GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

**EE 2200 Winter 1999**
**Lab #3: Harmonic Signals & Musical Notes**

Date: week of 25 Jan 1999

---

This is *the official* Lab #3 description; it consists of some material from the two labs in Appendix C.2 and C.3 of the text.

The Warm-up section of each lab must be completed in Lab and the steps marked *Instructor Verification* must also be signed off **during your scheduled lab time**. One of the laboratory instructors must verify the appropriate steps by initialing on the **Instructor Verification** line. When you have completed a step that requires verification, simply raise your hand and demonstrate the step to the instructor.

**Lab Report:** Please turn in Sections 3, 4 and 5 with explanations as this week's lab report. Staple the **Instructor Verification** sheet to the end of your lab report as evidence that the appropriate steps were witnessed by the instructor.

The report will **due during the week of 1-Feb at the start of your lab**.

---

# 1 Introduction and Overview

The goal of this laboratory is to gain familiarity with sums of complex exponentials and their use in representing more complicated signals such as speech and music. The general form can be expressed in two ways:

$$x(t) = \Re e\left\{\sum_{k=1}^{N} Z_k e^{j2\pi f_k t}\right\} \quad (1)$$

$$x(t) = \sum_{k=1}^{N} A_k \cos(2\pi f_k t + \phi_k) \quad (2)$$

In this lab we will synthesize waveforms composed of sums of sinusoidal signals of this form, sample them, and then reconstruct them for listening. We will use the sum in equation (4) to synthesize the following signals:

1. Sine waves at a specific frequency played through a D/A converter.
2. Periodic signals that are sums of harmonically related sinusoids
3. Music signals that match the frequency of a specific note.

One additional objective of this lab is to establish the connection between musical notes, their frequencies and sinusoids.

## 1.1 Harmonic Sinusoids

There is an important special case where $x(t)$ is the sum of $N$ cosine waves whose frequencies $(f_k)$ are *different.* If we concentrate on the case where the $(f_k)$ are all multiples of one basic frequency $f_0$, i.e.,

$$f_k = k f_0 \qquad \text{(HARMONIC FREQUENCIES)}$$

then the sum of $N$ cosine waves given by (4) becomes

$$x_h(t) = \sum_{k=1}^{N} A_k \cos(2\pi k f_0 t + \phi_k) = \Re e \left\{ \sum_{k=1}^{N} Z_k \, e^{j2\pi k f_0 t} \right\} \tag{3}$$

This particular signal $x_h(t)$ has the property that it is also periodic with period $T_0 = 1/f_0$, because each of the cosines in the sum repeats with period $T_0$. The frequency $f_0$ is called the *fundamental frequency*, and $T_0$ is called the *fundamental period*. Notice that $T_0$ is also the shortest possible period.

# 2 Warm-up

The instructor verification sheet is included at the end of this lab.

In this lab, the sine waves and music signals will be created with the intention of playing them out through a speaker. Therefore, it is necessary to take into account the fact that a conversion is needed from the digital samples, which are numbers stored in the computer memory to the actual voltage waveform that will be amplified for the speakers.

## 2.1 D-to-A Conversion

The digital-to-analog conversion process has a number of aspects, but in its simplest form the only thing we need to worry about at this point is that the time spacing $(T_s)$ between the signal samples must correspond to the rate of the D-to-A hardware that is being used. From MATLAB, the sound output is done by the `soundsc(x,fs)` function[1] which does support a variable D-to-A sampling rate if the hardware on the machine has such capability. A convenient choice for the D-to-A conversion rate is 11025 samples per second,[2] so $T_s = 1/11025$ seconds; another common choice is 8000 Hz. Both of these rates satisfy the requirement of *sampling fast enough* as explained in the next section. In fact, most piano notes have relatively low frequencies, so an even lower sampling rate could be used.

