< Subset Selection and Regularization (http://blogs.mathworks.com/loren/2011/11/21/subset-selection-and-regularization/)

|

Managing Deployed Application Output with... > (http://blogs.mathworks.com/loren/2011/11/04/managing-deployed-application-output-with-message-handlers/)

Generating C Code from Your MATLAB Algorithms (http://blogs.mathworks.com/loren/2011/11/14/generating-c-code-from-your-matlab-algorithms/)

Posted by **Loren Shure**, November 14, 2011

I am pleased to introduce guest blogger Arvind Ananthan. Arvind is the Product Marketing Manager for MATLAB Coder (http://www.mathworks.com/products/matlab-coder/) and Fixed-Point Toolbox (http://www.mathworks.com/products/fixed/). His main focus in this post is to introduce basics of MATLAB Coder, talk about the workflow, its use cases, and show examples of generated C code.

Contents

A Brief History of MATLAB to C

In April, 2011, MathWorks introduced MATLAB Coder as a stand-alone product to generate C code from MATLAB code. This tool lets user generate readable, portable, and customizable C code from their MATLAB algorithms.

Many astute readers will notice that C code generation from MATLAB isn't really brand new - and that we've had this capability to generate C code from MATLAB for quite some time now. Yes, that's true - we've been incubating this technology for quite some time till we felt it was ready to debut as a stand alone tool. Here's a timeline of this technology over the past few years:

2004 - Introduced the Embedded MATLAB Function block in Simulink
2007 - Introduced the emlc function in Real-Time Workshop (now called Simulink Coder) to generate stand alone C from MATLAB
2011 - Released MATLAB Coder, the first stand-alone product from MathWorks to generate portable and readable C code from MATLAB code.

A Simple Example

Let me introduce the basics of using MATLAB Coder through a very simple example that multiplies two variables. Let's generate C code from the following MATLAB function that multiplies two inputs:

```
dbtype simpleProduct.m
```

```
1     function c = simpleProduct(a,b) %#codegen
2     c = a*b;
```

To generate C code from this function using MATLAB Coder, I first have to specify the size and data types of my inputs - I can do this through the MATLAB Coder UI (shown in the section below) where I specify my inputs as a [1x5] and [5x2] single precision matrices, and subsequently generate the following C code:

```
dbtype simpleProduct.c
```

Recent Posts Archive

25 APR Election Poll Analysis in MATLAB (http://blogs.mathworks.com/loren/2014/04/25/election-poll-analysis-in-matlab/)

7 APR Debunking Bad News Analysis with MATLAB (http://blogs.mathworks.com/loren/2014/04/07/debunking-bad-news-analysis-with-matlab/)

17 MAR MATLAB Virtual Conference 2014 (http://blogs.mathworks.com/loren/2014/03/17/matlab-virtual-conference-2014/)

14 MAR Integer Programming and Hyper Sudoku (http://blogs.mathworks.com/loren/2014/03/14/integer-programming-and-hyper-sudoku/)

26 FEB Arithmetic Associativity - Not So Fast (http://blogs.mathworks.com/loren/2014/02/26/arithmetic-associativity-not-so-fast/)

Categories expand all

- Best Practice 55
- Blog Policies 1
- Cell Arrays 6
- Code Generation 1
- Common Errors 42

more

Recent Comments

Loren Shure (http://blogs.mathworks.com/images/Loren2.jpg) on Election Poll Analysis in MATLAB (http://blogs.mathworks.com/loren/2014/04/25/election-poll-analysis-in-matlab/#comment-40440)

Cleve Moler (http://mathworks.com) on Arithmetic Associativity – Not So Fast (http://blogs.mathworks.com/loren/2014/02/26/arithmetic-associativity-not-so-fast/#comment-40292)

Michael Hosea on Arithmetic Associativity – Not So Fast (http://blogs.mathworks.com/loren/2014/02/26/arithmetic-associativity-not-so-fast/#comment-40291)

Sanjan_iitm on Arithmetic Associativity – Not So Fast (http://blogs.mathworks.com/loren/2014/02/26/arithmetic-associativity-not-so-fast/#comment-40141)

