Lab 0: MATLAB Preliminaries

Welcome!

This course provides a theoretical and practical basis for the study of Signals & Systems. In the theoretical part of the course, you will be introduced to concepts ranging from complex numbers and Euler's formulas, to frequency-domain and complex-domain analysis of signals and the systems that process them. You'll be introduced to the digital filter, the sampling theorem, the Fourier series, and discrete and fast Fourier transforms.

As you'll see this week, the theoretical part of this course introduces concepts on a mathematical level. In the real world, you as a budding engineer will be asked to process real-world signals. To that end, the practical "lab" portion of this course aims to train you to apply your knowledge using the computational tools that are at your disposal. We ask you to apply your theoretical knowledge to solve problems; first computer-generated ones, then real-world ones. In future labs, you'll create sound, then arrange those sounds to make music. You'll learn about aliasing, then apply those principles to blur and de-blur images. You'll use filters to synthesize human voice. You'll use the Fast Fourier Transform to find the heart rate from a real-world electrocardiogram. By the end of the semester, we hope that you'll understand the theory behind all of these things, and be able to apply them in the real world. These labs can be a lot of work, but can also be a lot of fun. Be sure to start early!

A Note on MATLAB

These laboratory assignments will be done in a technical computing environment called MATLAB. You will need MATLAB access in order to complete them. If you don't already have MATLAB access, you should purchase a student version. We recommend that you purchase the version that includes toolboxes for $99.

MATLAB can also be accessed for free through UF's view.ece.ufl.edu remote login into UF computers. However, we recommend that you purchase the standalone version to use in case access to the internet is not available. As instructors, we are legally not allowed to endorse illegal downloading of MATLAB and other copyrighted software.

A Note on Lab Grading

By the time you finish this course, we expect you to have a beginner's working background knowledge for signal processing in the real world. Students that have completed this course have found the experience gained in it to be invaluable in their work as signal processing engineers for real-world companies. As such, our grading is oriented toward two aims. First, we look to see that you have completed your tasks. Second, we want to train you to reach a systems-level objective using many small theoretical steps. At the beginning of this course, we will ask you to follow specific steps en route to a solution. As the course goes on, we will expect you to come up with more and more of the intermediate steps by yourself. TAs will be present and available to assist you, but by the end of this course we want you to be comfortable breaking down real-world signal processing problems and solving them.

To that end, in your submissions, we need to see what you did. We want to see that you got the correct result, and that you came up with and thought about the intermediate steps on the way to your result. Show your code. **Show the output from the code**.*Explain your code*, preferably as comments (explained later). Answer any questions asked by the lab manual. If you made a plot, submit a screenshot of it. Do this in bite-sized pieces; i.e. code, answers, explanations, and plots for Section 1.1, then code, answers, explanations, and plots for Section 1.2, and etc., rather than code for the entire lab, answers for the entire lab, explanations for the entire lab, and plots for the entire lab. Submit everything for a lab in a single PDF document. In some labs, such as Lab 3, you will be asked to submit additional audio files using `audiowrite()` or `wavwrite()` commands. We will explain how to do that in that lab.

The objective of the lab portion of this course is to enhance your understanding of Signals & Systems, and to enable you to apply your knowledge to a variety of real-world applications. We will grade you strictly, and try to help you to improve. Since the flipping of this course, *no one* has ever earned a perfect score for the practical portion of this course. So please don't curse at your TA if you only earned 90 out of 100!

Pre-Introduction to Lab 0

By the end of Lab 0, make sure that you understand basic MATLAB commands and syntax, including the `help` system. For this subsection ((a) through (e)), you do not need to submit anything. It is just for your education; we will expect you to understand how to initialize a vector, for instance.

