ECE-210-B Homework #1

The Basics

Cat Van West, Spring 2024

Contents

In an effort not to visit the same horrors upon you as I did last year's students, we'll start with the *actual* basics: what MATLAB's good at, how to structure code, style points, and how to find & read documentation (the most important skill you can have). Everyone will get full credit for style on this assignment, as it's the first one and I haven't given any feedback yet. I *will*, however, comment on it, which I expect you to note for the next assignment! (Probably in an edited version of your source file.)

Assignments will follow the format implicitly laid out here: requirements and problems up front, supplementary (but probably required) reading appended. To submit, stuff everything into an organized `.m` file (or a few, though you shouldn't need more than one for this assignment), name it something logical (such as `lastname_hw#_date.m`) and submit through Teams. Same for future submissions. For some reason, Jacob wants everything on paper, which I think is insane for a programming class. Please do not tell him this.

1 The Assignment

Read the rest of this document – hopefully it helps with the coding, and it details style points that you'll want to remember. As for code… do the following. You may wish to read the MATLAB docs for a few of these!

Scale-'ers Create the following scalar variables (with these names – they'll be referenced later). These should all be doable in one line.

1. $a = |\sin(\pi/3) + {}^{j}/_{\sec(-5\pi/3)}|$ ($|\cdot|$ denotes absolute value)
2. $l = \sqrt[3]{8}$
3. $u = \sqrt{\frac{2}{6!}\sum_{n=1}^{80} n}$ (hint: use `sum` and the colon operator)
4. $m = \left(\mathrm{Im}\lfloor \ln\left(\sqrt{66}^{7j}\right)\rfloor\right)^2$ ($\lfloor\cdot\rfloor$ denotes the floor function. Note that floor and the natural log work on complex numbers!)

Mother...? Create the following vectors & matrices. Reference the above variables by name rather than writing the expressions again.

1. $A =$ (a column vector with those four scalars in any order)
2. $F = \begin{pmatrix} a & l \\ u & m \end{pmatrix}$ (bonus points if you index A to do it)
3. $T = F$-transpose (should be short!)
4. $B = TF$-inverse (verify this by multiplying B by TF)
5. $C = \begin{pmatrix} T & F \\ F & T \end{pmatrix}$ (should be a 4×4 matrix)

Cruelty Use `mean` to compute the mean of all four entries in B, as well as the row-wise means of C. Store the latter in a 4×1 column vector.

Odd Types Try evaluating $T + F$. Then $T + 1$. Then, just for kicks, $C + A$. What happens? Does this remotely make sense? Tell me your thoughts.

Not What It Seems... This MATLAB thing seems great for evaluating functions and manipulating matrices. What about something a little less... limited?

Create a (row or column) vector with $k = 3$ elements, with values evenly spaced between zero and one (think `linspace`). Square each individual element, take the sum, and divide by k. See what you get. Repeat this for $k = 5, 10, 300, 1e6$. Could you have predicted the value you're approaching? How?

A Style

For historical purposes, I've included Jon Lam's reminder on style (first given in a MATLAB seminar before my time), as it succinctly covers most of the important points:

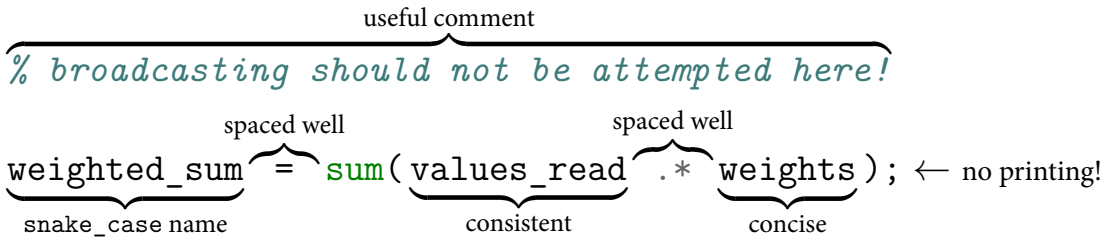
Remember, Good Code Style™ is important! Here are some recommendations, but feel free to do what suits you, so long as it is consistent and logical.

- Begin your scripts with `clc`, `clear` and `close all`. (Don't remember what these do? Use `help`!)
- Suppress outputs of intermediate values by ending the line with a semicolon. Long outputs printed in the command window are hard to follow. I prefer suppressing all outputs and storing answers in descriptive variables.
- Follow a convention for variable names. It can be `snake_case`, `camelCase`, `PascalCase`, `alllowercase`, etc. Names that are too short (e.g., `x`) are not descriptive, and `variableNamesThatAreTooLongLikeThis` become tedious. The exception to the first rule is when the name is clear from context, e.g., `x` and `t` to denote time series data, but even then it is usually nice to subscript them (e.g., `x_1` and `t_1` if you are working with multiple time-series). Be sensible!
- Long lines tend to be hard to read, especially on smaller screens. Try to limit lines to 80 characters. (MATLAB has a visual indicator for this.) To break an expression over multiple lines, use ellipses (`...`), e.g.:

```
this_is_a_long_variable_name ...
        = some_long_expression ...
        * another_long_expression;
```

- Use comments to explain code. (Recall that comments start with `%`.) The better and more consistently your variables are named, the less commenting you need to keep your code maintainable.
- Using sections and consistent spacing makes for easier reading/debugging. Section separators must have two percent signs at the beginning of the line, followed by the space, followed by the section title.

