ENGIN 100: Music Signal Processing

LAB #1: Matlab and Sampling

Professor Andrew E. Yagle

Dept. of EECS, The University of Michigan, Ann Arbor, MI 48109-2122

Revised July 2016

I. ABSTRACT

This introductory lab will give you some basic tools that will be used in the remaining labs. The goals of this lab are: (1) To gain the ability to apply the basics of Matlab *that will be used in Engin 100*; (2) To learn the basics of sinusoidal signals, which will be used extensively in Engin 100; and (3) To learn the basics of sampling continuous-time signals. Your projects in Engin 100 will be simple and small Matlab programs ("scripts") that operate on sampled continuous-time signals to determine their sinusoidal components.

II. Matlab

Bring earbuds or earphones or headphones to each lab.

A. Running Matlab

To run Matlab on a CAEN lab computer: When you log onto a CAEN lab computer (using your uniqname and Kerberos (UM) password), the website *http://software.engin.umich.edu* will open in Internet Explorer (IE). Check "no" when asked whether you want to make IE your default browser, then wait a few seconds to be "validated" as a user for CAEN software. Scroll down to "R2015a Matlab 32-bit Instructional" (64-bit is also OK) and click it. Then look for the "jukebox" which opens in a separate window, and click on "Matlab." It will take a few seconds to load. Look for a window with a prompt >>. Your Matlab directory is listed to the left of this window. When downloading files from Canvas, put them in this directory. I will use `this font` for Matlab commands, to be typed after the prompt.

B. About Matlab

Matlab is a computer program developed and sold by the Mathworks, Inc. It is the most commonly used mathematics computer program in signal processing, and it is used extensively in all fields of engineering. You may want to U-M presently has a program where you can obtain the student version of Matlab free.

"Matlab" is an abbreviation of MATrix LABoratory. It is based on a set of numerical linear algebra programs, written in FORTRAN, called LINPACK. I used LINPACK (using punch cards!) when I was a graduate student at M.I.T. So Matlab formulates problems in terms of vectors and arrays of numbers, and

often solves problems by formulating them as linear algebra problems. For example, Matlab computes the roots of a polynomial by computing the eigenvalues of the companion matrix formed from the polynomial coefficients. (If you didn't understand that last sentence, don't worry about it.) When using Matlab, you must formulate your problem in terms of arrays of numbers. "To a hammer, all problems look like nails."

A Matlab program ('script') is a list of Matlab commands executed in succession. This lab will teach you enough about Matlab to be able to use it. You will learn more in later Engin 100 labs, and in Engin 101.

C. Using Matlab

D. Vectors and Arrays

Since Matlab is based on linear algebra, in using Matlab you must think in terms of vectors and arrays. Examples of a *row vector*, a *column vector*, an *array*, and the array's *transpose* are, respectively,

$$X = \begin{bmatrix} 3 & 1 & 4 \end{bmatrix} \qquad Y = \begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix} \qquad Z = \begin{bmatrix} 3 & 1 & 4 \\ 1 & 5 & 9 \\ 2 & 6 & 5 \end{bmatrix} \qquad Z' = \begin{bmatrix} 3 & 1 & 2 \\ 1 & 5 & 6 \\ 4 & 9 & 5 \end{bmatrix}$$

In Matlab you type these in as: `X=[3 1 4] Y=[3;1;4] Z=[3 1 4;1 5 9;2 6 5] Z'`

You could also type `Y=X'` where the *transpose* X' means "convert rows to columns, and vice-versa."

A common problem in Matlab is thinking you have a row vector when in fact you have a column vector. You can check by using `size(X)`; here that gives `ans=1,3` which tells you that `X` is a 1×3 (row) vector.

Try the following commands and note what they are doing (`reshape` will be *very* useful later):

- `A=[1 2 3;4 5 6], A(:)` (stacks A by columns), then `reshape(A(:),3,2)` (unstacks to 3×2 array).
- `X=[3 1 4 1 5 9 2 6 5 3 5]; for I=1:10;B(I)=X(I+1)-X(I);end; B` This takes differences of X.
- `X=[3 1 4 1 5 9 2 6 5 3 5]; C=X(2:11)-X(1:10)` This is a faster way to take differences of X.
- `D=[1 2 3]; E=[4 5 6]; F=[D E]` This *concatenates* D and E (i.e., it appends E after D to get F).

