Tutorial 7: Speeding up your MATLAB code

Mex file generation

You can use MATLAB coder app to generate MATLAB executable (mex) files. Mex files run faster than the scripts.

Exercise 4

- Let's create a simple program and save it as 'Tute7_4.m'.

```
clear all, close all

a =3;
b=2;
c=a^2+b;
```

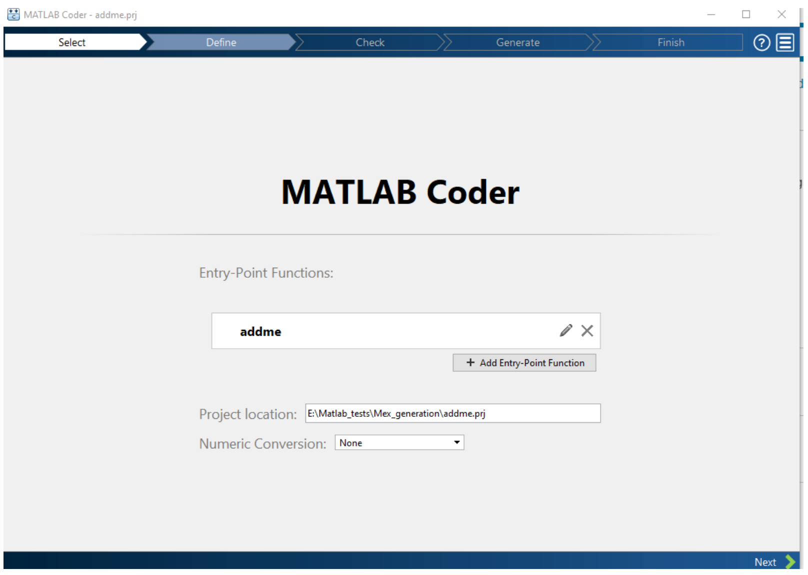
- Now, create a function to add two numbers.

```
clear all, close all

a =3;
b=2;
c = addme(a,b)

function c = addme(a,b)
c=a^2+b;
end
```

- Let's move the function to a new file and save it as, 'addme.m'.
- Now, we have two files- 'Tute7_4.m' and 'addme.m'.
- We can convert our function into a mex file.
- From MATLAB apps menu, open MATLAB Coder.
- Select addme function and click Next.



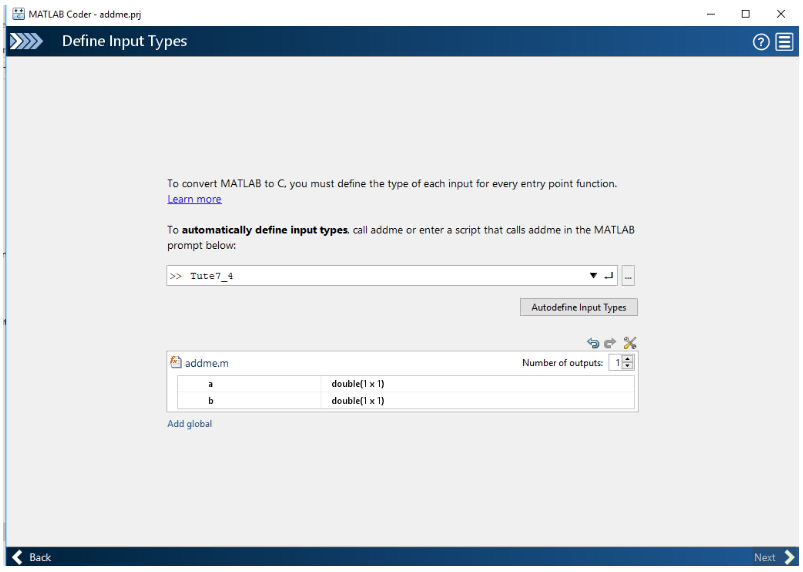
Entry-Point Functions:

addme

Project location: E:\Matlab_tests\Mex_generation\addme.prj

Numeric Conversion: None

- Next, you have to define input types for your function. This can be done by selecting the main script which calls the 'addme' function. Browse and select 'Tute7_4.m' and click on 'Autodefine Input Types'. Click Next.



