EE3TP4: Signals and Systems
Lab 1: Introduction to MATLAB

Objective

To help you familiarize yourselves with MATLAB as a computation and visualization tool in the study of signals and systems.

Report

No report is required for this lab. However, you are required to demonstrate to a TA that you have completed each of the numbered parts of this lab in order to obtain your mark.

Introduction to Matlab

Obtain a login name and password for your pairing from the TAs. Record this information, as you will need it for future labs. Log into the computer. Observe that you have two special disk drives. First, the Z: drive which belongs to your account only. You can read and write to this drive. You also have a Y: drive which you can read but to which you cannot write.

Read the brief introduction to MATLAB provided in the Appendix. Try out the examples discussed there. Once you have familiarized yourself with MATLAB in this way, you may attempt the following numbered exercises.

1 Sines and Cosines

In this section we will write a MATLAB function file called `gensin.m` which will produce samples of a sine wave of the form

$$y(t) = A\sin(2\pi f_0 t + \phi).$$

The input arguments should be the frequency of the sine wave in Hertz, f_0, the amplitude A, the initial phase ϕ, the frequency of the samples in samples-per-second, f_s, the time of the initial sample in seconds, t_0, and the number of samples to be taken, N. There should be two output variables, the first being a vector of the times at which the samples were taken, and the second a vector of the samples. The n-th element of this second vector should be the value of $y(t)$ when the value of t is that given by the n-th element of the first output vector. That is, if $T_s = 1/f_s$, then

$$y[n] = y(t + (n-1)T_s) = A\sin(2\pi f_0(t_0 + (n-1)T_s) + \phi), \quad \text{for } n = 1, 2, \ldots, N.$$

Since the MATLAB `sin` operator applied to vectors as well as scalars, the following is a possible implementation of `gensin.m`.

```
function [tt,yy]=gensin(fo,A,phi,fs,to,N)

Ts=1/fs;
tt=to + (0:1:(N-1))*Ts;
yy=A*sin(2*pi*fo*tt+phi);
```

Now use this function file to plot some sine waves. (Type `help plot` for information on plotting.) Set $f_s = 8192$ samples-per-second, and choose several f_0's between 200 Hz and 2000 Hz. Note that `plot` uses linear interpolation between the specified points. That is, it 'joins the dots' with straight lines.

Try playing some of these sinusoids using the `sound` or `soundsc` command. Please keep the duration of your sine wave to at most one second. If the tone persists, press `<CTRL-c>`.

2 Not All Discrete-Time Sinusoids are Periodic

Use the following sequence of commands to examine the nature of the periodicity of the discrete-time sinusoids you have generated. (You may want to save them in a script file.) Consult the help file on functions which you do not recognize. We will use the `stem` command to do the plotting in this case to emphasize that what we really have is a discrete-time signal.

```
figure
[tt,yy] = gensin(8192/16,1,0,8192,0,64);
subplot(2,1,1);
stem(tt,yy);grid;
[ttt,yyy] = gensin(8192/(4*sqrt(17)),1,0,8192,0,64);
subplot(2,1,2);
stem(ttt,yyy);grid;
```

Observe that the upper discrete-time sinusoid is periodic, but the lower one is not. You may need to click on the middle button on the top right corner of the window to make the figure occupy the whole screen in order to see the differences. Click on the same button to return the window to the standard size.

3 Complex Exponentials

Consult `help` on `exp`, `real`, and `imag` for this section.

1. Write a function similar to `gensin` which generates (samples of) the complex exponential signal $Ae^{j(\omega_0 t+\phi)}$ for $A = 3$, $\phi = -0.4\pi$ and $\omega_0 = 2\pi \times 1250$. Choose a duration which will cover three periods.

2. Type `figure` to generate a new figure window, and plot the real and imaginary parts on aligned graphs using `subplot(2,1,1)` and `subplot(2,1,2)`.

3. Verify that the real and imaginary parts are sinusoidal, and that they have the correct frequency, amplitude and phase.

4 Generation of Square Wave from its Fourier Coefficients

1. Consider a square wave of period ten milliseconds:

$$g_p(t) = \sum_{m=-\infty}^{\infty} g(t - m\,10 \times 10^{-3}),$$

   where

$$g(t) = \begin{cases} 1, & 0 \le t < 5 \times 10^{-3}, \\ -1, & 5 \times 10^{-3} \le t < 10 \times 10^{-3}. \end{cases}$$

2. The Fourier Series of such a square wave can be written as

$$g_p(t) = \frac{4}{\pi} \sum_{k=1}^{\infty} \frac{\sin\bigl(2\pi(2k-1)100t\bigr)}{2k-1}.$$

3. Calculate and plot (samples of the) approximations

$$\hat{g}_p^{(K)}(t) = \frac{4}{\pi} \sum_{k=1}^{K} \frac{\sin\bigl(2\pi(2k-1)100t\bigr)}{2k-1}$$

   for various values of K, and verify that it does indeed converge to a square wave. Choose a sampling frequency of 8192 samples-per-second. Play the square wave and its approximations if you can. Can you hear the difference?

4. Does the approximation really converge? It seems that something funny is happening at the edges. What is going on? The convergence of the Fourier Series is in the sense that the energy of the error between $\hat{g}_p^{(K)}(t)$ and $g_p(t)$ in one period goes to zero as K gets large. This does not necessarily mean that $\hat{g}_p^{(K)}(t) - g_p(t)$ goes to zero at a discontinuity.