## 2.2 Theory of Sampling

Chapter 4 treats sampling in detail, but this lab is usually done prior to lectures on sampling, so we provide a quick summary of essential facts here. The idealized process of sampling a signal and the subsequent reconstruction of the signal from its samples is depicted in Fig. 1. This figure shows a continuous-time in-

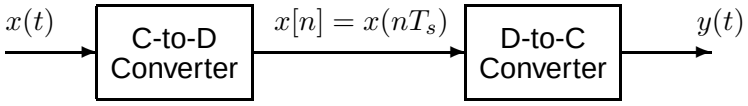


Figure 1: Sampling and reconstruction of a continuous-time signal.

put signal $x(t)$, which is sampled by the continuous-to-discrete (C-to-D) converter to produce a sequence of samples $x[n] = x(nT_s)$, where $n$ is the integer sample index and $T_s$ is the sampling period. The sampling rate is $f_s = 1/T_s$ where the units are samples per second. As described in Chapter 4, the ideal discrete-to-continuous (D-to-C) converter takes the input samples and interpolates a smooth curve between them. The *Sampling Theorem* tells us that if the input signal $x(t)$ is a sum of sine waves, then the output $y(t)$ will be

[1] In MATLAB version 5, the function `soundsc(x,fs)` performs automatic scaling to avoid saturating the D-to-A converter which would give a distorted sound. If the `sound()` function were used instead of `soundsc()`, it would be necessary to scale the vector `x` so that it lies between $\pm 1$.

[2] This sampling rate is one fourth of the rate (44,100 Hz) used in audio CD players.

equal to the input $x(t)$ if the sampling rate is more than twice the highest frequency $f_{\max}$ in the input, i.e., $f_s > 2f_{\max}$. In other words, if we *sample fast enough* then there will be no problems synthesizing the continuous audio signals from $x[n]$.

Most computers have a built-in analog-to-digital (A-to-D) converter and a digital-to-analog (D-to-A) converter (usually on the sound card). These hardware systems are physical realizations of the idealized concepts of C-to-D and D-to-C converters respectively, and for purposes of this lab we will assume that they are perfect realizations.

(a) The ideal C-to-D converter is, in effect, being implemented whenever we take samples of a continuous-time formula, e.g., $x(t)$ at $t = t_n$. We do this in MATLAB by first making a vector of times, and then evaluating the formula for the continuous-time signal and at the sample times, i.e., $x[n] = x(nT_s)$ if $t_n = nT_s$. This assumes perfect knowledge of the input signal, but we have already been doing it this way in Lab #2.

To begin, compute a vector `x1` of samples of a sinusoidal signal with $A_1 = 100$, $\omega_1 = 2\pi(660)$, and $\phi_1 = \pi/4$. Use a sampling rate of 11025 samples/second, and compute a total number of samples equivalent to 2 seconds time duration. You may find it helpful to recall that the MATLAB statement `tt=(0:0.01:3);` would create a vector of numbers from 0 through 3 with increments of 0.01. Therefore, it is only necessary to determine the time increment needed to obtain 11025 samples in one second. You should use the `makecos()` from the previous lab for this part.

Use `soundsc()` to play the resulting vector through the D-to-A converter of the your computer, assuming that the hardware can support the $f_s = 11025$ Hz rate. Listen to the output.

(b) Now compute another vector `x2` of samples of the sinusoidal signal (0.8 secs. in duration) for the case $A_2 = 150$, $\omega_2 = 2\pi(880)$, and $\phi_2 = -\pi/2$. Listen to the signal reconstructed from these samples. How does it compare to the signal in part (a)?

(c) **Insert** the signal `x2` in the middle of `x1` by using the colon notation. Do the insertion so that `x2` starts at time $t = 0.75$ secs. You should be able to use a statement something like: (assuming that both `x1` and `x2` are row vectors):

```
xx = x1;   xx(n1:n2) = x2;
```

Determine the values of `n1` and `n2` so that the second signal is in the correct place. Listen to this signal. Explain what you heard.

(d) To verify that the insert operation was done correctly in the previous part, make the following plot:

```
tt = 0:(1/11025):2;   plot( tt, xx );
```

This will show the "envelope" of the signal and verify that the amplitude changes from 100 to 150 at $t = 0.75$ secs. Make sure that the `tt` vector is the same as you used in part (a).

