[Setup 3](#_Toc58575797)

[Run your Autolev code 3](#_Toc58575798)

[Modify Autolev code for MEX conversion 3](#_Toc58575799)

[Creating a MEX function from auto generated C code 4](#_Toc58575800)

[Overview: 4](#_Toc58575801)

[Modification: 4](#_Toc58575802)

[Headers 5](#_Toc58575803)

[Main Function Declaration 5](#_Toc58575804)

[Declaring Inputs and Outputs 5](#_Toc58575805)

[Notes on printf and exit(0); 6](#_Toc58575806)

[Replacing inputs 7](#_Toc58575807)

[Replacing outputs 8](#_Toc58575808)

[Final words on converting from C to MEX 10](#_Toc58575809)

[Using MATLAB to run MEX code 10](#_Toc58575810)

[MATLAB setup 10](#_Toc58575811)

[Simple example 11](#_Toc58575812)

[Basic implementation notes 12](#_Toc58575813)

[Simulink 13](#_Toc58575814)

[Introduction 13](#_Toc58575815)

[MEX and MATLAB Interpreted Functions 13](#_Toc58575816)

[Basic Example 13](#_Toc58575817)

[Conclusion 15](#_Toc58575818)

# Overview

Autolev is a powerful tool for the computation of multibody dynamics for systems that are otherwise too complex to be solved for simply by hand, or by writing code for from the ground up. Autolev, though powerful, lacks some of the built in features that MATLAB offers. As these can be very useful to modeling systems, it would be advantageous for users to be able to merge these two software packages. Currently, Autolev does not offer MATLAB support as a built in capability. This guide is meant to help a user connect Autolev code to MATLAB via the use of MEX functions. This will work for most C codes, but the examples used in this guide will be specifically for Autolev auto generated code.

# Requirements

The following requirements are needed by the user in to successfully connect Autolev Code to MATLAB.

* Autolev, and working Autolev code
* MATLAB (This guide was developed for 2020a, 2020b and 2017b. Other versions may have additional undiscovered issues)
  + A MATLAB C compiler. MinGW64 works well (install via MATLAB’s add on explorer)
* Knowledge of how to code in MATLAB. Knowledge of SIMULINK can also be helpful
* Basic Knowledge of C or programming languages similar to C (Object oriented languages like Java, or even Python will be enough)

# Setup

## Run your Autolev code

Once the user has a complete Autolev code, it is advisable to test as much as possible now. Mistakes and errors in the base code will be far easier to debug when the code is in its most basic form, before the other layers of cross-platform complexity are applied. If you are running the auto-generated Autolev code with a language that is not C (Such as Fortran), apply the following command at the end of your code:



To code the dynamics in C, use a C compiler (such as Codeblocks IDE) to ensure that this still works as expected. Save a copy of your unmodified C code.

## Modify Autolev code for MEX conversion

All quantities that will be controlled/changed by MATLAB should be declared as a CONSTANT in Autolev. This includes control torques, forces applied, etc. This will allow MATLAB to feed the changed value into Autolev. Re-test all of your C code. Ensure that the code functions as expected with constant inputs.

# Creating a MEX function from auto generated C code

MATLAB support pages for additional help with MEX:

<https://www.mathworks.com/help/matlab/call-mex-file-functions.html>

<https://www.mathworks.com/help/matlab/ref/mex.html>

## Overview:

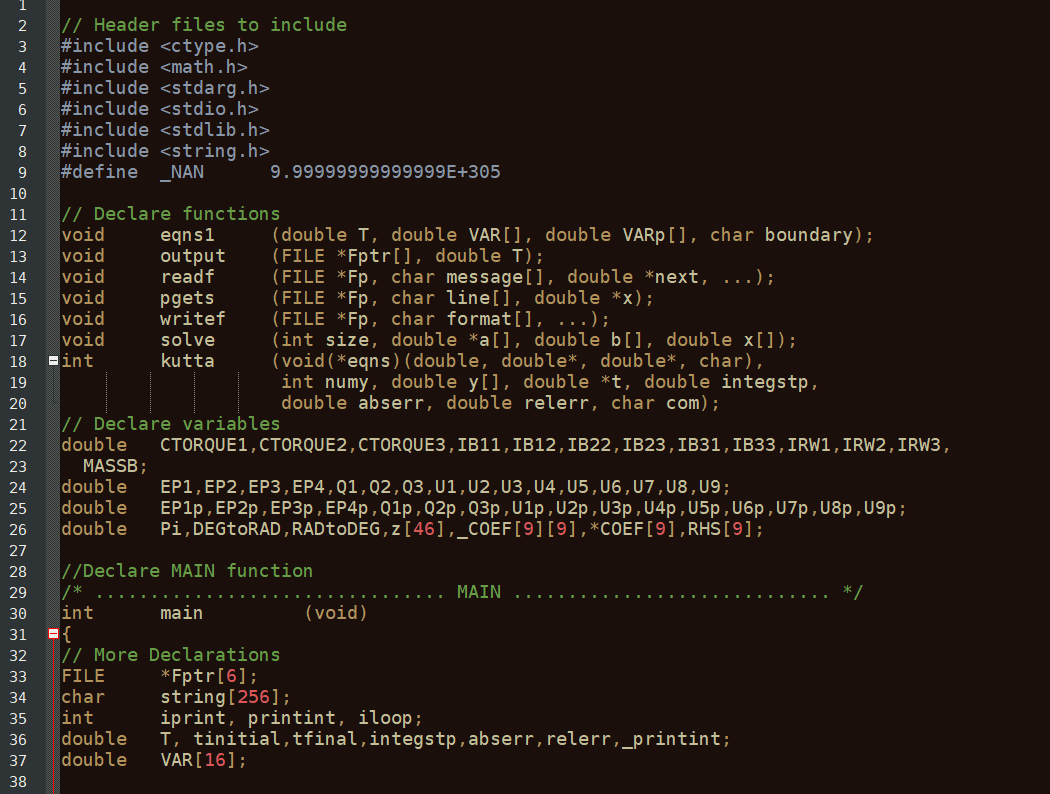
MEX stands for MATLAB Executable Function. This is a function written in slightly modified C that can be run in MATLAB just like a regular MATLAB function. It can take inputs, deliver outputs, write to the console, etc. The auto generated C code that Autolev returns can be simply modified into a MATLAB executable function, which will allow the code to be run in MATLAB, integrate into SIMULINK, etc.

A C code cannot be run as a MEX function without some modifications. Many of the commands generated by Autolev will cause errors or erroneous things occur. Due to the cross-platform nature of these two programs, there are often no error messages, or the messages provided are not clear. This guide is meant to help overcome some of these hurdles.

Before you begin to modify your own code, it can be advantageous to complete a simple example. A good place to start is to complete this tutorial by Shawn Lankton: <http://www.shawnlankton.com/2008/03/getting-started-with-mex-a-short-tutorial/>

## Modification:

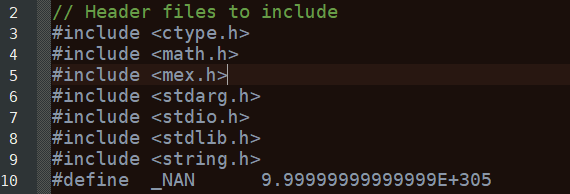
When Autolev generates code, the initial C code will look something like this:



Most of the modifications that are necessary will occur in the first 100 or so lines of code.

### Headers

The first thing to do is to include the MEX header function in the header declarations. This is what allows MATLAB to run all of the MEX specific commands (such as printing to the console, reading in variables, etc.).