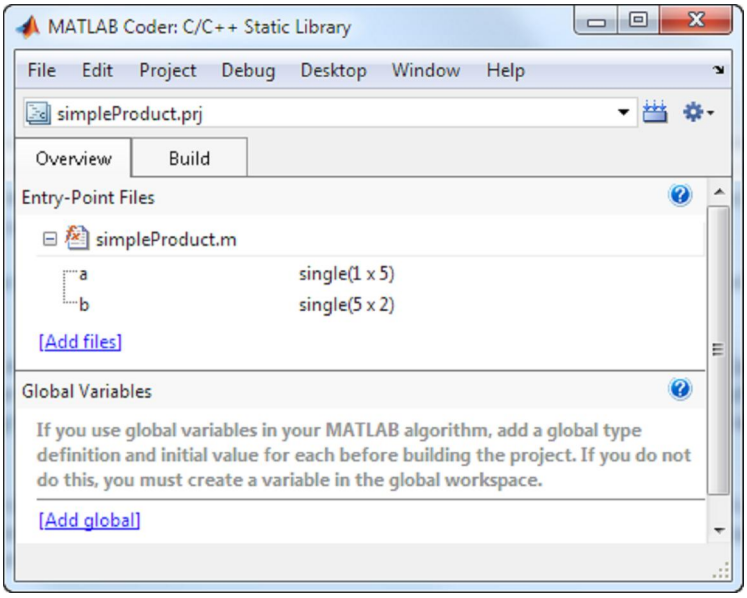
```
#include "simpleProduct.h"

void simpleProduct(const real32_T a[5], const real32_T b[10], real32_T c[2])
{
  int32_T i0;
  int32_T i1;
  for (i0 = 0; i0 < 2; i0++) {
    c[i0] = 0.0F;
    for (i1 = 0; i1 < 5; i1++) {
      c[i0] += a[i1] * b[i1 + 5 * i0];
    }
  }
}
```

MATLAB is a polymorphic language which means a single function, such as simpleProduct, can accept input arguments of different size and data types and output correct results. This function can behave as a simple scalar product, a dot product, or a matrix multiplier depending on what inputs you pass.

Languages like C are said to be “stongly typed,” which requires you to create a different version of a function for every variation of input data size and types. Hence, when you write C code to implement simpleProduct, you have to know ahead of time the sizes and the data types of your inputs so you can implement the right variant.

While MATLAB Coder helps you move from the highly flexible world of MATLAB to a strongly typed world of C, you still have to specify all of the constraints C expects. You do this in MATLAB Coder by creating a MATLAB Coder project file (simpleProduct.prj in this case) where you can specify the various code generation parameters including the sizes and data types of your inputs as shown below.



You can also edit the default configuration parameters by clicking on the 'More Settings' link which appears on the 'Build' tab of this project UI .

In the example above, I turned off a few default options such as including comments in the generated code just so you can see how compact the code is. You not only get the option to generate comments in the resulting C code, but also inlcude the original MATLAB code as comments in the corresponding sections of the generated code. This helps with traceability of your C code to your original algorithms.

Detailed MATLAB Coder Worfklow

The simple example above quickly illustrates the process of generating code with MATLAB coder and shows how the resulting C code looks.

Naturally, your real-world functions are going to be much more involved and may run into hundreds or even thousands of lines of MATLAB Code. To help you handle that level of complexity, you would need an iterative process, like the 3-step workflow described here, that guides you through the task of code generation incrementally:

Prepare: First, prepare your code to ensure that you can indeed generate code from your MATLAB algorithm. The convenience of MATLAB language doesn't map directly to the constrained behavior of C. You may have to re-write portions of your MATLAB code so it uses the MATLAB language features that support this mapping from MATLAB to C.
Test & Verify: Next, you generate a MEX function to test that your preparation step is correct. If the tool is successful in generating a MEX function then you are ready to verify the results of this MEX function against the original MATLAB code. If not, you'd have to iterate on the previous step till you can successfully generate a MEX function.

Generate: Finally, you generate C source code and further iterate upon your MATLAB code in order to control the look and feel or the performance of your C code.You can also generate an optimized MEX function by turning off certain memory integrity checks and debug options that could slow down its execution at this stage.