(e) Now send the vector `xx` to the D-to-A converter again, but double the sampling rate in `soundsc( )` to 22050 samples/second. *Do not recompute the samples in* `xx`, just tell the D-to-A converter that the sampling rate is 22050 samples/second. Describe how the *duration* and *pitch* of the signal were affected. Explain.

**Instructor Verification** (separate page)

## 2.3 Vectorizing in MATLAB

In MATLAB loops are usually very inefficient, so the `for` loop should be avoided when writing functions. This is especially true when the function is going to process a very long signal, e.g., an audio signal lasting many seconds or several minutes.

Learning to write "vectorized" code is easy *if you understand matrix and vector notation.* For example, a sum of products can be rewritten as an *inner product* of vectors:

$$c = \sum_{k=1}^{N} z_k^* y_k \qquad \Longrightarrow \qquad c = \mathbf{z}^* \mathbf{y}^{\mathbf{T}}$$

if we define the row vectors **z** and **y** in the following manner:

$$\mathbf{z} = \begin{pmatrix} z_1 & z_2 & \cdots & z_N \end{pmatrix} \qquad \text{and} \qquad \mathbf{y} = \begin{pmatrix} y_1 & y_2 & \cdots & y_N \end{pmatrix}$$

The superscript $^*$ as in $\mathbf{z}^*$ denotes the conjugate operator, and the superscript $^T$ as in $\mathbf{y}^T$ denotes transpose. Be careful in MATLAB because the prime operator does both the transpose and the conjugate together.

How does this impact coding in MATLAB? The answer is that most summations should not be done as `for` loops in MATLAB, but rather should be implemented as some sort of matrix (or vector) multiplication. It turns out that this strategy is not limited to arithmetic operations. It is also possible to vectorize logical operations as discussed in Appendix B, pp. 412–413.

## 2.4 Vectorizing Arithmetic Operations

Vectorize the following function with a loop; in other words, replace the loop with *one* MATLAB statement that does the same thing. Recall mag-squared identity for complex numbers: $|z|^2 = z^* z$.

```
function ss = sumsqd(z)
%SUMSQD  do the sum of the magnitude squared of all the elements in z.
%  (The input vector contains complex elements)
%
ss = 0;
for k=1:length(z)
   magsqd = real(z(k)).^2 + imag(z(k)).^2;
   ss = ss + magsqd;
end
```

Demonstrate that your vectorized function works by executing: `sumsqd( exp(j*rand(1,7)) )`.

**Instructor Verification** (separate page)

## 2.5 Vectorizing a Copy Operation

(a) The *outer product* is a special matrix product that multiples a column by a row. It can be used to do some clever operations. Execute the following lines of MATLAB code so that you can explain what operation they perform:

```
yy = ones(7,1) * rand(1,4);
%--The next 2 statements do the same operation using a different trick
%----that is much faster in execution, although harder to understand
xx = randn(1,4);
yy = xx(ones(7,1),:);
```

When unsure about a command, use `help`.

(b) Write a function that performs the same task as the following without using a `for` loop.

```
function Z = expand(x,ncol)
%EXPAND  Function to generate a matrix Z with identical columns equal
%        to an input vector x, i.e., make ncol copies of x.
%  usage:
%        Z = expand(x,ncol)
%        x = the input vector containing one column for Z
%     ncol = the number of columns needed in Z
%
x = x(:);  %--this turns the input vector x into a column vector
Z = zeros(length(x),ncol);
for i=1:ncol
   Z(:,i) = x;
end
```

## 2.6 Vectorizing Logical Operations

(a) Execute the following lines of MATLAB code so that you can explain what operation they perform:

```
A = randn(6,3);
A = A .* (A>0);
A = A + 2*(A==0);
```

(b) Write a new function that performs the same task as the following function without using the `for` loops. Use the idea in part (a) and also consult Section B.7.3 on vector logicals in Appendix B: *Using* MATLAB. In addition, the MATLAB logical operators are summarized via `help relop`.

```
function  Z = replacez(A)
%REPLACEZ  replace negative elements of a matrix with the number 77
%   usage:
%      Z = replacez(A)
%      A = input matrix whose negative elements are to be replaced
%
[M,N] = size(A);
for i=1:M
   for j=1:N
      if A(i,j) < 0
         Z(i,j) = 77;
      else
         Z(i,j) = A(i,j);
      end
   end
end
```

