ECE-210-A Homework Feedback

Jonathan Lam

Spring 2022

1 Introduction to MATLAB

Submission format Please submit in *.m (plaintext) file format, rather than *.mlx. I should have made this a requirement at the beginning. While *.mlx (live script) files may be more visually appealling, they are more difficult to grade and may not be as cross-compatible with different versions of MATLAB.

Notes on style (in addition to the style notes on the assignment sheet) Name your files consistently (e.g., `ece210_hw1_lam.m`), and use a header at the beginning of each file. It may look something like:

```
% Homework 1
% Jonathan Lam
% ECE210
% 02/16/2022
clear(); clc(); close('all');

%% Q1
% ...
```

At the least, have your name and the commands to clear the workspace at the top of every script file[1].

`linspace()` **vs. the colon operator** Be wary of the use cases and what the problem specifies! One is better suited for the case of a specified number of samples, and the other is better for cases of a specified frequency or period.

[1] A script file is one you would run directly. This is as opposed to a file that defines a function or class, in which you do not want to call `clear(); clc(); close('all');` at the top of.

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. Learn how to build test cases and check the error from an expected or true value. Use the documentation until you get tired of it. This falls under the category of standard debugging tips that are common in software engineering.

3 Logical indexing and functions

Logical indexing is different than multiplying by a logical matrix The first question involves multiplying by a matrix of logical values, which acts like a layer mask: it preserves the domain, and sets all the values corresponding to FALSE to 0. In the second question, one needs to extract the x- and y-values that meet a certain condition. In this case, we want to condense the vectors to only those that match the condition; if we multiply by the logical matrix, we end up with a lot of spurious `[x, y] = [0, 0]` values. In other words, make sure you understand the difference between:

```
A = [ 1 2 3 ; 4 5 6 ];
B = logical([ 1 0 0 ; 0 1 0 ]);
masked = A .* B;
filtered = A(B);
```

Padding vectors Don't pad a vector with arbitrary values to make it match the length of another vector. In the case of extending the the result of `diff()` to make it match the length of the original vector, do not pad with zeros – this may create an anamolous discontinuity at the padded end. A more reasonable method is to pad with the first value at the beginning, or pad with the last value at the end. Alternatively, truncating the first or last value of the longer vector may be equally appropriate and doesn't involve introducing new datapoints.

There may be times when padding with zeros is acceptable, e.g., when using the FFT, which expects a zero-padded vector of length 2^n, $n \in \mathbb{N}$.

The `sinc` function As many of you discovered, the `sinc` function requires the Signal Processing Toolkit. With your school account, you may install this toolkit for free. (If you have trouble with installing a toolkit, please let me know.)

You may also implement the `sinc` function on your own. The formula for the (normalized) `sinc` is well-known:

$$\operatorname{sinc} x = \frac{\sin \pi x}{\pi x}$$

The problem is the discontinuity (due to a division by 0 which leaves an undefined value at $x = 0$), which will cause issues with your calculus functions. Since the limit is well-defined at $x = 0$, we can be more explicit:

$$\operatorname{sinc} x = \begin{cases} 1, & x = 0 \\ \frac{\sin \pi x}{\pi x}, & x \neq 0 \end{cases}$$

We may implement this in MATLAB like so:

```
x = % some vector
y = sin(pi * x) ./ (pi * x) ;
y(x == 0) = 1;  % note: logical indexing and broadcasting!
```

4 Gram Schmidt orthonormalization

An open-ended problem This assignment was the first (and only one) in this course that is really open-ended. You learned about an algorithm called Gram-Schmidt in Linear Algebra or in Signals and Systems. Now you're told to automate it. Take a specification and implement it. Word problems like this are common in computer programming.

