GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Fall 2000
Lab #3: Harmonic Signals & Musical Notes

Date: 12–18 Sept 2000

This is *the official* Lab #3 description; some material comes 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**. When you have completed a step that requires verification, simply raise your hand and demonstrate the step to the instructor. Turn in the **Instructor Verification** sheet at the end of your scheduled lab time.

Lab Report: Please turn in Sections 3, 4 and 5 with explanations as this week's lab report.

The report will **due during the week of 19-25 Sept. 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. One additional objective is to establish the connection between musical notes, piano keys, and their frequencies. The general form can be expressed in two ways:

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

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

where the complex amplitude X_k is $X_k = A_k e^{j\phi_k}$.

In this lab we will synthesize waveforms composed of sums of sinusoids, sample them, and then reconstruct them for listening. We will use the sum in equation (1) 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 specific notes and chords.

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 frequencies (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 (1) becomes

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

This particular signal $x_h(t)$ has the property that it is guaranteed to be periodic. Its period is $T_0 = 1/f_0$, because each of the cosines in the sum repeats every T_0 secs. 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 periodic waveforms 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(xx,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 samples/sec. 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 input

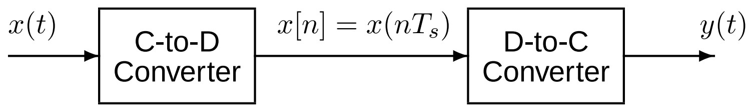

Figure 1: Sampling and reconstruction of a continuous-time signal.

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 of the text, 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 $y(t)$

[1]In MATLAB version 5, the function `soundsc(xx,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 `xx` so that it lies between ± 1. Consult `help sound.`

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

will be 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, but for purposes of this lab we will assume that the hardware A/D and D/A 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 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, create a vector `x1` of samples of a sinusoidal signal with $A_1 = 100$, $\omega_1 = 2\pi(800)$, and $\phi_1 = -\pi/3$. 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 `add_cos()` function 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 create another vector `x2` of samples of a second sinusoidal signal (0.8 secs. in duration) for the case $A_2 = 80$, $\omega_2 = 2\pi(1200)$, and $\phi_2 = +\pi/4$. Listen to the signal reconstructed from these samples. How does its sound 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:

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

assuming that both `x1` and `x2` are row vectors. This will replace part of `x1` with `x2`. Determine the values of `n1` and `n2` so that the second signal is in the correct place. Listen to this new 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 plot a huge number of points, but it will show the "envelope" of the signal and verify that the amplitude changes from 100 to 80 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 change the sampling rate parameter 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 as much as possible. 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:

$$\mathbf{z} = (\, z_1 \quad z_2 \quad \cdots \quad z_N \,) \qquad \text{and} \qquad \mathbf{y} = (\, y_1 \quad y_2 \quad \cdots \quad y_N \,)$$

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 (see `help transpose` and `help ctranspose`).

How does this impact the way we write code 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; 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 = conj(z(k)) * z(k);
   ss = ss + magsqd;
end
```

Demonstrate that your vectorized function works by executing: `sumsqd( exp(j*pi*(1:7)/7) )`. It is easy to figure out what the answer should be, by using properties of the complex exponential.

Instructor Verification (separate page)

2.5 Vectorizing a Copy Operation

(a) The *outer product* is a special matrix product that multiples a column vector by a row vector. 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(5,1) * [3,1,4,1,5,9]
size(yy)
```

```
%--The next 2 statements do the same operation using a different trick
%----that is much faster in execution, although harder to understand
xx = [3,1,4,1,5,9];
yy = xx([1,1,1,1,1],:)
```

(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
```

Instructor Verification (separate page)

2.6 Vectorizing Logical Operations

(a) Execute the following MATLAB code. Explain what operations are performed by the last 2 lines.

```
A = randn(6,3);
A = A .* (A>0);
A = A + pi*(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,afix)
%REPLACEZ  replace negative elements of a matrix with the number afix
%   usage:
%      Z = replacez(A)
%      A = input matrix whose negative elements are to be replaced
%   afix = replacement value for the negative elements
%
[M,N] = size(A);
for i=1:M
   for j=1:N
      if A(i,j) < 0
         Z(i,j) = afix;
      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. In the MATLAB function, we will use the form given in (4) with the complex amplitudes X_k defined as $X_k = A_k e^{j\phi_k}$.