# 3 Exercises: Complex Exponentials

## 3.1 Sinusoidal Synthesis with an M-file

Since we will generate many functions that are a "sum of sinusoids," it will be convenient to have a function for this operation. To be general, we will allow the frequency of each component $(f_k)$ to be different. The following expressions are equivalent if we define the complex amplitude $Z_k$ as $Z_k = A_k e^{j\phi_k}$.

$$x(t) = \Re e\left\{\sum_{k=1}^{N} Z_k e^{j2\pi f_k t}\right\} \quad (4)$$

$$x(t) = \sum_{k=1}^{N} A_k \cos(2\pi f_k t + \phi_k) \quad (5)$$

## 3.2 Write a Vectorized Sum of Cosines

Write an M-file called `makevcos` that will synthesize a waveform in the form of (4). In the previous lab, this function was written with `for` loops, but these are rather inefficient in MATLAB. *In this lab, you must rewrite the function with NO loops.* The first few statements of the M-file are the comment lines—they should look like:

```
function        [xx,tt] = makevcos(ff, ZZ, fs, dur)
%MAKEVCOS   VECTORIZED Function to synthesize a sum of cosine waves
%  usage:
%    [xx,tt] = makevcos(ff, Z, fs, dur)
%      ff = vector of frequencies
%              (these could be negative or positive)
%      ZZ = vector of complex amplitudes: Amp*e^(j*phase)
%      fs = the number of samples per second for the time axis
%     dur = total time duration of signal
%      xx = vector of sinusoidal values
%      tt = vector of times, for the time axis
%
%    Note: ff and ZZ must be the same length.
%             ZZ(1) corresponds to frequency ff(1),
%             ZZ(2) corresponds to frequency ff(2), etc.
```

The MATLAB syntax `length(ff)` returns the number of elements in the vector `ff`, so we do not need a separate input argument for the number of frequencies. On the other hand, the programmer (that's you) should provide error checking to make sure that the lengths of `ff` and `ZZ` are the same. See `help error`. *Include a copy of the* MATLAB *code with your lab report.*

## 3.3 Hints for Vectorizing the Sum of Cosines

Here are some ideas that would help in *vectorizing* the sum of cosines function, `makecos()`.

(a) Take advantage of the outer product property to generate a matrix that is a function of $t$ and $f_k$. The dimensions of the matrix should be $L \times N$ where $L$ is the number of time samples and $N$ is the number of frequencies.

(b) Take advantage of the fact that matrix-vector multiplication computes a sum of products. For example,

$$\mathbf{c} = \mathbf{A}\mathbf{b} \qquad \Longrightarrow \qquad c_n = \sum_{k=1}^{N} a_{nk} b_k \tag{6}$$

where $c_n$ represents the $n^{\text{th}}$ element of the vector $\mathbf{c}$, $a_{nk}$ is the element in the $n^{\text{th}}$ row and $k^{\text{th}}$ column of the matrix $\mathbf{A}$, $b_k$ is the $k^{\text{th}}$ element of the vector $\mathbf{b}$, and $N$ is the number of columns in $\mathbf{A}$.

It is possible (although not required) to complete this function in a single line. For some other hints, please refer to the review of matrix multiplication in the *Using* MATLAB Appendix B of the text.

### 3.4 Testing

In order to guarantee that your `makevcos` function is correct, repeat the tests that you did in Lab #2. Those tests are given below. *Make a subplot with all the working test cases for your lab report.*

```
[xx1,tt1] = makevcos([10],   [j],    150, 0.3);
[xx2,tt2] = makevcos([5],   [1,j],   150, 0.3);
[xx3,tt3] = makevcos([10,10,10], [j,-1,j], 150, 0.3);
[xx4,tt4] = makevcos([4,6,8], [j,1,0.3*j], 20, 1);   %--Notice the Period
```