I'll now demonstrate this concept using a slightly more involved example. The following MATLAB code implements the Newton-Raphson numerical technique for computing the n-th root of a real valued number.

```
dbtype newtonSearchAlgorithm.m
```

```
1     function [x,h] = newtonSearchAlgorithm(b,n,tol) %#codegen
2     % Given, "a", this function finds the nth root of a
3     % number by finding where: x^n-a=0.
4
5         notDone = 1;
6         aNew    = 0; %Refined Guess Initialization
7         a       = 1; %Initial Guess
8         count     = 0;
9         h(1)=a;
10        while notDone
11            count = count+1;
12            [curVal,slope] = fcnDerivative(a,b,n); %square
13            yint = curVal-slope*a;
14            aNew = -yint/slope; %The new guess
15            h(count)=aNew;
16            if (abs(aNew-a) < tol) %Break if it's converged
17                notDone = 0;
18            elseif count>49 %after 50 iterations, stop
19                notDone = 0;
20                aNew = 0;
21            else
22                a = aNew;
23            end
24        end
25        x = aNew;
26
27    function [f,df] = fcnDerivative(a,b,n)
28    % Our function is f=a^n-b and it's derivative is n*a^(n-1).
29
30        f  = a^n-b;
31        df = n*a^(n-1);
```

Our first step towards generating C code from this file is to prepare it for code generation. For code generation, each variable inside your MATLAB code must be intialized, which means specifying its size and data type. In this case, I'll choose to initialize the variable h to a static maximum size, h = zeros(1,50), as it doesn't grow beyond a length of 50 inside the for loop.

You can invoke the code generation function using the GUI or the command line (through the codegen command). Without worrying about the details of this approach, let's look at the generated C code:

```
dbtype newtonSearchAlgorithmMLC.c
```

```
#include "newtonSearchAlgorithmMLC.h"

void newtonSearchAlgorithmMLC(real_T b, real_T n, real_T tol, real_T *x, real_T
  h[50])
{
  int32_T notDone;
  real_T a;
  int32_T count;
  real_T u1;
  real_T slope;
  notDone = 1;
  *x = 0.0;
  a = 1.0;
  count = -1;
  memset((void *)&h[0], 0, 50U * sizeof(real_T));
  h[0] = 1.0;
  while (notDone != 0) {
    count++;
    u1 = n - 1.0;
    u1 = pow(a, u1);
    slope = n * u1;
    u1 = pow(a, n);
    *x = -((u1 - b) - slope * a) / slope;
    h[count] = *x;
    if (fabs(*x - a) < tol) {
      notDone = 0;
    } else if (count + 1 > 49) {
      notDone = 0;
      *x = 0.0;
    } else {
      a = *x;
    }
  }
}
```

The subfunction fcnDerivative is inlined in the generated C code. You can choose to not inline the code by putting this command coder.inline('never') in the MATLAB file.

Support for Dynamic Sizing

If you have variable in your MATLAB code that needs to vary its size during execution, you can choose to deal with this in three different ways in the generated C code:

Static allocation with fixed maximum size: you can initialize variables to the maximum possible static size (like I did in the example above). In the generated code, memory is prealloacted to this size.
Variable sizing with maximum size allocation: this option will declare the memory for the variable in the generated code to its maximum possible size, but you can dynamically grow or shrink the variable size within this allocated maximum.
Variable sizing with dynamic memory allocation: this results in the generated code using malloc to allocate the memory for the variables that change in size during code execution.

The last two options can be enabled by turning on the **variable-sizing** feature in the configuration parameters. However, do remember that enabling this option also makes the resulting C code bigger in size - so if you can avoid it, you can get much more compact and possibly more efficient code.

Use Cases of MATLAB Coder

The primary use of MATLAB Coder is to enable algorithm developers and system engineers working in MATLAB to quickly generate readable and portable C code. The generated C code can be used in different ways supporting different workflows. Here are a few use cases of MATLAB Coder:

Create standalone executables from your algorithms for prototyping on a PC
Speed up your MATLAB algorithm code (or portions of it) by generating and executing the MEX functions
Integrate your MATLAB algorithms as C source code or compiled library with your hand-written software

I won't be talking about creating standalone executables or creating libraries from MATLAB Coder in this blog; I'll, however, say a few words on generating MEX functions.

Generating MEX Functions for Simulation Acceleration

The MEX interface enables you to integrate C code into MATLAB. One common use of MEX is to speed up performance bottlenecks in MATLAB code by re-writing them in C and executing them back in MATLAB as MEX functions. MATLAB Coder can

save you time and effort by automatically generating MEX functions from your MATLAB code (and allowing you to import legacy C code easily as well).