### Main Function Declaration

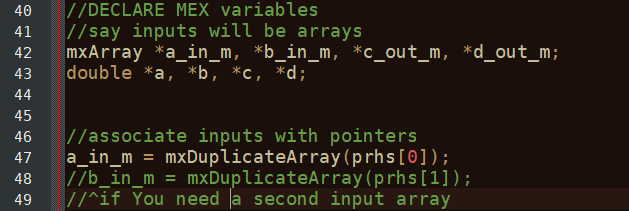
Next, the MAIN function declaration is modified. Instead of being a MAIN function, this is modified to be a mexFunction. The signature of the MEX function is always the same.



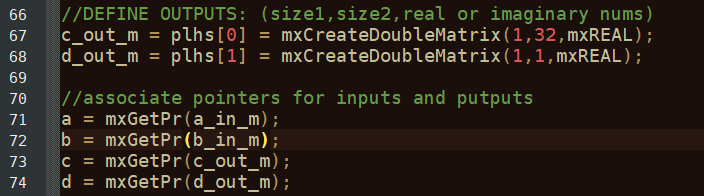
The mxArray plhr and prhs represent **P**ointers to the **L**eft **H**and **S**ide and **P**ointers to the **R**ight **H**and **S**ide. The Left hand side are outputs, and the right hand size are inputs

### Declaring Inputs and Outputs

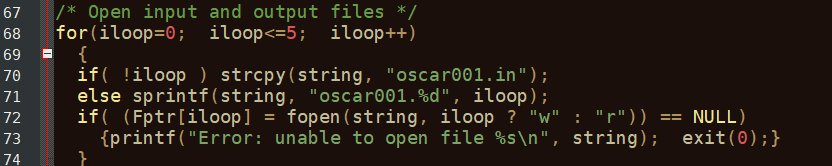
Now declare pointers for inputs



Then declare the outputs:

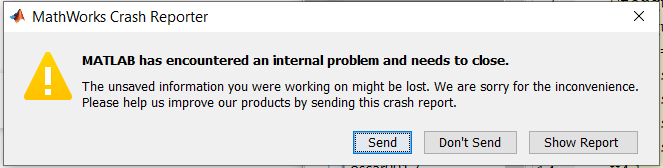


Now, you have declared the inputs and outputs. Locate where the C code opens the input and output files. Since the inputs and outputs are no longer files, this can be removed/commented out. In this example, this looks like this:



### Notes on printf and exit(0);

This is a good time to note some of the differences between MEX and C. In C, the command exit(0) will close the running C code, and the printf error message will be left in the open window, allowing the user to read it. In MEX, the command exit(0) will attempt to close the program it is running in (which is MATLAB). MATLAB is not designed to be closed in such a way. In earlier versions of MATLAB this will cause MATLAB to crash completely, while in later versions (at the time of writing, this is 2020b) it will give an error that looks like below, but will not crash.

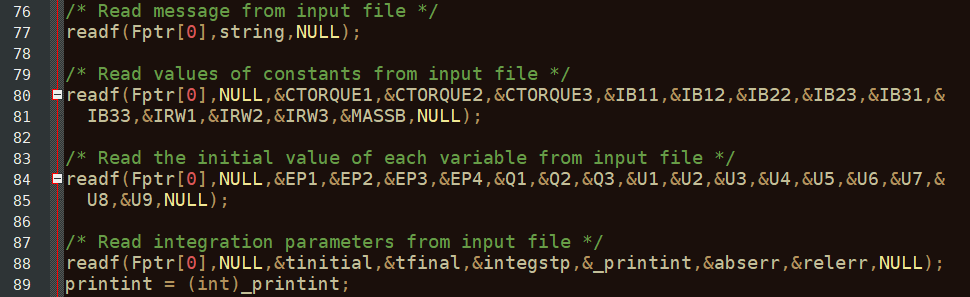


As such, all instances of exit(0) **MUST** be removed. When doing this, replace the print command with: mexPrintf(“error message”);