To convert MATLAB to C, you must define the type of each input for every entry point function. Learn more

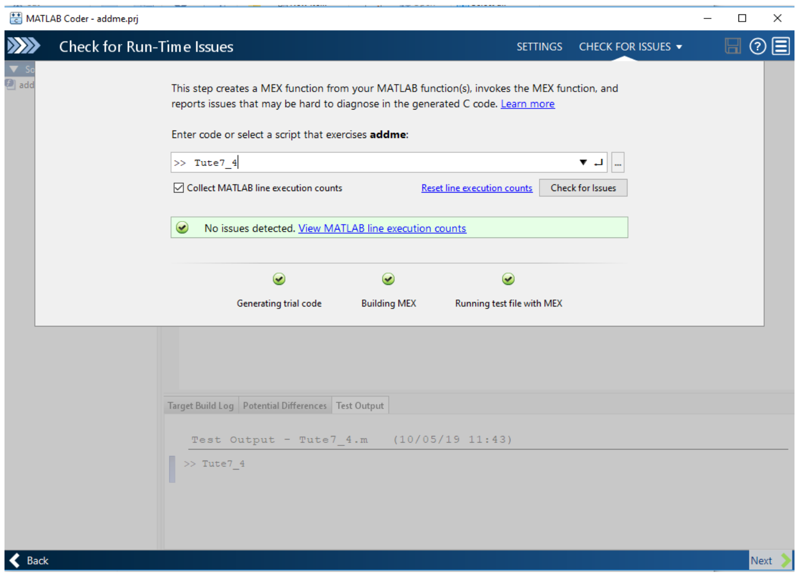
To **automatically define input types**, call addme or enter a script that calls addme in the MATLAB prompt below:

```
>> Tute7_4
```

addme.m — Number of outputs: 1

a	double(1 x 1)
b	double(1 x 1)

- Then, MATLAB will check for runtime issues. In the next window, click on 'Check for issues'.
- If no issues were detected you will see a window like below. Click Next.



This step creates a MEX function from your MATLAB function(s), invokes the MEX function, and reports issues that may be hard to diagnose in the generated C code. Learn more

Enter code or select a script that exercises **addme**:

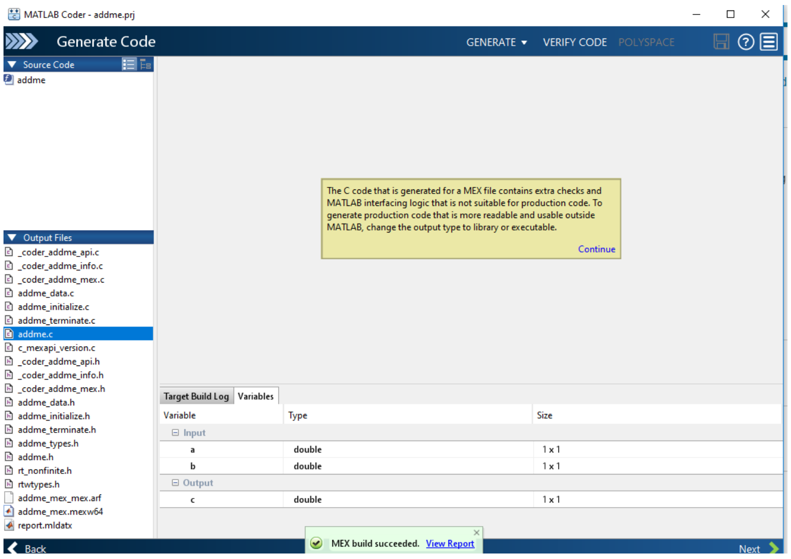
```
>> Tute7_4
```

☑ Collect MATLAB line execution counts

No issues detected. View MATLAB line execution counts

```
Test Output - Tute7_4.m    (10/05/19 11:43)
>> Tute7_4
```

- In the next window, select Build type as 'MEX', and click Generate.
- After generating the mex file, it will show a build succeeded message.


