

ECE-210-A Homework Feedback

Jonathan Lam

Spring 2022

1 Introduction to MATLAB

TODO

2 Vectorization and for loops

Suppress intermediate and long outputs. I don't need to see the outputs in the command window to evaluate your HW – there are other ways. Printing large matrices (and other I/O in general) is also very slow (it will definitely mess with timing).

Keep your code DRY (DRY = “Don't repeat yourself,” and is a common maxim in software engineering.) If you see a constant (e.g., 100, 1000) being repeated throughout your code, it's better (more maintainable, and better documented) to define it once as a variable and use that variable throughout. That way, you can easily change the variable by changing one value, or you can rename the variable using MATLAB's variable refactoring utilities.

Use sections to separate code execution. Sections are a double-edged sword. For one thing, they are very convenient for running a particular block of code, and they should be used to partition your code into logical segments. Being able to focus on and run a small bit of code, and inspect the workspace (environment) in the middle of your script is extremely helpful. On the other hand, it can cause problems that are addressed in the next bullet point.

Make sure your code runs in a clean environment. First of all, make sure your code compiles and runs. Submitting code that has multiple errors throughout is a big red flag, because coding is a feedback-driven problem-solving process: you are expected to run your code, and check that the results match your intuition. If your code is not in a state where it can be run, it shows that you haven't gone through this process. Additionally (related to sections), make sure that if you clear your workspace and run

all sections in your code from start to finish, that it runs correctly without errors. This is important because when you were debugging, you may have accidentally changed the behavior of your program.

Get familiar with standard debugging methods. Understanding error messages is key. That should help you handle syntactical errors. For semantic errors (when your code compiles but doesn't do what you think it does), start with a minimal, working example and build up from there step by step, until something begins to fail. This falls under the category of standard debugging tips that are common in software engineering.