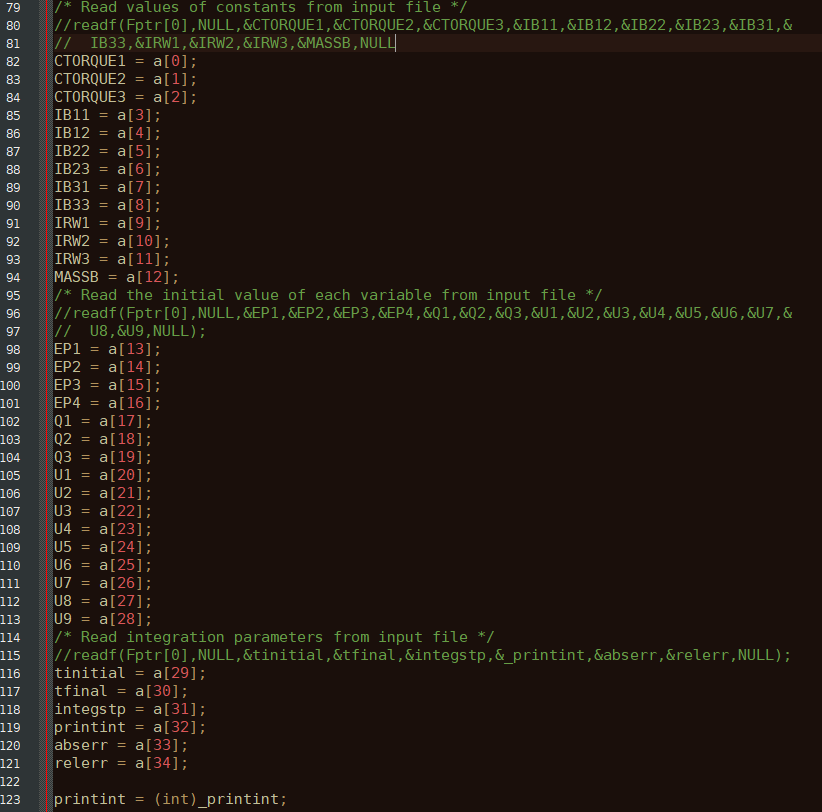
This prints to the MATLAB console, where it can easily be read by the user. Do this for all instances where the message should be printed to the console.

### Replacing inputs

Now, locate where the C code reads the input file. In this example, this looks like:

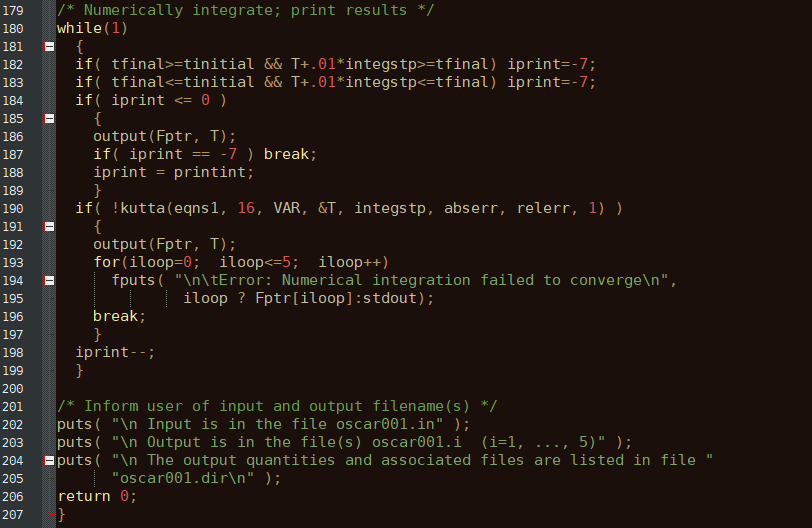


These are then replaced with the inputs that we have declared earlier. In this example, the readf functions are removed/commented out, as their inclusion will only slow down the function.

