Training Unit 6: Automatic Code Generation

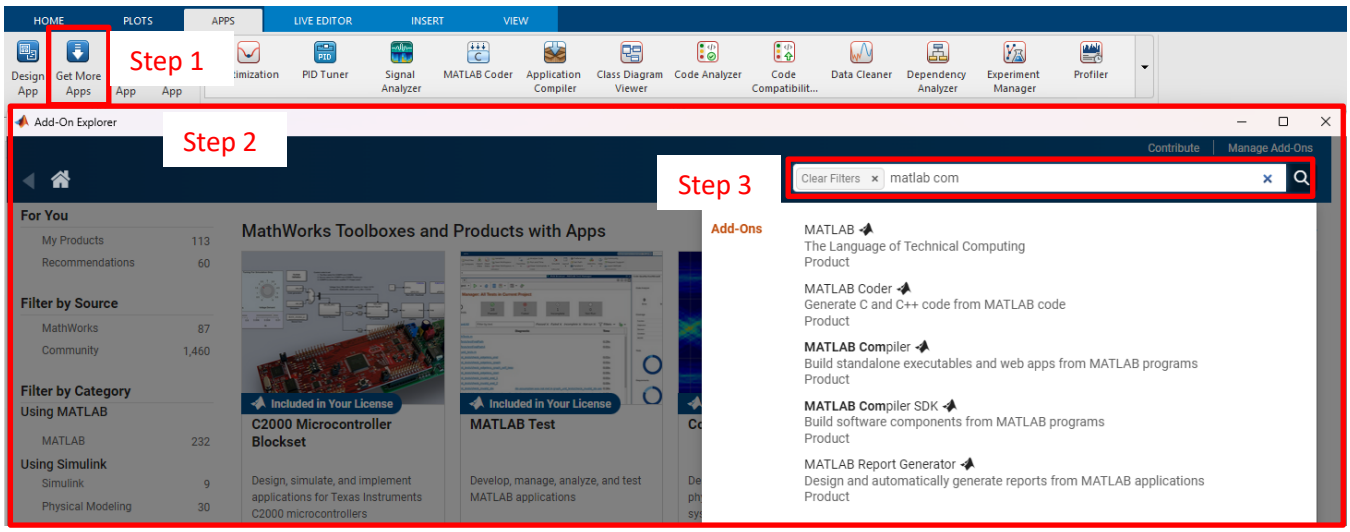
1 INTRODUCTION TO TRAINING UNIT

After learning the characteristics of code generation, it is time to get some hands-on experience with Matlab / Simulink's code generation possibilities. But before you start with code generation, be sure to go through the section 2 of this document **thoroughly**. Any problems in the initial configuration section can be tedious to resolve in later training unit tasks.
The general task of this training unit will be the code generation and performance evaluation of Matlab functions, Matlab executable files (MEX files) and binary files (called within the windows terminal).

2 PREPARATION

2.1 Get all necessary Add-on packages

- For this unit you will need following MathWorks Add-ons toolboxes (included in students license):
  - Matlab Coder
  - Matlab Compiler
  - Simulink Coder
  - Embedded Coder
- In case you are missing one of them be sure to install them via the "Apps" toolstrip under "Get more apps" (see Step 1 below) which will open the Add-On Explorer (see Step 2 below) and search for the missing Add-ons in the search field (see Step 3 below)



- Additionally you will also need to download the free **MinGW-w64 C/C++ compiler** suite to be able to compile C/C++ applications
- This **free Matlab support package** can be downloaded with the same steps as previously explained via the "Get more apps" button.
- Search for "mingw" and it should be the first add-on that appears (see picture below) and follow the installation instructions