E. Plotting

By far the most common Matlab task you will be performing is plotting functions.

Try running the following examples (no, this is not part of the lab yet):

- `T=[1:100];X=cos(2*pi*T/30);plot(X)` This plots a sinusoidal signal with period=30.
- `T=[1:100];Y=2.^(T/20);plot(Y)` This plots an exponential function growing in time.
- `T=[1:100];Z=2.^(-T/20).*cos(2*pi*T/30);plot(Z)` This plots a decaying sinusoid.

In each case typing `T` and `X` will give you a row vector of 100 numbers. You can get column vectors by typing `T'` and `X'` and put them side-by-side by typing `[T' X']` (think in terms of vectors or arrays).

- You can plot two functions at once (note the colors): `plot(T,X,T,Z)` You can plot X vs. Y: `plot(Y,X)`
- You can add a title to a plot using >>`title('This is the title of my plot')`

In both cases the number of elements in `T` and `X`, or `X` and `Y`, must be the same.

F. Miscellaneous

These simple examples illustrate several important points about Matlab:

- Defining X as a function of T produces a vector of X values corresponding to the vector of T values;
- Putting a semicolon ";" at the end of a command suppresses output; without it Matlab will type the results of the computation on your screen. This is harmless, but irritating to have numbers flying by;
- "`.*`" multiplies two vectors point-by-point, while "`*`" multiplies vectors as in linear algebra (dot product). Try these: `[1 2]*[3 4]'` and `[1 2]'*[3 4]` and `[1 2].*[3 4]` and `[1 2]*[3 4]` (gives an error message);
- Compare `[0:.1:1]` (11 numbers spaced by 0.1) and `linspace(0,1,10)` (10 numbers spaced by 0.111).

G. Getting Help for Matlab

- You can get help for, say, `linspace` by typing `help linspace`. Sometimes way too much information!
- See www.eecs.umich.edu/~aey/eecs451/whywont.html for ideas on what's wrong with your program.
- See www.eecs.umich.edu/~aey/eecs451/matlab.pdf for a quick tutorial on Matlab (for another course).

H. Programs (scripts or .m files)

You can access previously-typed commands using uparrow and downarrow on your keyboard. But it gets old. You can write a *script* (a sequence of Matlab commands) using *.m files*. At the upper left, click:

File→New→m-file (or click **new script**).

This opens a window with a text editor. Type in commands and then:

File→Save as→yagle.m (or whatever you want to name your file).

Make sure you save it with an .m extension. Then you can run the file by typing its name at the prompt: >> `yagle`. Make sure the file name is not the same as a Matlab command! Using your name is a good idea.

To *download a file* from a web site, right-click on it, select **save target as**, and use the menu to select the proper file type (specified by its file extension). m-files should have a .m extension (such as yagle.m).

I. Listening to sound files

>>`[X Fs]=audioread('yagle.wav')` converts the sound file *yagle.wav* into Matlab vector `X` with sampling rate `Fs` $\frac{\text{samples}}{\text{second}}$. Older versions of Matlab used `[X Fs]=wavread('yagle.wav')` to do this.

Write Matlab vector `X` and sampling rate `Fs` to .wav file *yagle.wav* using >>`audiowrite('yagle.wav',X,Fs);`

Listen to `X` using >>`soundsc(X,Fs)`. *soundsc* scales `X` so that it varies between –1 and +1 before playing it, so that the sound isn't clipped. The Matlab default for sampling rate `Fs` is (a low) 8192 $\frac{\text{samples}}{\text{second}}$.

J. Printing

You can print out the current figure (the one in the foreground; click on a figure to bring it to the foreground) by typing >> `print` at the prompt. You can print the figure to a encapsulated postscript file (.eps) by typing >> `print -depsc2 yagle.eps`, which will appear in the Matlab directory.

An .eps file can be inserted into a MS Word document using (in MS Word) **insert→picture→from file**, changing to the proper directory, and selecting "encapsulated postscript" from the **file type** menu, and then selecting the desired .eps file. You will be doing this frequently in your lab reports and presentations.

Printing one plot on a page is often wasteful of paper. You can put several plots on a page using `subplot`. `subplot(mni)` produces an m×n array of plots; i is a number between 1 and mn that specifies where in the array the next plot is to be placed. For example, for 6 plots on one page:

```
subplot(311),plot(T1,X1),subplot(312),plot(T2,X2),subplot(313),plot(T3,X3)
subplot(324),plot(T4,X4),subplot(325),plot(T5,X5),subplot(326),plot(T6,X6)
```

Store all of the variables in a Matlab session in file `yagle.mat` using >>`save yagle.mat`.