5 Sinc function

Use the MATLAB `sinc` function to plot (samples of) the 'sinc' function

$$\text{sinc}(t) = \begin{cases} \frac{\sin(\pi t)}{\pi t}, & t \ne 0 \\ 1, & t = 0 \end{cases}$$

for $-10 \le t \le 10$. Note the position of the zero crossings, and the decay of the "ripples". Plot $|\,\text{sinc}(t)|$ against t and $1/|t|$ against t on the same graph, and show that the ripples decay as $1/|t|$ when $|t|$ gets large. You will find the MATLAB commands `hold` and `axis` to be quite helpful in this task.

Appendix: Brief Introduction to Matlab

Starting Matlab: Open the MATLAB command window by double clicking on the MATLAB icon. This text-based interface will be the way you will control most of MATLAB's functionality.

Matlab as a calculator: You can use the MATLAB command window as a calculator, with basic operations `+, -, *, /` and `^` representing addition, subtraction, multiplication, division and exponentiation (raising a number to a given exponent), respectively. For example, type `4 + 6 + 2` and press `<Return>`. Observe that you can edit previous commands by using the up and down keys.

Matlab as a programming environment: You can also save the values of the variables as you go. For example, type `a=2; b=6; c=2; d=a+b+c;` To see your answer, type `d` and press `<Return>`.

Note that variable names are case sensitive, can contain up to 31 characters, must start with a letter and cannot contain punctuation. Note also that commands which are terminated by a semicolon will not produce any displayed output in the command window, whereas commands which are not terminated by a semicolon will display the results in the command window.

Complex numbers: The variables `i` and `j` are predefined to be $\sqrt{-1}$. Try not to use them as counters. For example, type `1-2i` and press `<Return>`.

Online help: MATLAB has an extensive on-line reference. To view information about a given command, type `help <command name>`. To do a keyword search, type `lookfor <keyword>`. Most commands have rather intuitive names, as we will see below. To determine how to stop the display from overflowing the command window, type `help more`.

Mathematical functions: Most standard mathematical functions have standard names. For example, type `help <command name>` for the following commands: `cos, sin, atan, exp, log, sqrt, abs, angle, conj, real, imag`. Note that most of these functions can be applied directly to vectors, as well as to scalars.

Constructing vectors and matrices: Vectors and matrices are most conveniently formed in a row-wise fashion, with brackets delimiting the matrix construction commands, commas separating the elements of the row, and semicolons separating the rows. For example, type `[1,2,3;4,5,6;7,8,9]` and press `<Return>`.

Special vectors and matrices: MATLAB provides some convenient routines for producing vectors and matrices with special structure. To discover some of these, consult the help for `colon, punct, linspace, logspace, zeros` and `ones`.

Indexing of elements of vectors and matrices: The first element of a vector is indexed by 1, and the top left corner of a matrix is indexed by (1,1). For example, type `A=[1,2,3;4,5,6;7,8,9]` and press `<Return>`. Then type `A(1,1)` and press `<Return>`. Repeat for `A(1,2)`, `A(2,1)` and `A(3,3)`.

Displaying and listening to results: MATLAB has some convenient routines for displaying results. Consult (first paragraphs of) the help on `plot`, `subplot`, `axis` and `hold`. You may also want to listen to some of the waveforms you generate. Type `help sound` or `help soundsc` for information on how to do this.

Flow control: MATLAB has many of the flow control functions of standard programming languages, such as C. Type `help for`, `help while`, and `help if` for information on for loops, while loops and conditional execution statements, respectively.

Aborting execution: A command can be aborted by typing `<CTRL-c>`. That is, by holding down the `<CTRL>` key and pressing `c`.

Clearing variables: You can clear the current list of saved variables using the `clear` command.

Matlab Script and Function Files

For simple problems, it often suffices to use MATLAB like a calculator, but for more complicated problems it would be more convenient to store the sequence of instructions in a (text) file, and simply type the name of that file. This is exactly what the MATLAB script and function files allow you to do. (MATLAB is an interpretive environment, so these commands are interpreted command-by-command each time you invoke the file. There is no need for external compilation and linking as in compiler-based languages, such as C.)

The MATLAB script and function files are text files and should be saved with a '`.m`' extension. For this reason they are often called m-files. You can use your favorite text editor to write these files, but on a Windows machine the simplest thing to do is to choose New from the File menu above the MATLAB command window, and select M-File. This will invoke a simple text editor with the standard Windows 'look and feel' which will suffice for our labs. You can add comments to these files by inserting `%` signs. Any text on a given line to the right of a `%` sign is ignored by MATLAB.

Script files are simply a collection of MATLAB commands which are executed sequentially. The commands operate on the variables in the current workspace, just as if they were typed at the prompt. Once you have typed the sequence of commands in the text editor, and have saved the file, you can run the script file by just typing the filename, without the '`.m`' extension, at the prompt. Note that MATLAB will have to have the folder in which the file is saved on its search path. You can add a folder to the search path by using the `editpath` command.

Function files are different from script files in that they have a formal set of input variables, a formal set of output variables, and work on a variable space which is independent of the main workspace. As such, they are much like 'function' subroutines in standard programming languages, such as C. You can create and save function files in the same way as script files. However, the first line of the text file must have the form:

```
function [output variable list] = functionname(input variable list)
```

Furthermore, the name of the text file must be `functionname.m`. Here's an example, which is to be stored in a file called `addmult.m`

```
function [x,y] = addmult(a,b,c)
% A function to compute the sum and product of three numbers
x = a + b + c; % x is the sum
y = a * b * c; % y is the product
```

You can call the function from the MATLAB command line, from a script file, or from another function file, using the same format as the first line of the file. For example,

```
[mysum, myprod] = addmult(3,2,1);
```