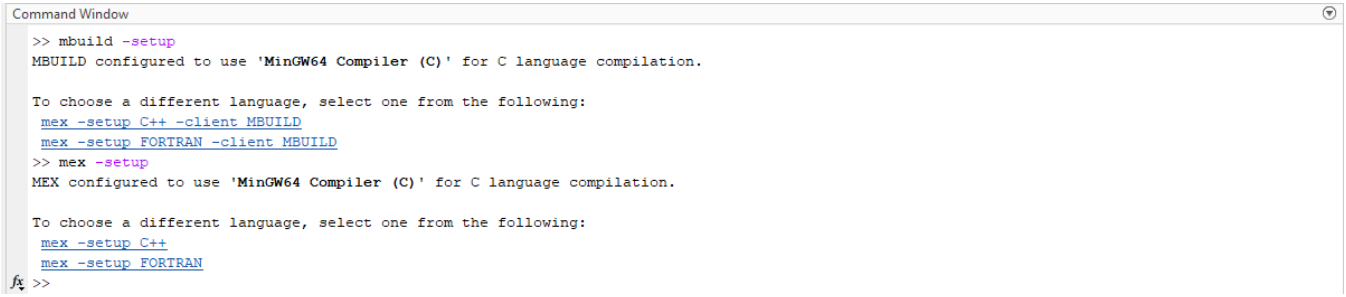
2.2 OPTIONAL (!) step (using already installed MinGW compiler)

Use manually installed MinGW-64 compiler (**at own risk**, as problems could arise!)

- If you already have installed MinGW-64 on your PC and you want to use this as your C compiler in Matlab, you could do so as well. As this **could cause some issues**, especially setting-up correctly, be sure to check out the compatibility table under this LINK and follow the instructions given in this LINK.
- To check, whether you already have a compiler installed on your PC, use your windows command window (terminal) and type "**g++ --version**" which should return the installed compiler version.

2.3 Verify correct Matlab setup / configuration

- To verify, that Matlab correctly uses the installed GCC compiler (MinGW support package or manually configured through the additional steps) use following commands in your Matlab command window:
  - mbuild -setup (see picture below for return message)
  - mex -setup (see picture below for return message)



- To verify the correct functioning of your Matlab compiler, do the following steps:
- Navigate your current Matlab working directory in a folder with read/write access
- Copy following command in your Matlab command window
  - *copyfile(fullfile(matlabroot,'extern','examples','mex','yprime.c'),'.','f')*
  - If this, for any reason does not work (e.g. older Matlab version) use the yprime.c file provided in the Moodle training unit task section
- This should copy a single file "yprime.c" into your working directory
- Copy following command into your Matlab command window again
  - *mex yprime.c*
    - *→ Building with 'MinGW64 Compiler (C)'.*
    - *→ MEX completed successfully.*
- This should produce a MEX function which can be called in your Matlab command window like:
  - *yprime(1,1:4)*
    - *ans = 2.0000 8.9685 4.0000 -1.0947*
- Now you have successfully managed setting-up your Matlab environment for the following training unit
- HINT (!) You can use "Matlab Online" in your browser as well for this training unit. Here the setup should already be done for you. You can perform the above-mentioned verification as well within "Matlab Online"

3 REPORT

- The report for this training unit is based on your **Matlab live script** and **all your files** that are created during the training unit, that you will start from the first task. (zip your working directory and upload it to Moodle)
- Make sure to structure your work in the live script using the available editor tools (structure your code in section, add headers to your sections, table of contents, use **bold** or *italic* formats, bullet points, etc.)
- As you will consistently add new features to your live script, make sure to add new sections to each task, to provide a well-documented and structured live script. This will require you to copy-paste some of your code throughout the document most likely. Use the **subtask number** of this training unit description and **header** for your live script as **section header** for a more **consistent documentation**
- Try to make use of UI-controls throughout the script, to generate an interactive script
- Be sure to check the correct function of your scripts before uploading it. If your script **does not work**, you **will not receive any points** for this training unit.
- Make sure, that the script shows **your work** on this training unit task. Using solutions from other colleagues will result in zero points for you and your colleague.
- **Caution (!)** be sure to check the **due date** that is set for uploading the files on Moodle!