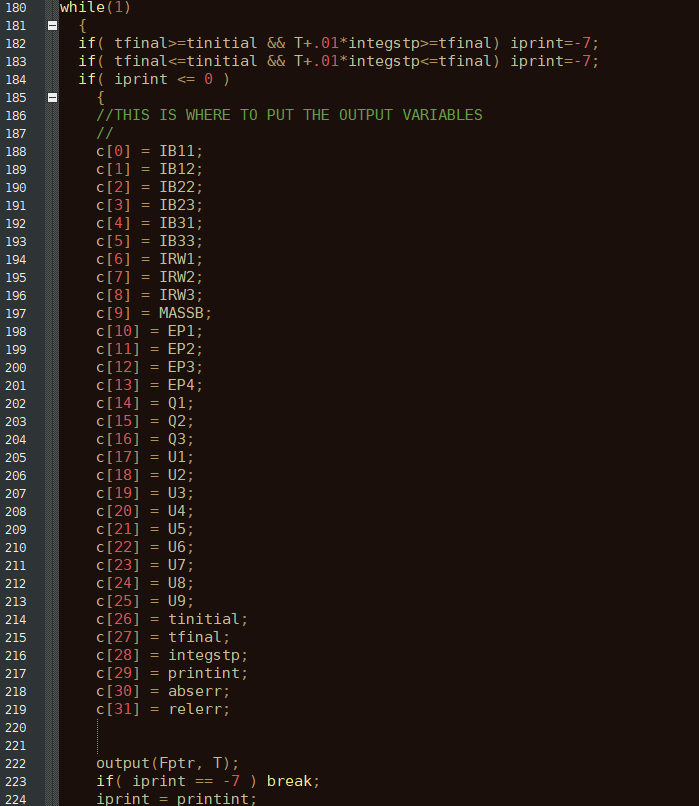


### Replacing outputs

Next, the outputs will be modified so they report to MATLAB rather than printing to an output file. There will be a section that has the comment “print result”. In this example, this looks as follows:



Add the code below before the output command. This assigns values to the output variables. Note: you will have to select the variable names based on what you want your specific code to output.



Finally, remove the “Return 0;” command from the last line of the function, as the mexFunction does not return anything.



### Final words on converting from C to MEX

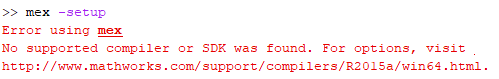
MATLAB will not run some C commands properly. As mentioned earlier, one of these is exit(0). Another is the print function. The standard printf function must be replaced with the MEX command mexPrintf, which will print to the MATLAB console everywhere that the program wishes to print something for the user. These two are the more common and troublesome in auto generated C code from Autolev, but it is not an exhaustive list. Check the MATLAB MEX documentation for a complete list.

# Using MATLAB to run MEX code

## MATLAB setup

To get started, MATLAB must be setup to run MEX code. This is done first by running MATLAB’s automatic setup via the command:

mex –setup

If you do not have a C compiler, you will get the error: 

If this occurs, install one. The recommended one is MinGW64 Compiler (C), which is available via the MATLAB add-on installer.



Next, with the modified C code in your current folder, use the command:

mex Filename.c

This creates the MEX function from the code you have modified. This will give errors if you have made any mistakes. Fix these according to the error messages.

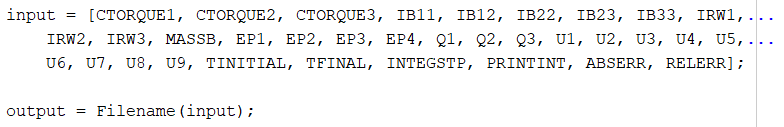
If the code is successful, you will see



You should see a new file appear in your current folder with the file extension “.mexw64”. This is the actual MEX file MATLAB will run.

## Simple example

You can now run a simple instance of your code.

