Lab 0: MATLAB Preliminaries

Introduction

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 development platform from Mathworks called MATLAB. The labs are written for MATLAB's stable version R2017b , although most versions within recent years will also work. MATLAB can also be accessed for free through UF's https://view.ece.ufl.edu and https://apps.ufl.edu remote login into UF computers. However, we recommend that you download the standalone newest version to use in case access to the internet is not available.

Installing MATLAB

Mathworks offers student version of MATLAB that can be purchased at this website. There are different toolboxes that contains application specific optimized APIs for development. In our case, the only toolbox needed is the **Signal Processing Toolbox**

There are 2 purchasing options:

- If you only want the Signal Processing Toolbox, the MATLAB core would cost 50 dollars and the Signal Processing Toolbox would cost 10 dollars, total of 60 dollars.
- Mathworks offers the MATLAB core along with all available toolboxes for 100 dollars.
- We suggest also purchasing the symbolic math toolbox at this time. It will be used in other courses and is cheaper if purchased along with MATLAB

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:

- We look to see that you have completed your tasks.
- 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.

This semester we will expect the lab documents to be submitted using MATLAB publisher resulting in a PDF for each section. An explanation on how to use the tool can be found at: https://www.mathworks.com/help/matlab/matlab_prog/publishing-matlab-code.html?fbclid=IwAR3REw_0GAC1lzx7QK9Z_UkL3O-9Rki-QS85YtL8En6eK3Bvbg2gFjDN4WY. Use %% to separate the code into sections and fprintf() to provide any answers to the questions that are asked. Click publish and be sure to save as a PDF. **BE SURE to submit Part 1 and Part 2 separately**

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.

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

2. Experiment with the capacity of MATLAB's help command. This command provides an overview including purpose, input arguments, and output arguments of a function. Run the following lines in command window:

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

3. Read about loops, conditions, and functions on the following MATLAB help pages:

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

4. 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.

5. Learn vectorization techniques via MATLAB's documentation.

1. Lab Part One

1.0 Download the SPFirst Toolbox

1.1 Variables

1. *Execute and explain the following commands* (pay attention to what variable ans points to):

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

2. MATLAB can be used as a very expensive calculator. *Execute and explain the following commands:*

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

3. Just like in other programming languages, you can assign variables in MATLAB. *Execute and explain the following commands* 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
```

Question 1: What is the numerical value of x and y ?

1.2 Vectors and Colon Operator

Read the help colon page. In essence, the colon creates a series of values, from start value to end value, incremented by a step value whose default is 1.

1. *Execute and explain the following commands:*

```
>> x = 0:6
>> y = 2:4:17
>> z = 2:(1/9):4
>> t = pi*[0:0.1:2]
```

2. With this colon notation in MATLAB, extracting/inserting numbers into a vector is made easy. *Execute and explain the following commands:*

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

   Execute and explain the following commands:

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

3. Now, after learning how the above code works, *Write one line of code* (using vectorization) that will take the vector xx and take the elements with an even index {xx(2), xx(4), …} and replace them with π^{π}(that's π to the exponent π).

1.3 Loops

Just like any other programming language, MATLAB can handle loops. However, loops are serial operations which are not nearly as efficient as parallel operations like vectorization. Despite inefficiency, loops are inevitable when operations are cascaded, i.e. when output of one operation is dependent on the output of another operation.

1. Read help for page. *Execute and explain the following commands:*

   ```
   x = 0;
   for k = 1:4
   x = x+1;
   end
   ```

2. Using for loop, generate a vector containing the following elements: 1^1, 2^2, 3^3, …, 9^9. *Write code and explain*

3. In many cases, code with loops can be optimized by vectorization. Functions like exp() and cos() are defined for vector inputs.

   Example:

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

   The piece of code above can be optimized as:

```
M = 200;
y = cos(0.001*pi*(1:M).*(1:M));
```

Now use this same idea to optimize the following code by removing the for loop. *Write 2 or 3 lines of code, plot, and explain*:

```
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
```

Question 2: What's the value of x after the code executes?
Question 3: What is the purpose of the dot before the asterisk in line two of the above?

1.4 Plot

Visualization is crucial in signal analysis. In MATLAB, the plot() function takes in a vector x and a vector y , where each corresponding entry of x and y forms one coordinate point (x,y) . All the points are put on a graph and connected with straight lines.

1. *Execute and explain the following commands*

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

*Question 4: What is the difference between a*b and a.*b, where a and b are matrices?*

1.5 Functions

1. Write a function called oddsummer 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.

2. Write a function called hellos, taking a positive integer n as its argument and displays the word "hello" n times.

   Submit your code, and some examples of your function being run from the command line with different values of n.

1.6 Comments

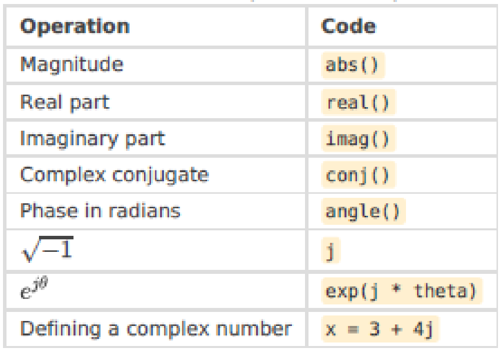
In general, commenting is a good practice of coding. Comments are text that are ignored by the computer, i.e. not compiled. Comments can be used to explain sections of code so that other people (sometimes even yourself) can interpret the functionality of that code. In MATLAB, a comment is inserted using the percent symbol %. **Example:**

```
here_be_code = 123 % here's a comment
more_code([1, 2]) % comments don't get evaluated
```

1.7 Complex Numbers in MATLAB

Before proceeding in this section, make sure that you have watched the lecture video on Euler's formula, Ae^(jϴ)= cos(ϴ) + jsin(ϴ) Common MATLAB complex number operators

Common MATLAB complex number operators



Operation	Code
Magnitude	`abs()`
Real part	`real()`
Imaginary part	`imag()`
Complex conjugate	`conj()`
Phase in radians	`angle()`
$\sqrt{-1}$	`j`
$e^{j\theta}$	`exp(j * theta)`
Defining a complex number	`x = 3 + 4j`

1. MATLAB reserves the letter i or j as $\sqrt{-1}$. In this class, the letter i is usually used as iterator of loops. Hence, the letter j will mostly be reserved as $\sqrt{-1}$.

```
>> z = 3 + 4*j, w = -3 +4*j % Declaring complex numbers
>>real(z), imag(z) % Real and imaginary components
>>abs([z,w]) % Computes magnitudes
>>conj(z+w) % Computes complex conjugate of the sum
>>angle(z) % Computes the phase of the complex number
% Also written as atan2(imag(z)/real(z))
>>exp(j*pi) % Using Euler's formula, cos(pi)+j*sin(pi)
>>exp(j*[pi/4, 0, -pi/4])
```

2. MATLAB can compute complex valued formulas and display the results as phasor diagrams. Use the following zvect function to plot five vectors all on one graph. *Execute and plot:*

```
>> zvect([1 + j, j, 3 -4*j, exp(j*pi), exp(2j*pi/3)])
```

3. Use $z_1 = 10e^{-j\frac{2\pi}{3}}$ and $z_2 = -5 + 5j$ for all parts of this section.

   a. Plot z1 and z2 using zvect and print them with zprint.

   b. *Execute and explain:*

zcat([1j, -1, -2j, 1]).

c. Compute $z_1 + z_2$, and plot the sum using zvect.
Using hold on, *Plot all three vectors (, ,) on the same plot* where and are concatenated using zcat .
Display the numerical values of $z_1, z_2, z_1 + z_2$

d. Compute the product and *Display the numerical result.*
Plot $z_1, z_2, and\ z_1 z_2$ on the same plot.

e. Compute the quotient $\frac{z_2}{z_1}$, *Display the numerical result*

Plot it on a plot with $z_1, z_2, \frac{z_2}{z_1}$

f. Compute the conjugates $z_1\ and\ z_2$ *Display the numerical results and plot them.*

g. Compute $\frac{1}{z_l}\ and\ \frac{1}{z_2}$ *Display the numerical results and plot them.*

4. Create a script file called mylab1.m containing the following code:

```
tt = -1:0.01:1;
xx = cos(5*pi*tt);
zz = 1.4*exp(j*pi/2)*exp(j*5*pi*tt);
plot(tt, xx, 'b-', tt, real(zz), 'r--')
grid on
title('TEST PLOT OF A SINUSOID')
xlabel('TIME (sec)')
```

Question 5: What does zcat() do with a vector of complex numbers?
Question 6: What is the relationship between the two initial angles and the angle of the product?
Question 7: Why is the plot of real(zz) a sinusoid even though no cos or sin is present in its equation?
Question 8: What are its phase and amplitude? Calculate the phase based on a time-shift measured from the plot. Take a screenshot of the plot.

1.8 Sounds

1. Run the MATLAB sound demo by typing xpsound. If you don’t hear anything, check your speakers and volume settings. If running this through UF Apps, you will not be able to hear this demo.

Sounds are represented by sinusoids, so to create sound using MATLAB, you need to create a sinusoid. Your script mylab1.m creates a sinusoid with frequency 2.5 Hz. *Modify your script* so that it produces a 2000 Hz sound, with sampling frequency 11025 Hz, which is 0.9 seconds long. The appropriate time vector is therefore tt = 0:1/11025:0.9;

Now use soundsc() (type help soundsc) in order play the sound. If you are using UF Apps, you will need to save the audio file in order to hear it. Type help audiowrite to find out how. *Play your generated sound for your TA*, who will check you off on canvas.

Question 9: What is the length of your tt vector?

2. Lab Part Two

2.1 Generating Sinusoids

Write a script to do the following steps. Include this script in your submission.

1. *Create a time vector t that is two cycles of the 5000 Hz sinusoid defined in (b).* (Hint: define t similar to in Section 1.3) Set T equal to the period of the sinusoid and define the start of t to be –T, and the end to be +T.

   Make sure that the sine wave has at least 25 samples per period. In other words, when you make the time vector, make sure the step size is small enough to generate 25 samples per period.

2. *Generate two 5000 Hz sinusoids:*

$$x_1(t) = A_1 \, cos\left(2\pi(5000)(t - t_{m_1})\right)$$

$$x_2(t) = A_2 \, cos\left(2\pi(5000)(t - t_{m2})\right)$$

   Let A_1 be equal to your age and set $A_2 = 1.2A_1$. If D and M are the day and month of your birthday and T is the period of the sinusoid, set $t_{m_1} = \frac{37.2}{M} T$and $t_{m_2} = \frac{-41.3}{D} T$.

3. *Create a third sinusoid that is the sum:*

$$x_3(t) = x_1(t) + x_2(t)$$

4. *Plot all three sinusoids* over the range $-T \leq t \ \leq T$. Use subplot(3,1,1) and subplot(3,1,2) and subplot(3,1,3) (see help subplot). Put a title on each subplot, and include your name in one of the titles (see help title, help print, and help orient)

5. *Write a function to generate a sinusoid*, by using four inputs: amplitude, frequency, phase, and duration. The function should have two outputs: the values of the signal and the corresponding times at which values are known. The function should generate exactly 25 values of the sinusoid per period. Call the function one_cos() . Use your answer to 1.3 of part 1 of this lab as a starting point.

Submit the code, and plot the output for the following: $A = 95, \omega = 200\pi \frac{rad}{s}, dur = .025s, \phi = 5rad$

Question 10: What is the expected period in milliseconds?

2.2 Analyzing Sinusoids

Remember from lecture that given two out of three values – the phase, the frequency, and the time location of a peak – the third value can be calculated.

So, given a frequency, the phase of a sinusoid may be calculated by measuring the time location of a positive peak.

1. *Measure the time-location* of a positive peak and the amplitude of the plots of $x_1(t)$and $x_2(t)$.

   Calculate by hand the phases of the two signals by converting each time shift, to phase, using the known frequencies of $x_1(t)$and $x_2(t)$.

   (Hint: use data cursor)

2. *Measure the amplitude* A_2 and the time-shift t_{m3} of $x_3(t)$ from the plot and *calculate the phase by hand.*

   Annotate the plot to show how the time-shift and amplitude were measured, and how the phase was calculated (use something like MS Paint).

3. Use phasor addition of the complex amplitudes for $x_1(t)$and $x_2(t)$ to determine the complex amplitude of $x_3(t)$. Use this answer to verify that your previous calculations of A_3 and ϕ_3 were correct. *State and explain your percent error.*

2.3 Complex Amplitude

1. *Write one line of code* that will generate $x_1(t)$ above by using the complex-amplitude representation:

$$X_1(t) = Re\{xe^{j\omega t}\}$$

2.4 Concatenation

Concatenation two arrays means chaining them together. For sound, this means putting signals together to hear them one at a time. In MATLAB, two vectors can be concatenated through the notation:

```
zz = [xx, yy];
```

Run a script or function for each example for your lab doc.

1. *Run soundsc() on an input signal/sinusoid*. Using $f_s = 11025\frac{samples}{s}$, instantiate a time vector of 0.5 second duration. Create a vector x1 of samples of a sinusoid with amplitude $A = 100$, angular frequency $\omega = 2\pi(800)$, and phase $\phi = -\frac{\pi}{3}$. Play the vector and listen to the output.

   Note: if soundsc() does not work for you, in this case use audiowrite() (run help audiowrite) to create a .wav file. The .wav file can be played by most media players.

2. *Run soundsc() on a second input signal/sinusoid*: Create a vector x2 of samples of a sinusoid with $A = 80$, $\omega = 2\pi(1200)$, and $\phi = \frac{\pi}{4}$. Use $f_s = 11025\frac{samples}{s}$ and obtain a number of samples equivalent to a 0.8 second duration. Play the vector and listen to the output.
3. *Concatenate the input signals together*: Concatenate x1 and x2 into a vector x, leaving a duration of 0.1 seconds of silence in between.

   Hint: use zeros().

   Listen to this new signal to verify that it is correct.

4. Plot and identify the two input signals. *Execute the following commands, and plot:*

   ```
   >> tt = (1/11025)*(1:length(xx));
   >> plot(tt,xx);
   ```

   This will plot a huge number of points, but you should be able to discern the "envelope" of the signal. Verify that the amplitude changes from 100 to 0, and then from 0 to 100 at the correct times.

5. Different sampling frequencies affecting pitch: Call soundsc() on x again (or use audiowrite() and play the output file), but this time use $f_s = 22050\frac{samples}{s}$.

   Describe how the duration and pitch of the signal were affected and explain.

6. Using the t vector from section 2.1.4 of this lab, create sounds for the following frequencies:

   [523, 587, 659, 698, 784, 880, 988, 1047]

   Concatenate the sounds together, and *Submit the concatenated sound (a musical scale) as a single .wav file using audiowrite.

   Remember to include code and .wav file.

Question 11: How does this sound compare to the output of the previous sound?

2.5 Magic Square

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). I want you to 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 magicsquare. 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, talk to us! We are here to help.

Preparation for Next Week:
Download and install the SPFirst toolbox, found on the author's website:
dspfirst.gatech.edu/matlab

And have a great first week of class!