4 LABORATORY WORK

4.1 Generating a simple Matlab function

- The first task is to create your live script, that will document your work
- Add an external Matlab function (*.m) file that includes a simple mathematical function
- For this training unit we will use a Fibonacci number generator as our main function to investigate
- This function is given by:

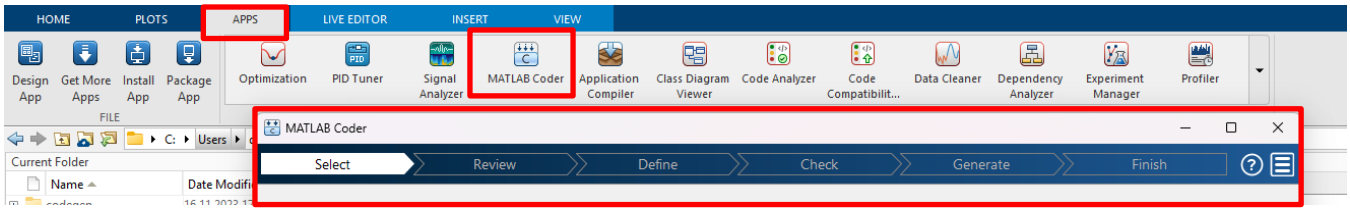
$$F_n = F_{n-1} + F_{n-2}$$

$$F_0 = 0, F_1 = 1$$

- The Matlab function shall have one parameter **n** and one return value **F_n** (return only the **final** Fibonacci number, not the Fibonacci sequence)
- Implement the function in a recursive way!
- Test your Matlab function by calling it in your live script (be aware to test only small numbers at first to avoid too long calculation times e.g. n = 10)
- No let's try to use some undefined input parameters to verify that your function can also handle false inputs. (Hint! Look into this article LINK !) Therefore, test wit **n =**
    - -1
    - 'a'
    - [1,2]
    - [2,1]
    - 5i (--> equals a complex number)
    - 3.41
    - "Hello Fibonacci"
    - etc.
- If your function is error-proof, continue testing your PC by increasing the parameter **n** slowly in steps of 5
    - What is the "limit" of your machine? **limit** = when the calculation takes more than 10 seconds

4.2 Generating a Matlab executable function from your Fibonacci function

- Start with opening the **Matlab coder app** under the "Apps" toolstrip. It should be located in the toolbar (there is also a command line possibility to generate MEX functions and C functions, but let's start with a more visual approach using the in-built Matlab app)



- This will open the Matlab Coder app which guides you through the coding process (4 to 5 steps)
- Tipps for the Coder App:
  - When "Defining Input Types" you can add an example function call with your Matlab function name to let Matlab define the input for your, or you can specify the input manually as well
  - When asked, to generate code, be sure to select "MEX" as Build type and "C" as language
  - When everything compiled as configured, a new file should be available in your workspace that should have the same name as your Matlab function with "_mex.mex64" as extension
  - If you have troubles, check out this tutorial for additional help LINK
- Make sure to open the code generation report and examine the autogenerated MEX code:
  - What can you see?
  - How many files are generated?
  - What is the purpose of those files?
  - ***Make sure to document your thoughts within your live script***
- Now it's time to test the correct function of your MEX function
  - Call the MEX-function in your live script, using its function name and the parameter **n**
  - Call the MEX-function with false input parameters, as you did in the first subtask. Does it behave similar than calling the original Matlab function?
  - Now let's try to use your maximum input parameter value from the previous subtask
    - Does it also work? Can you see a difference in the performance? Can you even increase your maximum number further?