Here's a snippet of MATLAB code written in the above style:



There's more to say on style (of course), and we'll get to that as the semester progresses.

A.1 Arranging MATLAB Files

This should be more of a reminder – I'll go over this in class, and the lecture notes are formatted similarly to this. You can also deviate from this style if you believe it right – I provide these guidelines as, well, guidelines, so you have somewhere to start.

Begin the file with a preamble:

```
%%% A Title Befitting the Extraordinary Works You Have Produced
% Your Name, The Date

% A description, perhaps across multiple lines if the project was
% sufficiently complex, of what you have created.
close all; clear; clc;
```

This tells whoever reads it what it is and clears out the MATLAB environment in preparation for further works.

Each section of the file should start with a section designator (*%%*) to allow running sections individually. This will make your life easier, as you'll be able to run (and thus debug) sections of the file individually using MATLAB's **Run Section** button (under **Editor**). It's somewhat similar to notebook-style programming (e.g., Jupyter), if you're familiar with that.

A section might look like this:

```
%% The First Task

% (1) in the beginning...
x = linspace(0, 1, 1000);
y = sin(x);

% (2) ...there was Mathworks!
figure;
plot(x, y);
title('why did i bother putting a title here?');
```

For reference, this is what my submission for this assignment might look like:

```
%%%% ECE-210-B Homework #1 - The Basics
% Cat Van West, 9 Jan 2024

% Just the basics...
close all; clear; clc;

%% Scale-'ers
% ...

% (1) ...
a = ...;

% (2) ...
l = ...;

% ...

%% Mother...?
% ...

% and so on...
```

B Reading the Docs

There are two ways to get documentation: via `help` and via Mathworks' website.

The built-in `help` command yields text-format help for a given command. For example:

```
>> help log
 log    Natural logarithm.
    log(X) is the natural logarithm of the elements of X.
    Complex results are produced if X is not positive.

    See also log1p, log2, log10, exp, logm, reallog.

    Documentation for log
    Other uses of log
```

This documentation is pretty terse and is useful as a refresher if you've forgotten how to do something.

If you want more depth (or are learning a function for the first time), Mathworks provides extensive documentation for MATLAB at mathworks.com/help/matlab/. If you need help on anything from syntax to function arguments to design strategies, this is the place to go! StackOverflow is, of course, a wonderful resource too, but being able to read Mathworks' own documentation is a useful skill. For this assignment, you might check out the following pages:

1. the colon (`:`) operator: mathworks.com/help/matlab/ref/colon.html
2. the `sum` function: mathworks.com/help/matlab/ref/sum.html
3. the `mean` function: mathworks.com/help/matlab/ref/mean.html
4. the page for `floor`
5. the page for `imag`
6. …

…you get the idea. These pages are also available within the MATLAB interface itself via the built-in command `doc`, which takes arguments like `help` does.

B.1 How To Learn Something New

What do you do when you need to do something but don't know the first thing about it?

I don't have a magic answer for this. So much in software comes down to knowing where the right documentation lies, and, frustratingly, there's no one answer about how to find it. Knowing a few places to look is helpful – hence the references to Mathworks, `help`, and SO above – but that doesn't solve everything, and traditional software documentation is *not* written in a way that makes learning a library, a language, etc. from scratch very easy. I'm no expert in researching stuff in general, neither am I one in software, but here's what I do when I need something I don't have:

1. Realize I need new knowledge. This is more important than you might think and *I'm kinda bad at it*. Signs you might need to learn something are: your work seems harder than expected, you keep doing the same thing over and over, you suspect the problem you're working on is useful outside the bounds of what you're doing, you're bored – in gereral, someone might have encountered exactly what you're facing now and already worked out a solution to it.
2. Feed it into a search engine (or several – using Bing and Google simultaneously often gives non-overlapping results). But not just once: I often reword queries on the order of ten times before I find what I'm looking for. Pay attention to *language* – how do other people describe the problem you're facing? What words do they use that you could use in your next search?
3. If you find examples with links to the relevant documentation pages, great – use those. But *type them yourself* rather than copying them. This ensures you think about what you're writing – which, after all, is the point.
4. If you're left with software docs: good luck. More on that below.

Reading software documentation is a research art in itself. Most of the time, docs are set up heirarchically, starting with general packages and working their way down towards individual features. The "How To Read This Manual" section of the docs for GNU `make` gives what is often a decent approach: read the first few sections of whatever you find, looking for general information, and skip the technical details on the first pass. Look up language or concepts you don't know, either in the manual itself or on the wider internet. Learning from docs is, again, a skill, and an explorative process. It will take time.

Speaking of exploration: one unorthodox means of discovery is to simply read other people's code and look for things you don't understand. Don't be afraid of talking to them, either!