You might wonder as you go through the process: *What is the most efficient way to do this? Should this part be vectorized? Can this part even be vectorized?* What you might not realize, however, is that programming is inherently an iterative process, and you won't know until you try it the first time. And then when you run your code and get results you don't expect, you debug it and refactor until you get a polished piece of code. I will also give you feedback when you submit your code, and you are allowed to resubmit code (for credit) because that's when you expect your code to improve. Don't expect to know what you're doing when you code; treat it as an interactive process.

To vectorize, or not to vectorize Gram Schmidt is an algorithm that cannot be totally vectorized. This is because it is a sequenced algorithm; the second row depends on the first row, and the third depends on the first two, and so on and so forth. The columns are not independent and thus the algorithm is not parallelizable. Thus we need a `for` loop somewhere in the algorithm. However, it turns out that you can vectorize both `ortho_proj` and `is_orthonormal`, because they both don't have this restriction.

Matrix multiplication as inner product A common definition for inner product of two (complex[2]) vectors is the following:

$$\langle x, y \rangle = \sum_i x_i^* y_i$$

Now, assume we have in MATLAB two column vectors `x` and `y`. We can write this inner product[3] as the expression `dot(conj(x), y)`. However, there's a simpler way: `x' * y`. The place where this really shines is if you have two matrices A and B (with the same interpretation as given in the problem set, where each column is a vector in the set), and want to take all the dot products $\langle A_i, B_j \rangle$ where A_i indicates vector/column i from matrix A. In this case, you can do this very simply:

$$C = A^H B$$

The beauty is that $C_{ij} = A_i^H B_j = \langle A_i, B_j \rangle$.

[2] Did you consider the case of complex vectors?

[3] In math notation, this would be $x^H y$, where ^{H} indicates the Hermitian transpose (conjugate transpose).

Vectorizing `is_orthonormal` Consider a set of vectors M. If the set of vectors is orthonormal, what do you expect the output of $M^H M$ to be? (Hint: see previous bullet point.)

Vectorizing `ortho_proj` The projection of a vector on a basis is the sum of the projections of the vector on each of the basis vectors. You can write orthogonal projection using the formula given in your first linear algebra class:

$$\text{proj}_{\{\vec{a_i}\}} \vec{b} = \sum_i \text{proj}_{\vec{a_i}} \vec{b} = \sum_i \frac{\langle \vec{a_i}, \vec{b} \rangle}{||\vec{a_i}||^2} \vec{a_i}$$

However, we can greatly simplify this for our `ortho_proj` case. First of all, if we assume that $\{\vec{a_i}\}$ is an orthonormal set of vectors, then $||\vec{a_i}|| = ||\vec{a_i}||^2 = 1$. The rest is simply three operations: an inner product, a regular (element-wise) product, and a sum. Can you vectorize the composition of these operations?

Additional error checking for Gram Schmidt Consider the case when there are more vectors in the set of vectors than the dimensionality of the space (the number of elements per vector). (This will be the case when M is wider than it is tall.) Then, some vectors will necessarily linearly dependent. This in turn will cause some of the vectors from Gram Schmidt orthonormalization to be zero. Do you handle this case? (Most submissions don't check if the norm is zero before normalizing, and they don't attempt to remove zero vectors from the basis during Gram Schmidt). Note that the assignment doesn't ask you to check these, but neither does it give you the assumption that the input vectors are necessarily linearly independent. (Luckily, a set of random vectors where M is at least as tall as it is wide will very likely be a linearly-independent set.)

5 Object-oriented programming and cell arrays

Not much to say about this assignment, it should be pretty straightforward.

6 Z-transform, polynomials, and basic filtering

An intuitive reading of the pole-zero plot Zeros near or on the unit circle cause increased attenuation (decreased gain), and poles near the unit circle (but hopefully not on the unit circle) caused increased gain. The rightmost part of the unit circle represents a zero frequency, whereas the leftmost part represents higher frequencies. We can use some combination of these facts to intuit what kind of filter (lowpass, highpass, bandpass, bandstop) a filter is by inspecting its pole-zero plot.