## 4 Piano Keyboard

In the next lab, you will have to build a large program that will synthesize the notes of a well known piece of music.[3] Since these signals require sinusoidal tones to represent piano notes, a quick introduction to the frequency layout of the piano keyboard is needed. The result will be that we can develop a formula that gives the frequency for each key. On a piano, the keyboard is divided into octaves—the notes in each octave being twice the frequency of the notes in the next lower octave. For example, the reference note is the A above

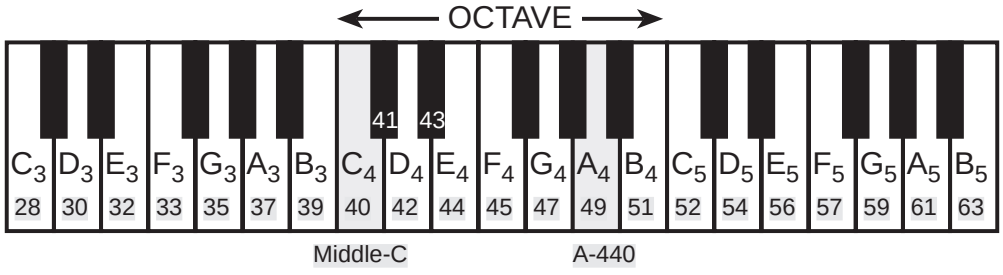


Figure 2: Layout of a piano keyboard. Key numbers are shaded. The notation $C_4$ means the C-key in the fourth octave.

middle-C which is usually called A-440 (or $A_4$) because its frequency is 440 Hz. One octave below is $A_3$ which has a frequency of 220 Hz. Each octave contains 12 notes (5 black keys and 7 white) and the ratio between the frequencies of the notes is constant between successive notes. Thus this ratio must be $2^{1/12}$.

[3] If you have little or no experience reading music, don't be intimidated. Only a little knowledge is needed to carry out this lab. On the other hand, the experience of working in an application area where you must quickly acquire knowledge is a valuable one. Many real-world engineering problems have this flavor, especially in signal processing which has been applied in diverse areas such as geophysics, medicine, radar, speech, etc.

Since middle C is 9 keys below A-440, its frequency is approximately 261 Hz. Consult chapter 9 for even more details.

Another interesting relationship is the ratio of fifths and fourths as used in a chord. Strictly speaking the fifth note should be 1.5 times the frequency of the base note. For middle-C the fifth is G, but the frequency of G is about 392 Hz which is not exactly 1.5 times 261.6. It is very close, but the slight detuning introduced by the ratio $2^{1/12}$ gives a better sound to the piano overall. This innovation in tuning is called "equally-tempered" and was introduced in Germany in the 1760's and made famous by J. S. Bach in the "Well Tempered Clavichord."

You can use the ratio $2^{1/12}$ to calculate the frequency of notes anywhere on the piano keyboard. For example, the E-flat above middle-C (black key number 43) is 6 keys below A-440, so its frequency should be $f = 440 \times 2^{-6/12} = 440/\sqrt{2} \approx 311$ Hertz.

(a) Generate a sinusoid of 2 seconds duration to represent the note E above A-440 (key number 56). Choose the appropriate values for $T_s$ and `fs`. Remember that `fs` should be at least twice as high as the frequency of the sinusoid you are generating. Also, $T_s$ and `fs` must "match" for the note played out of the D-to-A converter to sound correct.

(b) Now write an M-file to produce a desired note for a given duration. Your M-file should be in the form of a function called `note.m`. You may want to call the `makevcos` function that you wrote previously. Your function should have the following form:

```
function tone = note(keynum,dur)
% NOTE  Produce a sinusoidal waveform corresponding to a
%       given piano key number
%
%  usage:  tone = note (keynum, dur)
%
%        tone = the output sinusoidal waveform
%      keynum = the piano keyboard number of the desired note
%         dur = the duration (in seconds) of the output note
%
fs = 11025;     %-- or use 11025 Hz
tt = 0:(1/fs):dur;
freq =
tone =
```

For the `freq =` line use the formulas given in the previous section to determine the frequency for a sinusoid in terms of its key number. You should start from a reference note (middle-C or A-440 is recommended) and solve for the frequency based on this reference. For the `tone =` line generate the actual sinusoid at the proper frequency.