(a) Walkthrough an introduction to MATLAB by typing `demo` into MATLAB and selecting the "Matlab Introduction." It will show you some basics of the tool

(b) Experiment with the capacity of MATLAB's `help` function. Run the following lines, for yourself (they demonstrate how to use various matlab functions):

```
>> help plot
>> help colon
>> help ops
>> help zeros
>> help ones
```

(c) Read about loops and nested loops on the following MATLAB help pages:

```
>> help for
>> help while
>> help until
>> help if
>> help else
>> help function
```

(d) Learn to write and edit your own script files in MATLAB, and run them as commands. MATLAB comes with a variety of videos and scripts to demonstrate different functions and capabilities. Type `demo`, search "script" (in the top search bar), and explore the listings; these tools will come in handy later.

(e) Learn a little about advanced programming techniques for MATLAB, i.e., vectorization. If you don't know what that is, Google it. I mean it, type "MATLAB vectorization" into Google and click on the first non-advertisement link. It should be a page from Mathworks, the company that produces MATLAB, teaching you about vectorization. Read it now; it will help you synthesize what you have learned from the `help` pages, and will be helpful later in this lab (this googling technique will come in handy for later labs too!).

Lab Part One

1.1 Getting Started with MATLAB Basics

(a) Pay attention to what the variable `ans` points to, execute and explain the following:

```
>> 6^2
>> ans
>> ans/6
>> ans
```

What is the purpose of the MATLAB variable `ans`?

(b) MATLAB can be used as a very expensive calculator. Execute and explain the following lines:

```
>> pi*pi - 10
>> sin(pi/4)
>> ans^2
```

You can assign variables in MATLAB, just like in other programming languages. Execute the following (again: submitting code, output, and explanation), and watch the area labeled "workspace" on the side of your screen. The variables will appear there as they are assigned.

```
>> x = sin(pi/5);
>> cos(pi/5)
>> y = sqrt(1 - x*x)
>> ans
```

What is the numerical value of `x`?

1.2 Colon Operators

Reread and study the `help colon` page. The colon creates a series of values, from start value to end value, stepped by a step value whose default is 1.

(a) The colon notation is ***very important*** in MATLAB, and you need to make sure you understand it in order to be successful in future labs. Execute and explain, in words, what each of the following lines of code does:

```
>> jkl = 0:6
>> jkl = 2:4:17
>> jkl = 99:-1:88
>> ttt = 2:(1/9):4
>> tpi = pi*[0:0.1:2];
```

(b) With this colon notation, extracting/inserting numbers into an array is made easy with MATLAB. Explain the results of the last four lines of the following code:

```
>> xx = [zeros(1,3), linspace(0,1,5), ones(1,4)]
>> xx(4:6)
>> size(xx)
>> length(xx)
>> xx(2:2:length(xx))
```

(c) Execute the following lines of code and try to understand what is happening. As usual, include code, results, and explanation of what is happening in your lab report:

```
>> yy = xx;
>> yy(4:6) = pi*(1:3)
```

Now, after learning how the above code works, write some code that will take the vector `xx` and take the elements with an even index `{xx(2), xx(4), …}` and replace them with pi^pi (that's taken to the exponent). You need to do this in one line of code using vector replacement, not by using a loop.

(d) Remember when I had you read about vectorization? Instead of running a (very slow) loop, in many cases you can simplify your MATLAB code through vectorization. Functions like `exp()` and `cos()` are defined for vector inputs:

```
>> cos(vv) = [cos(vv(1)), cos(vv(2)), cos(vv(3)), …, cos(vv(N))]
```

where `vv` is an N-element row vector. The following code is an example:

```
M = 200;
for k = 1:M
      x(k) = k;
      y(k) = cos(0.001*pi*x(k)*x(k));
end
plot(x, y, 'ro');
```

Barukh Rohde University of Florida Spring 2017

can be replaced with the following 3 lines, with the same result:

```
M = 200;
y = cos(0.001*pi*(1:M).*(1:M));
plot(1:M, y, 'ro-')
```

What is the purpose of the dot before the asterisk in line two of the above?