MATLAB Coder - addme.prj

Generate Code

addme

Output Files

_coder_addme_api.c
_coder_addme_info.c
_coder_addme_mex.c
addme_data.c
addme_initialize.c
addme_terminate.c
addme.c
c_mexapi_version.c
_coder_addme_api.h
_coder_addme_info.h
_coder_addme_mex.h
addme_data.h
addme_initialize.h
addme_terminate.h
addme_types.h
addme.h
rt_nonfinite.h
rtwtypes.h
addme_mex_mex.arf
addme_mex.mexw64
report.mldatx

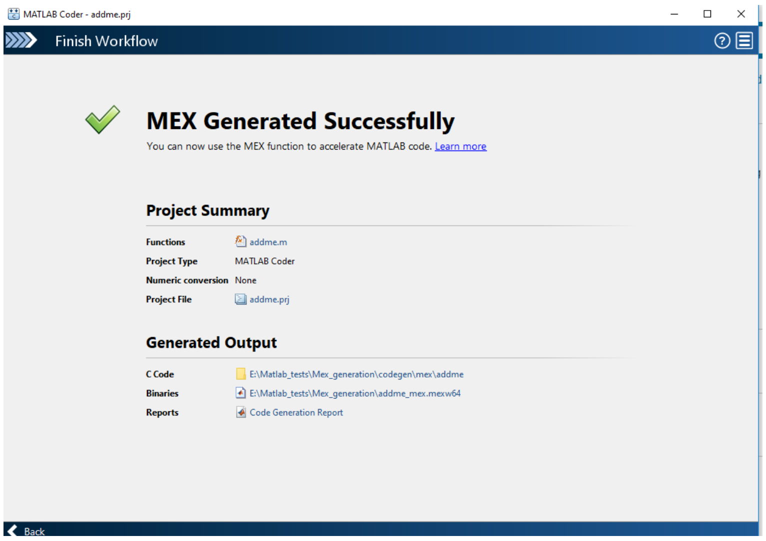
The C code that is generated for a MEX file contains extra checks and MATLAB interfacing logic that is not suitable for production code. To generate production code that is more readable and usable outside MATLAB, change the output type to library or executable.

Continue

Variable	Type	Size
Input		
a	double	1 x 1
b	double	1 x 1
Output		
c	double	1 x 1

MEX build succeeded. View Report

- Click Next.


MATLAB Coder - addme.prj

Finish Workflow

MEX Generated Successfully

You can now use the MEX function to accelerate MATLAB code. Learn more

Project Summary

Functions	addme.m
Project Type	MATLAB Coder
Numeric conversion	None
Project File	addme.prj

Generated Output

C Code	E:\Matlab_tests\Mex_generation\codegen\mex\addme
Binaries	E:\Matlab_tests\Mex_generation\addme_mex.mexw64
Reports	Code Generation Report

Now, modify your ‘Tute8_4.m’ code as follows to use the mex file. New mex file does the same task but it is faster.

```
clear all, close all

a =3;
b=2;
% c=addme(a,b);

c= addme_mex(a,b)
```

- Let’s check the speeds of the two functions by adding tic,toc commands.

```
clear all, close all

a =3;
```

```
b=2;

tic
c=addme(a,b);
toc

tic
c= addme_mex(a,b)
toc
```

You will see the execution times on the command line. Mex function is always faster than the script.

Exercise 4

Let’s do another example. You are given an example code from your Tutorial 6 – ‘Tute_6_5_mex.m’. This is the original code you tested last week.

In the given code, move the below code block to a new function called ‘findBoundaries’ and save it in the same directory.

```
function findBoundaries(B,stats,threshold)
% loop over the boundaries
for k = 1:length(B)

  % obtain (X,Y) boundary coordinates corresponding to label 'k'
  boundary = B{k};

  % compute a simple estimate of the object's perimeter
  delta_sq = diff(boundary).^2;
  perimeter = sum(sqrt(sum(delta_sq,2)));

  % obtain the area calculation corresponding to label 'k'
  area = stats(k).Area;

  % compute the roundness metric
  metric = 4*pi*area/perimeter^2;

  % display the results
  metric_string = sprintf('%2.2f',metric);

  % mark objects above the threshold with a black circle
  if metric > threshold
    centroid = stats(k).Centroid;
    plot(centroid(1),centroid(2),'ko');
  end

  text(boundary(1,2)-35,boundary(1,1)+13,metric_string,'Color','y',...
       'FontSize',14,'FontWeight','bold');

end

title(['Metrics closer to 1 indicate that ',...
       'the object is approximately round']);
```

```
end
```

- Call ‘findBoundaries’ function from your ‘Tute_6_5_mex.m’ code.
- Compare the speeds of two functions using tic,toc commands.

You can generate mex functions to most of your codes and speed up your programs.

You are advised to use mex files for your project to speed up the processing.

A video tutorial is available on: https://au.mathworks.com/videos/generating-c-code-from-matlab-code-68964.html