The speed-up you get through automatic MEX generation can vary quite a bit depending on the application. In some cases, you may not get any speed up at all, or possibly even a slow-down, as MATLAB language has gotten quite smart and efficient in computing many built-in functions by automatically taking advantage of processor specific routines (such as Intel Performance Primitives) and multithreading to utilize multiple cores.

A few guidelines that you can use to determine if your algorithm is a good candidate for speed-up with MEX are:

your algorithm contains multiply nested for loops, and possibly handles state calculations (making each iteration dependent on the previous one)
your MATLAB code is hard/impossible to vectorize
most of the processing cycles in your algorithm are **not** spent on already optimized built-in functions such as fft or inv, etc.
you don't rely heavily on toolbox functions (especially those unsupported for code generation) in your algorithms

In these cases, you can expect to see a speed-up. Let's look at a realistic example that illustrates this concept.

The example I chose here is one that my colleague Sarah Zaranek (http://blogs.mathworks.com/loren/2008/06/25/speeding-up-matlab-applications/) had highlighted in her guest blog on vectorization.

I modified the original MATLAB script from that article into a function as only functions are supported by MATLAB Coder. The other change that I made to the original script was to initialize the output variables subs and vals (as all variables in your code has to be defined/intialized once when used with MATLAB Coder). And by turning on the variable sizing feature (using dynamic memory allocation), I didn't have to preallocate it to a sufficiently large enough size to accomodate its growth, which would use more memory (often a lot) than needed.

The following statements create and use a MATLAB Coder configuration object with support for variable size arrays and dynamic memory allocation to generate a MEX function.

```
% Create a MEX configuration object
cfg = coder.config('mex');
% Turn on dynamic memory allocation
cfg.DynamicMemoryAllocation = 'AllVariableSizeArrays';
% Generate MEX function
codegen -config cfg gridSpeedUpFcn -args {10}
disp('MEX generation complete!')
```

```
Warning: There is no short path form of 'H:\Documents\LOREN\MyJob\Art of
MATLAB' available. Short path names do not include embedded spaces. The
short path name feature may be disabled in your operating system. Short
path names are required for successful code generation, therefore it may
not be possible to successfully complete the code generation process. To
avoid this problem, enable short path name support in your operating
system, or do not attempt code generation in paths containing spaces.
MEX generation complete!
```

This generates a MEX function in the current folder. I'll use tic, and toc to estimate the execution times of the original MATLAB function and its MEX version.

I run each function multiple times inside a for loop and use only the last few runs for our computation time calculation so we can minimize the effects of initialization and caching.

```
MToc = zeros(1,30);
MexToc = zeros(1,30);
for j = 1:30
    tic; gridSpeedUpFcn(10);
    MToc(j) = toc;
end

for j = 1:30
    tic; gridSpeedUpFcn_mex(10);
    MexToc(j) = toc;
end

eff_M_time = mean(MToc(1,21:30));
eff_Mex_time = mean(MexToc(1,21:30));

disp(['MATLAB code execution time (with nx=10): ', num2str(eff_M_time)])
disp(['MEX code executing time (with nx=10): ', num2str(eff_Mex_time)])

disp([' Speed up factor: ', num2str(eff_M_time/eff_Mex_time)]);
```

```
MATLAB code execution time (with nx=10): 0.31652
MEX code executing time (with nx=10): 0.0027057
 Speed up factor: 116.9806
```

The speed up we see in this example is more pronounced for larger values of nx (and would vary between different computers).

Vectorization and Code Generation

Sarah, in her blog article (http://blogs.mathworks.com/loren/2008/06/25/speeding-up-matlab-applications/), explains quite nicely on how to vectorize this code using meshgrid and ndgrid to explode the variables in support of vectorization, and get similar levels of speed up.

Sarah paid a small price in code readability for her vectorization efforts, but a much larger price in memory consumption. This points to a common trade-off when vectorizing MATLAB code: execution time vs. memory consumption. The temporary variables which enable us to remove loops through matrix math are often very large, effectively limiting the number of iterations you can take before running out of memory. For instance, if we run her algorithm for a 1,000x1,000 grid instead of a 100x100 grid, we would create 9 double precision variables that are about 7.5 GB each!

Code generation can effectively deliver similar speed-up benefits (and with less effort in some cases), as it's trying to explicitly achieve the same for loop operations in C code that a vectorized code achieves. However, the downside for most users with the code generation approach is that it's available with a separate product (higher cost), and only MATLAB code that supports code generation can automatically be converted to MEX as well.