Reload all of the variables in file `yagle.mat` using >>`load yagle.mat`.

To exit Matlab (quit? when you're having so much fun?), type >> `quit`. You've returned to reality!

III. Signals, Sinusoids, and Sampling

A. Sinusoids

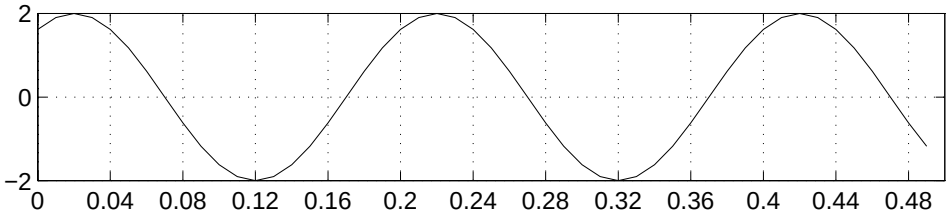
A *sinusoid* or *sinusoidal signal* is a function or signal of the form

$$x(t) = A\cos(2\pi f t + \theta), -\infty < t < \infty \tag{1}$$

For the sinusoid in equation (1), we define the following constants:

- A=amplitude. A has the same units or dimensions (e.g., "volts") as $x(t)$. $A \geq 0$ always.
- f=frequency in Hertz=$\frac{\text{CYCLES}}{\text{SECOND}}$. $x(t)$ has *period*=$\frac{1}{f}$ seconds, so $x(t)$=$x\left(t+\frac{1}{f}\right)$ for all t.
- t=time in seconds. $x(t)$ is a function of t; it associates a value of x with each value of t.
- θ=phase or phase shift in radians. To get degrees, multiply θ by $\frac{180}{\pi}$.
- Phase=θ is equivalent to time delay=$-\frac{\theta}{2\pi f}$ seconds.
- $\sin(x) = \cos(90^o - x) = \cos(x - 90^o)$, so a sine is a cosine with phase=-90^o. Its phase=-90^o.

Sinusoidal signals look like this:



- Amplitude=2=A (the *peak-to-peak* amplitude=4).
- Period=0.22–0.02=0.42–0.22=0.2=$\frac{1}{f}$ → f=5 Hertz.
- Time delay=0.02→ θ=phase=–$2\pi 5(0.02)$=$\pi/5$.
- So $x(t) = 2\cos(2\pi 5(t - 0.02)) = 2\cos(10\pi t - \pi/5)$.

Sinusoids will play a central role in Engin 100. As we will discover, they are the building blocks of music.

B. Sampling

Unless you have an analog computer, continuous-time signals such as music must be *sampled* or *discretized* into a string of numbers which can be processed on a computer. The numbers themselves must also be *quantized* or *digitized* into a finite number of bits (finite precision). We will not explicitly deal with quantization in Engin 100, although we will see some of its effects in the next lab. Note that "sampling" here does *not* mean incorporating a short segment of one recording into another, as is often done in music today.

The usual way to sample a continuous-time signal $x(t)$ is to set $t = n\Delta$ for integers n and a *sampling interval*=Δ seconds. This is *sampling at rate*=S=$\frac{1}{\Delta}\,\frac{\text{SAMPLES}}{\text{SECOND}} = \frac{1}{\Delta}$ "Hertz" ("Hertz" is a misnomer since it does not mean frequency here). Then the continuous time signal $x(t)$ is stored as the string of numbers

$$\{\ldots x(-2\Delta), x(-\Delta), x(0), x(\Delta), x(2\Delta)\ldots\} = \{\ldots x[-2], x[-1], x[0], x[1], x[2]\ldots\} \text{where}\, x[n] = x(n\Delta) = x(t)|_{t=n\Delta}.$$

If $x(t)$ has finite *duration* (length), i.e., it starts at some time and stops at another, as all music does (e.g., musical chairs), then $x[n]$ is a finite number of numbers. This makes $x[n]$ suitable for computer processing.