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

Consult equation (2) for the equivalent sinusoidal form.

3.2 Write a Vectorized Sum of Cosines

Write an M-file called `vsynthesis` 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] = vsynthesis(fk, Xk, fs, dur, tstart)
%VSYNTHESIS  VECTORIZED Function to synthesize a sum of cosine waves
%  usage:
%    [xx,tt] = vsynthesis(fk, Xk, fs, dur, tstart)
%      fk = vector of frequencies
%              (these could be negative or positive)
%      Xk = vector of complex amplitudes: Amp*e^(j*phase)
%      fs = the number of samples per second for the time axis
%     dur = total time duration of the signal
%  tstart = starting time (default is zero, if you make this input optional)
%      xx = vector of sinusoidal values
%      tt = vector of times, for the time axis
%
%    Note: fk and Xk must be the same length.
%              Xk(1) corresponds to frequency fk(1),
%              Xk(2) corresponds to frequency fk(2), etc.
```

The MATLAB syntax `length(fk)` returns the number of elements in the vector `fk`, 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 `fk` and `Xk` are the same. See `help error`. *Include a copy of the* MATLAB *code with your lab report.*

3.3 Testing

In order to demonstrate that your `vsynthesis()` function is correct, synthesize the “vowel” signal shown in lecture and described in Chapter 3 of the book. These signals are shown in Figs. 3.9, 3.10 and 3.11 in the *DSP First* textbook. Make plots for two different cases: the sum of three components which is the signal $x_5(t)$ in Fig. 3.10 (top), and the sum of four components which is shown in Fig. 3.10 (bottom). Plot these two signals in the same window with a two-panel subplot. *Include these plots in your lab report.*
Note: there is an error in the text for Figs 3.9–3.11 because the MATLAB program had a minus sign in the exponent. However, your plots should look nearly the same except they should appear to be reversed.

4 Periodic Waveforms: Fourier Analysis and Synthesis

The Fourier Series method provides both *analysis* and *synthesis* capability.

1. If we start with a periodic signal $x(t) = x(T+T_0)$, then we can compute the Fourier Series coefficients by using the following analysis integral:

$$a_k = \frac{1}{T_0} \int_0^{T_0} x(t) e^{-j2\pi kt/T_0}\, dt$$

2. If we are given the Fourier Series coefficients, $\{a_k\}$, then we can synthesize a periodic signal using:

$$x_N(t) = \sum_{k=-N}^{N} a_k e^{j2\pi kt/T_0}$$

   where $2N+1$ is the number of terms used to form the signal and T_0 is the period.

3. In order to use the `vsynthesis()` function for Fourier synthesis, we must include all the $\{a_k\}$ coefficients for both the positive and negative k. Likewise the frequency vector, `fk` must contain both positive and negative harmonics.

In this exercise, you must synthesize a set of waveforms using different numbers of Fourier coefficients. The formula for the coefficients is:

$$a_k = \begin{cases} \dfrac{1}{j\pi k}\left(e^{jk\pi} - 1\right) & k \neq 0 \\ 10 & k = 0 \end{cases} \qquad (5)$$

(a) Write a short MATLAB function that will generate the Fourier coefficients $\{a_k\}$ over the range $-N \leq k \leq N$. The function should have one input N, and its output should be a vector containing the a_k coefficients.

(b) Use your `vsynthesis` M-file with a fundamental frequency of $f_0 = 3$ Hz, to generate synthesized signals, $x_N(t)$, with a finite number of coefficients. For the time interval of the synthesis, use $-0.3 \leq t \leq 0.7$ secs.

Make plots of $x_N(t)$ for three different cases: $N = 3$, 9, and 17 (where N is the largest subscript for a_k). Use a three-panel subplot to show all three signals together.

(c) Explain what happens as N gets larger and also draw a sketch of the waveshape obtained as $N \to \infty$. Relate this case to the known Fourier Series of a square wave. If necessary, run your program for $N = 99$ to get something close to the "converged" answer.

Note: You have to set up a time grid for the time interval used in the synthesis. The spacing between grid points must be small enough so that the sampling rate (grid points per second) is at least *twice as high as the highest frequency component in your signal.* This fact will be explained in great detail when sampling is discussed in Chapter 4. However, for this section you should just choose a very small spacing between grid points to get a smooth plot. You must determine the highest frequency harmonic to make an intelligent choice, e.g., five times the highest frequency (in Hz) is a conservative choice.

5 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. From this layout 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

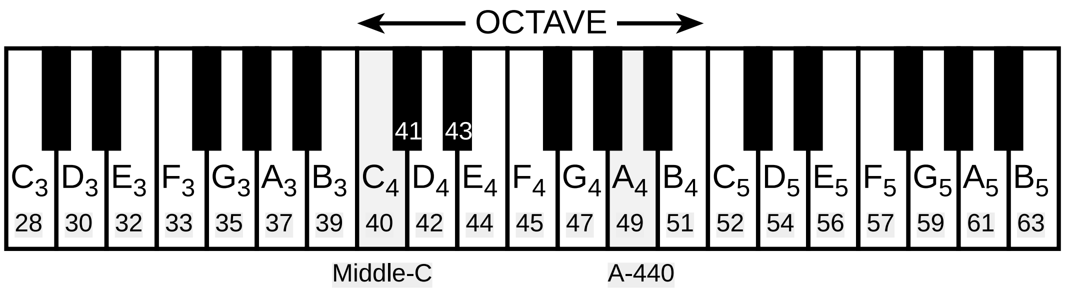

Figure 2: Layout of a piano keyboard. Key numbers are shaded. The notation C_4 means the C-key in the fourth octave. The key A_4 is A-440.

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; one octave above is A_5 at 880 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 $2^{1/12}$. 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 $f = 440 \times 2^{-6/12} = 440/\sqrt{2} \approx 311$ Hz. 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 `key2note.m`. You may want to call the `vsynthesis` function that you wrote previously. Your function should have the following form:

```
function xx = key2note(X, keynum, dur)
% KEY2NOTE  Produce a sinusoidal waveform corresponding to a
%       given piano key number
%
%  usage:  xx = key2note (A, phi, keynum, dur)
%
%        xx = the output sinusoidal waveform
%         X = complex amplitude for the sinusoid, X = A*exp(j*phi).
%    keynum = the piano keyboard number of the desired note
```

[3]If you have little or no experience reading music, don't be intimidated. Only a little knowledge is needed to carry out this upcoming lab. On the other hand, the experience of working in an application area where you must quickly acquire new domain 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, financial markets, etc.

```
%       dur = the duration (in seconds) of the output note
%
fs = 11025;     %-- or use 8000 Hz
tt = 0:(1/fs):dur;
freq =
xx = real( );
```

For the `freq =` line, use the formulas given above 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 `xx = real( )` line, generate the actual sinusoid as the real part of a complex exponential at the proper frequency. *For your lab report, include the* MATLAB *code for* `key2note()`.

5.1 Testing `key2note()` with Sounds

Concatenation: Utilize `key2note()` to generate a **sequence** of notes for $\{C_5,\, E_5,\, G_5,\, C_6,\, G_5,\, E_5,\, C_5\}$. These notes should be played individually, one after another. For the seven notes, choose the amplitudes so that the loudness decreases by 5% for each successive note; pick all the phases to be $-\frac{1}{2}\pi$. Make the duration of each note equal to 0.3 seconds, and also put a very short pause of 0.033 secs. in between each note. Use the MATLAB function `zeros(M,N)` to create a vector containing enough zeros to last 0.033 secs. 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 addition, draw a sketch of the* ideal *spectrogram showing the frequencies, timings and durations of the notes.* You can also use MATLAB to make a spectrogram of `xxx`, but that is not required for the lab report.

Lab #3
ECE-2025
Fall-2000
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. Explain the plot and how you placed the shorter sinusoid at the correct starting time.

Verified:____________________ Date/Time:__________

Part 2.4
Vectorize the function `sumsqd()` and demonstrate that it works by executing

```
sumsqd( exp(j*pi*(1:7)/7) ).
```

Tell the TA/instructor what you expect the answer to be by doing a mathematical derivation. Write your derivation in the space below.

Verified:____________________ Date/Time:__________

Part 2.5
Vectorize the copy operation function `expand()` and demonstrate that it works by executing

```
expand( exp(j*pi/2*(1:8)), 3 ).
```

Verified:____________________ Date/Time:__________