Customizing and Optimizing the Generated Code

Now with most of the basics of code generation out of the way, a few questions might be on your mind:

"If I want the resulting code to look different, can I just edit the generated code?" You can certainly edit the generated code; as you have just seen in the examples shown here, the code is very readable. However, once you modify it, you lose the connectivity to the original MATLAB code that generated it.
"How can I optimize the code, or customize its look and feel?" One easy way to modify the generated code is to change MATLAB Coder configuration parameters. Another way to customize the generated code is to directly edit the MATLAB code such as explicitly breaking up certain compact vector/matrix operations into for loops or other constructs you'd like to see in the generated code.
"What about incorporating hand-optimized or processor specific intrinsics for certain portions of the code?" coder.ceval is a command in MATLAB Coder that let's you integrate custom C code into your MATLAB code for both simulation and code generation. Using this you can use hand-optimized or legacy C code in your algorithms. Embedded Coder is another product that adds many code optimization and customization capabilities to MATLAB Coder such as using code replacement technology to incorporate optimized C code. It also lets you to customize the look-and-feel of the code.

Learning More About MATLAB Coder

You can find more information about MATLAB Coder on the product page (www-external-test.mathworks.com/products/matlab-coder/), which includes demos and recorded webinars. The product documentation (http://www.mathworks.com/help/releases/R2011b/toolbox/coder/ug/ug_intropage.html) is also an excellent resource to find answers to questions you might have when starting with this product.

Please feel free to share your thoughts and comments on topics discussed here in this posting here (http://blogs.mathworks.com/loren/?p=296#respond).

Get the MATLAB code

Published with MATLAB® 7.13

Category: Code Generation, Deployment

Send

< Subset Selection and Regularization (http://blogs.mathworks.com/loren/2011/11/21/subset-selection-and-regularization/)

|

Managing Deployed Application Output with... > (http://blogs.mathworks.com/loren/2011/11/04/managing-deployed-application-output-with-message-handlers/)

You can follow any comments to this entry through the RSS 2.0 (http://blogs.mathworks.com/loren/2011/11/14/generating-c-code-from-your-matlab-algorithms/feed/) feed. Both comments and pings are currently closed.

9 Comments

Oldest to Newest

Sarah replied on November 14th, 2011 at 18:51 UTC : **1** of 9

Hi Arvind,

Just a note, that the “overhead” of large matrices created for vectorization can also often be avoided by the use of bsxfun in core MATLAB.

Cheers,
Sarah

Venn Ravichandran replied on November 15th, 2011 at 18:52 UTC : **2** of 9

Arvind,

Is there a reason that the coder doesn’t use templates (in C++) or #define pre-processor headers (in C) to get around strong-typing?

Thanks,
-Venn

Oyster Engineer replied on November 15th, 2011 at 21:23 UTC : **3** of 9

A couple of comments.

1st, I don’t see any logic in the History you discuss. How is new capability in MatLab related in any way to the previous capability in Simulink?

2nd, You should have introduced MEX in more detail up front rather than at the end as a “by the way” method of using this new product’s capability. Further, don’t users need a clear understanding of how to revise this autogenerated C code to re-use in MatLab? Where is this sound guidance? Frankly, one of the main advantages of MatLab is that I don’t have to be a master of C to write high performance technical computing software.

3rd, I think you’ve violated some sound Coding Style Guidelines by using style:

notDone = 1
.
.
while notDone

Rather than:

Done = 0
.
.
while ~Done

Paul J. replied on November 16th, 2011 at 03:43 UTC : **4** of 9

Arvind,

1. As you point out, autogenerated mex files can be much slower than m-code. Here’s an example: http://www.mathworks.com/matlabcentral/answers/1115-why-does-emlmex-generate-a-slow-mex-file (http://www.mathworks.com/matlabcentral/answers/1115-why-does-emlmex-generate-a-slow-mex-file). However, Mike Hosea pointed out in that thread that one possibilit is to use eml.extrinsic (I assume there’s something similar in codegen) to use use the Matlab version of certain functions like FFT that may be better optimized than the eml (or codegen version).

I just upgraded to R2011A and tried codegen to autogenerate a mex-file from an m-file. Couldn’t do it because I don’t have a license for Coder. However, I could still use emlmex, though I was warned that emlmex is going away in future versions.

