**Computational Modeling & Simulation I (Interdisciplinary Science)**
**Colby College CS346, Spring 2020**

# CS346 Matlab Programming Style Notes & Guidelines

Here are some style notes and guidelines regarding Matlab programming on all assignments for CS346. Many of them are repeated from the guidelines included on HW 1, but some are new to provide additional guidance. (Also, note that some of the CS346 HW Notes & Guidelines document also refers to elements of Matlab programming for CS346.)

Many of these notes are also Generally Good Ideas regarding all programming, in any language! As always, feel free to ask me any questions about them. I hope you find them helpful!

## Comment Code for Easy Readability

All code must be appropriately commented! Readability and clear communication are always important for good programming projects, and even more so when writing code for interdisciplinary applications that involve non-programmers. At a minimum, every function or script should have a descriptive comment at its beginning (see more about this below). Comments should also be included in the bodies of programs when they improve clarity and readability.

It is **essential** that readers of code can *easily* tell the purpose of each variable in a program—this can be accomplished with well-chosen variable names and good commenting / documentation. On a CS346 assignment, if it is not immediately clear from the code and its documentation what a variable is for and how it is employed in a program, that program will not receive full credit—please document code in ways that make your code easy to reuse, maintain, and extend!

As part of this, programs that require *excessive* testing to determine correctness will not receive full credit—the code and comments / documentation should make it straightforward to determine if the code works correctly.

## Include Attribution Comments in Every File

In addition, at the beginning of every file, please include an attribution comment with your name, the date, and the context for which the file was created. As examples:

```
% Alan Turing
%
% myMachine.tm
% CS0 -- Creating Computer Science
% Fall, 1936
```

or

```
% engineAnalysis.adb
% Introduction to Computer Programming
% Fall, 1842
% Ada Lovelace
```

The precise formatting, whitespace, and content aren't the point—the essential part is some kind of comment indicating attribution, date of presentation / last modification, and context for the file. This will make it easier for code to be shared and re-used (or extended) in different contexts.

**Include Descriptive Comments for Every Script or Function**

In general, it is helpful for the descriptive comment at the beginning of each function and script to include a *contract*: a description of the input(s)—giving parameter names, their types, and what they represent / are used for—if any; a description of the output(s) and their types, if any; and a description of what the function / script does. This commenting style is helpful in many programming languages; even if you haven't used it before, please "try it on for fit" in CS346!

**Ensure Users Know How To Run Your Code**

Every script should have documentation stating what a user needs to do to run that script. (This can be either in documentation at the top of a script or in a separate write-up, or both.) If it's simply hitting F5 or typing the script name in the command window, please say so, but if other command-line arguments are needed or other instructions apply, they must be documented as well. If code cannot be straightforwardly run based on the provided documentation, it will not receive full credit in CS346.

**Use Named Constants; Avoid "Magic Numbers"**

As part of code style for CS346 exercises, please ensure that (unless explicitly instructed otherwise) code is presented in scripts that are easily readable, modifiable, and reusable in different experimental settings. As part of this, avoid "magic numbers"—using named constants instead of "magic numbers" can greatly simplify testing code with different key parameter values!

**Use Good Code Design (e.g., *Encapsulation*) when Creating Functions**

From a general perspective of good code design for re-usability, testability, etc.—and *encapsulation*, a term you may have heard when learning object-oriented programming—please think carefully before creating a *function* that does more than calculate and return a value. For example, in the Pi exercise on HW1, how would a function that both estimated $\pi$ *and* plotted the points used in the estimate fit with principles of encapsulation in code design?

Please feel free to ask me questions about all aspects of code design, including encapsulation!

**Use Spacing / Whitespace for Clarity and Readability**

For CS346, no line of code should be longer than 80 characters (with editor wrap-around off, if you're using an editor with that feature). Recall that an *ellipsis* (i.e., ...) can enable you to split a statement in Matlab over two lines in your editor.

Use whitespace (tabs, blank lines) for clarity and readability.

**Avoid `break`, `pass`, and Similar Statements**

Please do not use `break`, `pass`, or `continue` statements or related constructs that alter flow of control in loops—instead, redesign loops to make each *termination condition* explicit using boolean expressions in the loop statement itself. (There may be exceptions to this, such as a `switch` statement that requires `break` as part of its syntax, but as a general rule, avoid `break` and related statements in code.) I understand that there may be some controversy around this, but this style *does* improve readability and analyzability of code for many people, and I will ask you to use it for this class, as part of "trying it on for fit." As always, please see me if there are questions about the use of `break` statements or termination conditions in your code!

**Be Mindful of Floating-point Imprecision**

Please be mindful of floating-point imprecision! When determining if a number $x$ is within a range around another number (e.g., within 0.05 of 5), one common way to do this is to check if $abs(x - 5) \leq 0.05$. The absolute value function is commonly used, and very useful, in such circumstances.

**Ensure Figures are Presented as Intended, for Easy Readability**

Here are some guidelines for style regarding generating figures in your Matlab scripts:

- Please put a `figure` command explicitly opening a new figure window before your first `plot` statement. Without doing so, there is the risk of overwriting existing figures that a user of your code might have wanted to stay unaltered.
- Large headings for figures may not fit in the Figure window opened by the `plot` command. Please ensure any text put on figures will fit on a standard Figure window; use line breaks as needed.

**Avoid `clear` and `close` Statements in Scripts**

A `clear` statement (or a `close all` statement) in your code can make it impossible to inspect data generated by your code after it has finished its run, or it could inadvertently alter other data generated by the user of your code. Be sure not to put `clear` or `close` statements in your scripts for CS346! (If you think there is an exceptional case where a `clear` or `close` would be appropriate, please ask me about it!)

**Avoid Built-in Differential Equation Solvers**

One of the goals of this course is to emphasize problem solving skills involving building up solutions from basic tools. This is consistent with the CS346 restriction that we will not use anything more than plain Matlab—e.g., no additional libraries (e.g., Simulink)—in our programming. As part of this, do not use any Matlab differential equation solvers (*diff*, *dsolve*, *ode45*, etc.) or similar tools in your work for CS346. To learn how to create such tools and software libraries, not merely how to employ them, we'll be building our own differential equation solvers when we need them!

In general, good style and good coding practices matter, and the above guidelines apply to every programming assignment in this course. As usual, not following guidelines may result in deductions on an assignment. Please feel free to ask me any questions about coding style or CS346 guidelines or instructions!