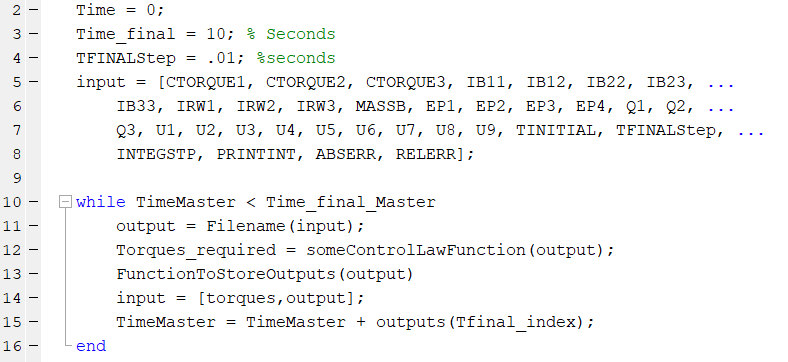


If all goes well, the variable output will contain all of the outputs in the form of an array, and should be the same as if you ran your original C code. Check the two against each other. Do they match? If they do not, check the MEX modification that you have written and see if you have made an error.

## Basic implementation notes

Once fully configured the MEX function can now be run just like any other MATLAB function. One example of how this could be advantageous is that one can now use built-in MATLAB functions that would otherwise be difficult to write in C, such as linear quadratic regulators, numerical integrators and differentiators, etc.

Below is an example of *pseudocode* running the code used above in a simple example of a control loop, where the control law is defined by the function someControlLawFunction.



After setting up the initial inputs, as well as the initial and final times, the code is run as normal, except the input to the MEX function has a Tfinal as something quite small, on the magnitude of hundredths of a second. The control law is calculated, the outputs are stored, and the outputs of the MEX function and the calculated control torques are set as the inputs for the next iteration of the MEX function, continuing until the desired end time is reached. The time step must be small, as the inputs will remain constant over it.

This is reasonably representative of most systems, as the input thruster, momentum wheel, etc. can only updated so fast. It also allows for convenient execution of the script. As the MEX function will be run many times, it can be advantageous to strip it down to its bare components, and remove unnecessary function calls (such as opening and closing files, writing to outputs as well as to the MEX outputs, etc.) in order to optimize the amount of computational time this will require.

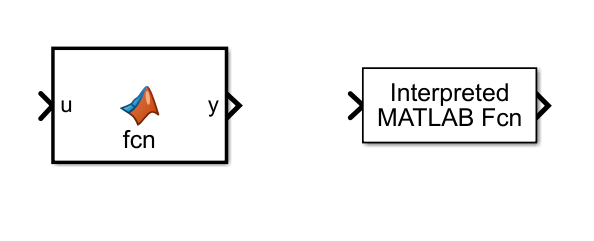
## Simulink

### Introduction

One of the more powerful MATLAB add-ons is Simulink, which allows users to create diagrams of code, and is taught in many RPI classes as the tool to use for controlling systems. SIMULINK is a diagram-like program, where variables traverse from one block to another.

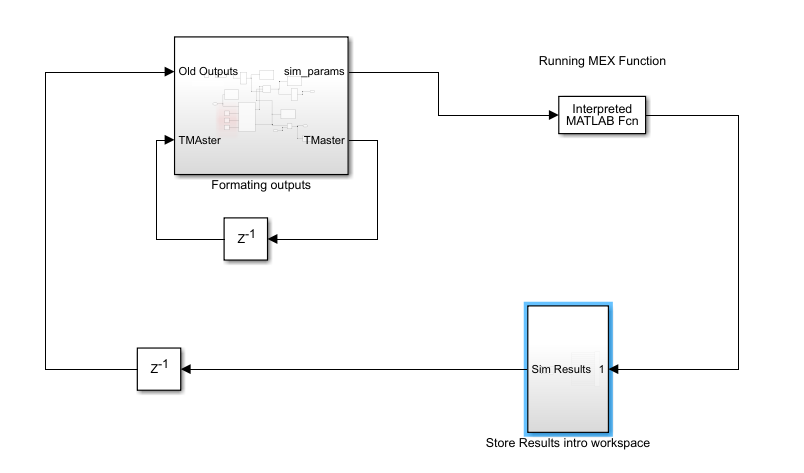
### MEX and MATLAB Interpreted Functions

In order to implement MEX files in SIMULINK, a minor modification must be made. The MEX function cannot be run in a standard MATLAB function block, as it is not supported in current versions of SIMULINK (although this may change in later versions). Instead am interpreted MATLAB function block must be used. They function quite similarly, except the Interpreted MATLAB function block runs code that is saved in a .m file specified, rather than a file saved in the Simulink simulation. Additionally, only one output/input can be made (although these can be arrays/matrices).