4.3 Function performance testing

- In the last task, you compared the two function calls subjectively and maybe discovered some differences in the function performance (timing). Now let's try to test both functions objectively!
- For this, make sure to read through the Matlab online help section of the command tic and toc. These two function calls will be your stopwatches for this task to help you evaluate the function performance accurately
- Add a new section to your live script for performance testing
- Define a variable the controls the amount of repetitive function call runs
    - To be able to reduce the influence of other OS tasks, that maybe influence your function call performance, it is always good to run the function several times iteratively
        - Be sure to close all running tasks, that are not required for this test (e.g. browser)
    - Add an array to store the single function run times
    - Add for-lops for your function testing
    - Stop the time for each function call and save it to your array
- Start with following parameters settings
    - Make **n** (Fibonacci number) small at the beginning, to make the function call fast (let's say a value beneath n = 10, e.g. 8)
    - Make the number of loop runs high (let's say 1,000 or 10,000)
    - Both values heavily depend on your hardware setup and your running tasks
        - Make sure to close as much tasks as possible to avoid high variance in your function calls
        - Try to find a good set of parameters that works best with your hardware
- Evaluate the performance of your function call statistically
    - Plot your result in a meaningful way (Hint: check out following visualisation techniques and choose the ones you think visualizes the task best histogram, bar, pareto, scatter)
        - What can you see?
        - Maybe try to improve your plot by comparing two timings (Matlab function vs. MEX function)?
        - What is the **mean function calculation time** over all runs?
        - What is the **time-saving** in percent, when using MEX functions instead of Matlab functions?
    - If you pre-allocated the array for saving the simulation runs --> GOOD JOB! Paying attention to what Matlab suggests, when not pre-allocating an array and changing its size during run-time
    - Now try to add a testing loop, where you intentionally do not pre-allocate the array.
    - Note that you should have more than 1,000 simulation runs (10 K would be better), to really see an effect of pre-allocating the array with the amount of simulation runs
    - Make a plot that shows the effect of pre-allocation on the calculation performance
- Now let's tune your parameter settings and increase calculation load
    - Add a new section to your script
    - Increase the Fibonacci number to higher values (depending on hardware!)
        - Start with n = 20 and continue to increase the number until your complete calculation takes several seconds
    - Try to keep the for loop runs high to be able to get a good average result of calculation performance
        - Keep the simulation runs to approximately 500
    - What is changing compared to the first task (**n** was small and simulation runs were high)? --> Make sure to document it in your live script

4.4 Generating a windows executable (*.exe) file from your Fibonacci Matlab function

- Start with opening the **Matlab coder app** under the "Apps" toolstrip and setup the coding settings once again or duplicate the coder project file (*.prj) and give it another **name** and go to section 4 "generate".
- Now instead of selecting "MEX" select "Executable".
- To be able to generate an executable file from your Fibonacci function you need to perform an intermediate step, as you need to take care of the I/O configuration as well
    - First, let Matlab generate only a C-function from your Matlab function
    - Therefore tick the option "Generate Code Only"
    - As Hardware Board use "Matlab Host computer"
    - And the toolchain should be automatically detected by Matlab, when you performed the steps in section 2 of this document correctly
    - Generate Code
- View the Code generation report and examine the output
    - Make screenshots and implement notable findings in your live script, that you find worth mentioning
- If you click "Next" and change to the "Finish Workflow" section, you should not be able to find an executable binary. What files are listed under the option binaries?
- You should be able to find an "example" main.c and main.h file in your "codegen" folder in your workspace
    - These files will be necessary for further tasks
    - As mentioned before, Matlab cannot create an executable binary file straight away.
        - Have you maybe tried it?
        - What do you think, why is it not possible, to generate an executable binary, solely with the Matlab function available for the Matlab coder, that can directly be called in your terminal?
    - Extract the main.c and main.h files from the examples to the folder structure above, where also all other generated files are located
    - Rename the main.c and main.h file to a different name
    - Open the file and review it. Can you find a reason, why a compilation would not produce the desired results?
- Manipulate the renamed main.c file to be able interact with the windows command line (terminal)
    - Ask the user to input a number for the Fibonacci calculator
    - Read the number and forward it to the autogenerated Fibonacci function
    - Write the output to the terminal again
- Now configure the code generation to use your modified main function as entry function for the executable
    - Untick the option "Generate code only" as we now want an executable as well
    - Go to "More Settings" under "Custom Code" and provide the path to your modified main.c function to the "Additional Source Files"
    - Click "Close" and Generate Code. Now a binary file should be created
    - Click "Next" and check the list of binaries now. Does it list an *.exe file now?
- Open a power shell window and navigate to your working directory and start the program by simply typing its file name with ".\" as a prefix. (e.g. ".\fibonacci_func_exe.exe")
    - Your program should start as ask for a Fibonacci number
    - Congratulations! Your programmed your first windows executable with Matlab!