2a. Is emlmex synonymous with codegen when used for generating mex-files? That is does emlmex generate the same c-code that codegen does?

2b. When emlmex goes away, will base matlab come with a “codegen lite” that has the emlmex functionality?

Arvind replied on November 17th, 2011 at 03:21 UTC : **5** of 9

Venn:
We currently don’t support C++ templates; moreover, we don’t generate templates for many reasons. First of all C++ template semantics and dispatching rules are complex and would not be easy to build a tool that would reliably get this right. Secondly, it would be hard to establish what parameters would be appropriate. Discovering how to generalize code for the purpose of creating templates is a hard problem that’d d require some sort of symbolic evaluation.

Preprocessor directives really don’t help with this problem, unless you imagine recompiling the code with different header files. Even in this approach, discovering the parameters, as in the case of templates, is the challenge.

Arvind replied on November 17th, 2011 at 04:14 UTC : **6** of 9

Oyster engineer:
Re: “… don’t users need a clear understanding of how to revise this autogenerated C code to re-use in MatLab?”
It’s not clear what you mean by “revise the autogenerated code”.
If you are referring how to execute the generated code in MATLAB without manually creating a MEX wrapper, one option is to use the MEX generation capability in MATLAB Coder to automatically compile your MATLAB code directly into a MEX function. Another approach is to use the coder.ceval function (also in MATLAB Coder), that lets you call not just the auto generated C code, but any legacy C code for simulation and code generation in your MATLAB code.

Arvind replied on November 17th, 2011 at 17:30 UTC : **7** of 9

Hi Paul J:
1. You could certainly try coder.extrinsic to call built-in/toolbox functions in your MATLAB code that are not supported for code generation. A more uncommon usage for coder.extrinsic is to keep the processing of certain (MATLAB Coder supported) built-in functions in MATLAB instead of generating code for them, especially if they are more efficient in MATLAB as is. But that begs the question, why you would even want to use MATLAB Coder instead of just staying in MATLAB. As I state in the article, if bulk of your processing is spent on such functions, you may be better off working within MATLAB to identify ways to speed up the rest of your code.

2a/b. Default option for the codegen command is to generate MEX. The C code that is generated for MEX is a bit different from the codegen command when choosing to generate C source code. This is because, the code itself will look different when we auto-generate the MEX wrappers within the algorithm C code, but also introduce certain run-time and CTRL-C checking as the resulting MEX is intended for execution within MATLAB. You can certainly turn off some of these memory integrity checks and other options from the build settings.

emlmex is a legacy function that’s not supported any longer and this capability will be available only through codegen. current versions of emlmex are being maintained in the product (and not documented) for backward compatibility for a limited period, and will be removed soon.

Going forward you would need a MATLAB Coder license to be able to generate MEX functions automatically. We may be considering a ‘codegen lite’ option but currently don’t have plans for releasing such a capability.

Paul J. replied on November 17th, 2011 at 22:55 UTC : **8** of 9

Arvind,

1. In this case I was in the following situation. I had an m-file that I wanted to run from Matlab and also call from an embedded function in Simulink. The simulink model slowed to a crawl and as best I could determine the issue was that code generated for Simulink, which I assume is using the same engine as emlmex, was slowing down because it doesn’t use the same optimized version of FFT as Matlab does. So in this case it might have been better to specify eml.extrinsic(‘fft’) so that the Simulink code would use the optimized Matlab version. I hope I explained it well.

3. In the future, will Simulink still support embedded functions and Simulink Accelerator without an extra license for MATLAB Coder?

4. I’m still a little hazy about the relationship between EML and Coder. Does the latter replace the former?

Arvind replied on November 21st, 2011 at 16:56 UTC : **9** of 9

Paul:
1. Thanks for the clarification. You could certainly try coder.extrinsic here to force execution of certain function back in MATLAB if they are faster there (like FFT). Do keep in mind that the marshalling of data back and forth adds some overhead and in some cases could offset the speed advantages you may get by running in native MATLAB.

2(3). Yes

4. MATLAB Coder replaces EML – but certain functionality such as the MATLAB Function block in Simulink that uses code generation technology in MATLAB will continue to work without a separate MATLAB Coder license in the foreseeable future.

These postings are the author's and don't necessarily represent the opinions of MathWorks.