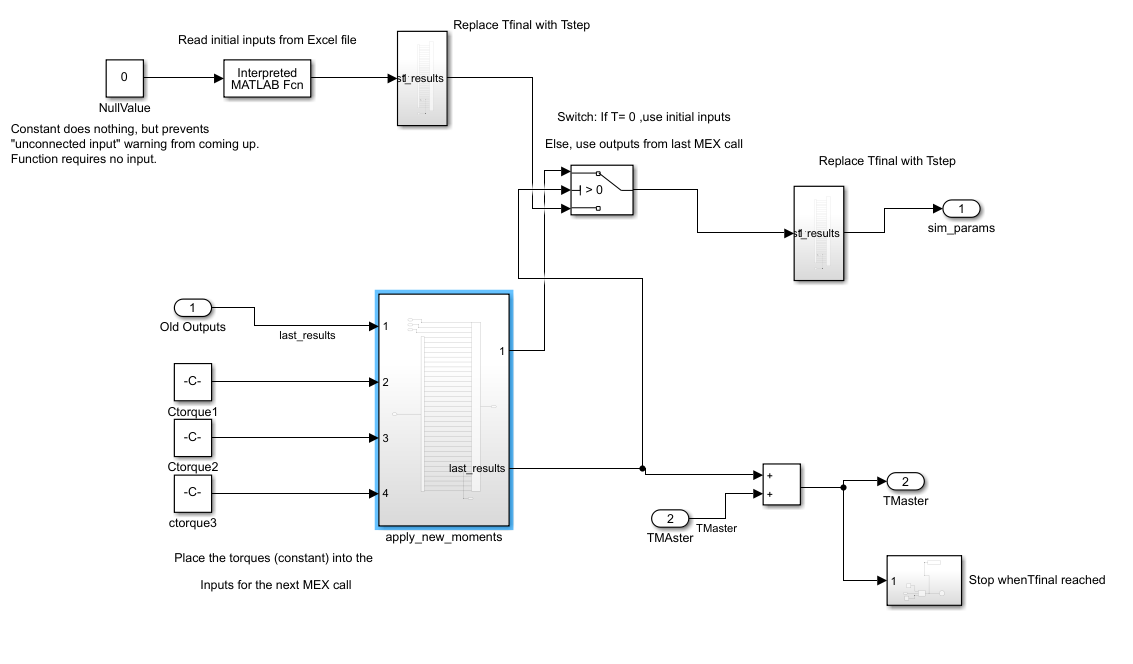


### Basic Example

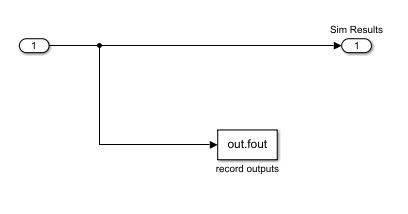
A basic example of running a MEX function in Simulink without a control law can be seen below:



Inside the Formatting outputs subsystem is the initial setup and formatting the outputs from the MEX function into the inputs for the next iteration.



The Store Results subsystem simply stores the results to the workspace for plotting later:



This SIMULINK diagram will work the same as the sample MATLAB function above. A control law could be implemented to vary the control torques.

# Conclusion

Both Autolev and MATLAB are powerful tools for computation. By mastering the ability to combine the two, they can both be used for their respective strengths, and create more complex system with less effort by the user. Once one has done several conversions, they are quick to implement, and can provide a simple way of running complex multibody code in MATLAB.