Now use this same idea to condense the following code to 2 or 3 lines, removing the `for` loop:

```
N = 200;
for k = 1:N
    xk(k) = k/50;
    rk(k) = sqrt(xk(k)*xk(k) + 2.25);
    sig(k) = exp(j*2*pi*rk(k));
end
plot(xk, real(sig), 'mo-')
```

You will submit your code, a screenshot of your plot, and an explanation.

Vectors in MATLAB are sets of numbers, much like an array in other programming languages. Execute the following line of code:

```
>> xk = cos(pi*(0:11)/4)
```

Explain what this command stores into `xk`.

What is `xk(1)`?

Notice that `xk(0)` is not defined. Why is this?

(e) MATLAB can handle loops, just like any other programming language; however, loops are not very efficient in MATLAB. Colon notation is much faster, and takes fewer lines of code. The code below computes values of cosine. Vectorize the code to remove the loop:

```
yy = [ ];
    for k = -5:5
        yy(k + 6) = cos(k*pi/3)
    end
yy
```

Explain why the original code uses `yy(k + 6)` instead of just `yy(k)`. What would happen if we used `yy(k)`?

(f) Plotting is made easy in MATLAB, whether the numbers are real or complex. The `plot` command plots a vector `y` versus a vector `x`, connecting points with straight lines. Run the following code:

```
>> x = [-3 -1 0 1 3];
>> y = x.*x - 3*x;
>> plot(x, y)
>> z = x + y * sqrt(-1)
>> plot(z)
```

Write a short explanation, in your own words, explaining dot and matrix multiplication and the MATLAB representation of each.

(g) Write a function called `oddsummer.m` that takes a positive integer `n` as its argument, and returns the sum of all odd numbers between 1 and `n`. Submit your code, and some examples of your function being run from the command line with different values of `n`.

(h) Write a function called `hellos.m`, taking a positive integer `h` as its argument and displays the word "hello" `h` times. Submit your code, and some examples of your function being run from the command line with different values of `h`.

Lab Part Two

In the modern world, a program is more than just a series of instructions to a machine. It is a collaborative project between multiple people to send better instructions to a machine. How do programmers communicate within a program? Through comments and variable names that explain the purpose of each line or section of code. In MATLAB, a comment is inserted using the percent symbol `%`.

In the following lab exercise, you will not just write a program. You will write an easy-to-understand program to create something called a magic square. You will use descriptive variables, such as the position variables `x` and `y`, the iterator `i`, and the delimiter `n`. You will place comments after each line and subsection, describing the functionality of your code. This is an easy exercise, with a simple way to check your answer - the built-in MATLAB function, `magic(n)`. Code the below algorithm using loops, and to explain your code well through comments:

A magic square is a square of numbers in which the sum of each column, the sum of each row, and the sum of each diagonal is the same. A description is found at

http://en.wikipedia.org/wiki/Magic_square.

A magic square is best obtained using the Siamese method,

http://en.wikipedia.org/wiki/Siamese_method.

Write a function that uses this method, iterating through a matrix, to create a magic square of size `n`, where `n` is an odd number equal to or greater than 3.

Start by making a matrix using the `zeros()` command, and create a for loop using iterator `i`. Initialize `x` and `y`, and iterate through the loop as follows:

The goal here is to create bounds to contain the incrementing of the magic square. You can do this by making two position variables `x` and `y` as part of a loop with iterator `i`. You will start at `x` position `(n-1)/2 + 1` and `y` position as 1. Place a 1 there. Move upward and to the right, subtracting 1 from `y` and adding 1 to `x`. If there is a number there (i.e. if the number in that position is nonzero - use an `if` statement here), move back, and move one square down instead. If there is a number there, move down until an empty space is reached. If the edge of the square is reached - use an `if` statement here too - then wrap around to the opposite edge of the square.

Come see us in office hours, email us, and talk to us! We are here to help.

Preparation for Next Week

Download and install the SPFirst toolbox, found on the author's website:

spfirst.gatech.edu/matlab

And have a great first week of class!