For a sinusoid sampled at S "Hertz"=S $\frac{\text{SAMPLE}}{\text{SECOND}}$ (i.e., every $\Delta = \frac{1}{S}$ seconds), the sampled signal is

$$x[n] = x(n\Delta) = A\cos(2\pi f n\Delta + \theta) = x\left(\frac{n}{S}\right) = A\cos\left(2\pi\frac{f}{S}n + \theta\right). \tag{2}$$

Note $\frac{f}{S}$, $f\Delta$ and n are all dimensionless. It is always a good idea to put equations in dimensionless form.

It might seem as though we lose information about $x(t)$ in representing it with its samples $x(n\Delta)$–how do we get $x(t)$ back from $x(n\Delta)$? Amazingly, we will see later in Engin 100 that we can reconstruct $x(t)$ *exactly* from its samples $x[n] = x(n\Delta) = x(\frac{n}{S})$ as long as $S > 2$(maximum frequency of $x(t)$). This was discovered by Claude Shannon, a UM alumnus. That's his bust outside the EECS building on the North Campus diag.

To link all of these concepts together, consider a standard music CD that holds 64 minutes of music. The musical signal is *sampled* at $44100\frac{\text{SAMPLE}}{\text{SECOND}}$, *quantized* using 16 bits to 2^{16}=65536 different values, and has two components since it is stereo (although early Beatles recordings were in mono). So the CD must store

$$\left[\frac{1\,\text{byte}}{8\,\text{bits}}\right]\left[16\frac{\text{bits}}{\text{channel}}\right]\left[2\frac{\text{channel}}{\text{sample}}\right]\left[44100\frac{\text{sample}}{\text{second}}\right]\left[60\frac{\text{second}}{\text{minute}}\right][64\,\text{minutes}] = 677\,\text{MBytes}.$$

IV. LAB #1: WHAT YOU HAVE TO DO

A. Sinusoids

Print all four of these plots on one page using `subplot` (saving paper and saving you money!):

- Type >> `T=[0:10];X=3*cos(T)+4*sin(T);plot(T,X)` This should be a jagged plot.

The reason is that it is only sampled at integers, and samples are connected by straight lines.

Type >> `T=[0:0.1:10];X=3*cos(T)+4*sin(T);plot(T,X)` This should be a smoother plot.

Are you surprised that the sum of a sin and a cos is a pure sinusoid? See Appendix for why.

• Type >> `T=[1:4000];X=cos(2*pi*440*T/8192);soundsc(X,8192)`. This should sound like note "A."

Type >> `plot(X)`. This should be a blue smear! It's about 200 cycles squished together.

Type >> `plot(X(1:100))` to "zoom in" on the first 100 samples of X to see the sinusoid.

• Type >> `soundsc(X(1:2:4000),8192)`. This omits every other value of X, halving its length.

The sinusoid is squished together and is speeded up, so the tone is now "A" one octave higher.

B. Sum of Two Sinusoids

Print all four of these plots on one page using `subplot` (saving paper and saving you money!):

• Type >> `T=[1:8192]/8192;Z=cos(2*pi*440*T)+cos(2*pi*444*T);soundsc(Z,8192)` Describe this sound.

Does it sound like two tones close together? Or like a single tone with a time-varying amplitude?

• Type >> `plot(Z)` and `plot(Z(1501:1700))` to zoom in on `Z`. This is a zoom of the first plot.

Now can you *see* why this sounds like one sinusoid with a varying amplitude? See Appendix for why.

• Type >> `T=[0:.01:.99];F=02;X=cos(2*pi*F*T);stem(X)`. Just a sampled 02 Hz sinusoid.

Type >> `T=[0:.01:.99];F=98;X=cos(2*pi*F*T);stem(X)`. Just a sampled 98 Hz sinusoid.

Sampling both 2 Hz and 98 Hz sinusoids at Δ=0.01→S=100 gives the same result! This is called *aliasing*.

C. Musical Signals

Matlab itself is boring. Now let's use it to listen to, display, and process some musical signals.

Print all four of these plots on one page using `subplot` (saving paper and saving you money!):

• Download the file `proj3.wav` from Canvas. You will use this file in Project #3.

It contains samples of an electric guitar, a clarinet, a trumpet, and a pure tone.

Each instrument is playing a complete musical scale (from note 'G' to note 'G').

• Type >>`[X fs]=audioread('proj3.wav');` This outputs the signal in `X` and sampling rate in `fs`.