### 4.1 Testing `makevcos()` with Sounds

**Summation:** Use your `makevcos()` function to generate the **sum** of four sinusoids with frequencies that correspond to notes in the A-minor piano chord which consists of the keys $\{A_3,\ C_4,\ E_4,\ A_4\}$. If these four notes are played at the same time, the result should be an A-minor chord. Make all the amplitudes equal, pick the phases to be random, and make the length of the signal 1.2 seconds. Use a sampling rate of 11025 Hz. Play the signal with `soundsc()` to verify that it makes a pleasant sounding chord.

**Concatenation:** Utilize your `note()` function to generate a **sequence** of notes for $\{F_4,\ A_4^\flat,\ A_4,\ A_4^\flat\}$ These notes should be played individually, one after another. Make the duration of each note equal to 0.2 seconds, and also put a very short pause (`zeros`) of 0.05 secs. in between each note. Note: in MATLAB row vectors can be concatenated by writing `xxx = [xx1,zz,xx2,zz,xx3,zz,xx4]`.
*For your lab report, include the* MATLAB *code used to generate the sounds in both cases.*

## 5 Periodic Waveforms

Each of the following waveforms can be synthesized with a simple call to the function `makevcos`. Plot a short section of the signal to observe its characteristic shape.

Note: It is important to have a sampling rate that is at least *twice as high as the highest frequency component in your signal*, a fact that will be explained when sampling is discussed in Chapter 4. However, for this lab you should just choose a very large number for $f_s$ to get a smooth plot. Let $f_s$ be at least ten times the highest frequency.

(a) Try your `makevcos` M-file with the fundamental $f_0 = 7$ Hz, $f_k = kf_0$, and

$$Z_k = \begin{cases} \dfrac{j}{3k} & k = 1, 3, 5, 7, \ldots \quad \text{(i.e., an odd integer)} \\ 0 & k = 0, 2, 4, 6, 8, \ldots \end{cases} \tag{7}$$

Specify the duration to get three periods of the waveform.

Make plots for three different cases: $N = 3$, 7, and 15 (where $N$ is the largest subscript for $Z_k$). Use a three-panel subplot to show all three signals together. Explain how the period of synthesized waveform is related to the fundamental frequency.

Explain what happens as $N \to \infty$. What wave shape do the plots converge to? Although the waveshape is converging to a simple form, it is not perfect. Describe any unusual features in the converging waveform as $N \to \infty$.

(b) It is informative to listen to these signals as a different number of coefficients are added. Repeat the synthesis from part (a) with $f_0 = 500$ Hz and listen to the cases where $N = 1, 3$, and 7. You will need about 1 second of the signal to hear differences. Describe how the sound changes as the number of coefficients $N$ increases.

Note: when using `soundsc(x, fs)`, you must specify the sampling frequency. It must be greater than twice the highest frequency to avoid aliasing effects (as will be discussed in Chapter 4).

(c) Now try the coefficients

$$Z_k = \begin{cases} \dfrac{j^k}{k^2} & k = 1, 3, 5, 7, \ldots \\ 20 & k = 0 \\ 0 & k = 2, 4, 6, 8, \ldots \end{cases} \tag{8}$$

Choose the fundamental frequency to be $f_0 = 7$ Hz. Make plots for three different cases: $N = 1$, 3, and 7 (where $N$ is the largest subscript for $Z_k$), and plot all three functions together with a three-panel subplot. What waveshape is approximated with this sum of cosines as $N \to \infty$? Explain how the period of synthesized waveshape is related to the fundamental frequency.

# Lab #3
# EE-2200
# Winter-1999
# INSTRUCTOR VERIFICATION SHEET

Staple this page to the end of your Lab Report.

Name: _____ Date of Lab: _____

Part 2.2(e) Synthesizing two sinusoids, combining them and playing them at two different D-to-A rates:

Verified:_____ Date/Time:_____

Part 2.4
Vectorize the function `sumsqd()` and demonstrate that it works by executing `sumsqd( exp(j*rand(1,7)) )`:

Verified:_____ Date/Time:_____