What is the duration of all of this? Find out using `N=length(X);N/fs`,

since $\frac{\text{N SAMPLES}}{\text{fs } \frac{\text{SAMPLE}}{\text{SECOND}}} = \frac{N}{fs}$ seconds. Pay attention to units (dimensions)–they tell you what to do.

• To listen to the trumpet playing note G, we could use `soundsc(X(851969:884736),fs)`. Ugh!

• It's easier to use `reshape`: `XX=reshape(X',length(X)/4,4)` puts each instrument in a column.

Then `soundsc(XX(1:32768,3),fs)` does the same as `soundsc(X(851969:884736),fs)`. Try it.

• Plot each instrument playing note 'G': `for I=1:4;subplot(2,2,I),plot(XX(1:300,I));end`.

In the `title` for each plot: Include the identity of each instrument.

• Measure the approximate period of the sinusoid `XX(1:32768,4)` , and compute its frequency.

This is not a good way to compute frequency! We'll use a simple DSP algorithm to do it in Lab #2.

• We can raise the frequency of the note being played by an octave by skipping every other sample.

Compare `soundsc(XX(1:2:32768,3),fs)` to `L=size(XX,1);soundsc(XX(L-32767:L,3),fs)`.

The latter is the highest trumpet note in `X`, one octave higher than the lowest trumpet note in `X`.

D. Dereverbing

Now let's do some simple digital Signal Processing (DSP). Download the file `reverb.mat` from Canvas. Then >>`load reverb;soundsc(Y,24000)`. Describe this sound, which was produced using the formula $y[n] = x[n] + (0.8)x[n - 3(1024)] + (0.8)^2x[n - 6(1024)] + (0.8)^3x[n - 9(1024)] + \ldots$ for some signal $x[n]$.

Using this, write an expression for $0.8y[n - 3(1024)]$, and show that $x[n] = y[n] - 0.8y[n - 3(1024)]$.

Dereverb the signal $y[n]$ (in Matlab vector `Y`) to get the original signal $x[n]$ (in Matlab vector `X`) using the Matlab command

>>`X=Y-0.8*[zeros(1,3072) Y(1:length(Y)-3072)];soundsc(X,24000)`. Describe this sound.

E. Lab Report

Submit your report on Canvas as a Word (.doc) file.

F. Next Week

You will learn how to measure the frequency of a sinusoid using a computer, and use this and some elementary data visualization to learn that musical tones in "The Victors" occur in a 12-tone scale.

V. APPENDIX

- Why was the sum of a sine and a cosine a pure sinusoid, not something messier?
- Why was the sum of two sinusoids at close frequencies one sinusoid with varying amplitude?

Both results follow from the *cosine addition formula* $\cos(x - y) = \cos(x)\cos(y) + \sin(x)\sin(y)$:

First, set $x = 2\pi ft$ and $y = \theta$ and multiply by C to get

$C\cos(2\pi ft - \theta) = C\cos(\theta)\cos(2\pi ft) + C\sin(\theta)\sin(2\pi ft) = A\cos(2\pi ft) + B\sin(2\pi ft)$.

Setting $t = 0$ and $t = \frac{1}{4f}$ gives the formulae (the last two follow from the first two):

$A = C\cos(\theta)$; $\quad B = C\sin(\theta)$; $\quad \tan\theta = \frac{B}{A}$; $\quad C = \sqrt{A^2 + B^2}$

We will use this equation (think about polar coordinates) extensively in Lab #3.

Second, set $x = 2\pi 442t$ and $y = \pm 2\pi 2t$ and add and subtract the results to get:

$\cos(2\pi 440t) + \cos(2\pi 444t) = 2\cos(2\pi 2t)\cos(2\pi 442t)$.

So 440 Hz and 444 Hz sinusoids together are identical to a 442 Hz sinusoid with varying amplitude.

We will use this result (think tuning a piano) in Lab #2 to measure frequency.

A. Don't Always Believe Computers

- Is Matlab infallible? Type >> `roots(poly(ones(20,1)))`. This computes the polynomial $(z-1)^{20}$ having 20 multiple roots at $z = 1$, and then computes the roots of that polynomial. Did you get 20 ones? No!
- To see what happened, type >> `zplane(roots(poly(ones(20,1))))` (this won't work in Central Campus computer labs, since they don't have Matlab's Signal Processing Toolbox installed). This plots the 20 roots. Are computers infallible? Matlab has real problems with multiple roots. (Don't print this plot.)