Massachusetts Institute of Technology
Dept. of Electrical Engineering and Computer Science
Spring Semester, 2007
6.082 Introduction to EECS 2

Lab #2: Time-Frequency Analysis

Goal:

Examination of signals in both the time and frequency domains, and observation of the benefits that each domain provides toward understanding the structure of a given signal. We will also briefly touch upon signal modeling concepts which are used to approximate certain signals through knowledge of their underlying structure. The context of our exercises will be examination of a recorded song from a piece of piano music, and an attempt to synthesize that music.

Instructions:

1. Complete the Prelab exercises ***BEFORE*** Wednesday's lab.
2. Complete the activities for Wednesday's lab (see below) and get checked off by one of the TAs before leaving. Be sure to work in pairs with one computer per pair.
3. Complete the Postlab exercises ***BEFORE*** Friday, and turn them in at the beginning of lecture on Friday.

Prelab:

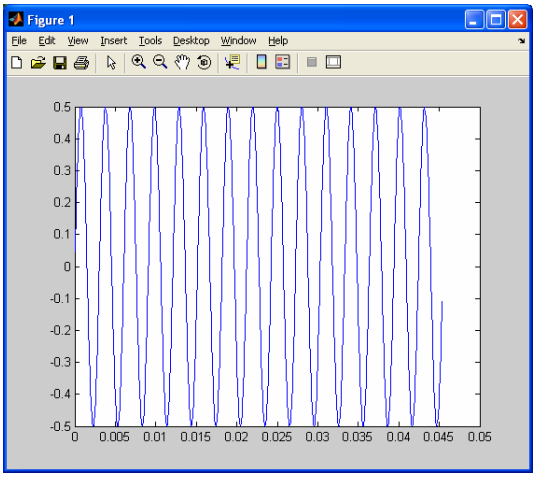
Understanding Sample Period, Exponential Damping, and the FFT function

Before we enter Lab 2 and examine signals created by a piano, we need to provide some additional background that will help us in our future analysis.

- To understand the concept of **sample period**, type the following commands in Matlab (note that you need not include the statements following the **%** symbols since they are simply comments):

```
Fs = 22050;                 % sample frequency in Hz
Ts = 1/Fs;                  % sample period in seconds
t = (1:1000)*Ts;            % time vector with 1001 samples
y = 0.5*sin(2*pi*330*t);    % sine wave with amplitude 0.5
                            %          and frequency 330 Hz
plot(t,y);
```

  - You should see a **Figure 1 plot** that appears as below. Note that the amplitude of the sine wave is easily seen to be 0.5, but that its frequency of 330 Hz requires more work to verify. To do so, note that the period of the sine wave should be 1/330 Hz = 0.003 seconds. To verify that the sine wave period is indeed that value, type **axis([0 0.003 -1 1])** in Matlab and observe the resulting plot.



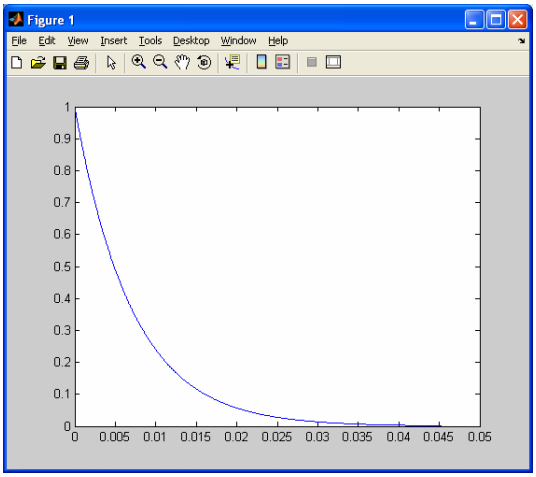
  - In the above example, we see that **sample period** corresponds to the increment between each time sample in the vector **t** that we created. To better understand this concept, consider the following two examples:
    - t = 1:1000 creates a vector [1 2 3 ….. 1000]
    - t = (1:1000)*Ts creates a vector [1*Ts 2*Ts 3*Ts …. 1000*Ts]

- We also see that **sample frequency** simply corresponds to the inverse of **sample period**. Note that the sine wave frequency and period are different than the sample frequency and period. In the case where the sample frequency is much *higher* than the sine wave frequency (or, equivalently, where the sample period is much *lower* than the sine wave period), the sampled waveform resembles a continuous-time waveform. We will provide more details of sampling later in the class.

- To understand the concept of **exponential damping**, type the following commands in Matlab:

```
tau = 7e-3;
r = exp(-t/tau);
plot(t,r);
```

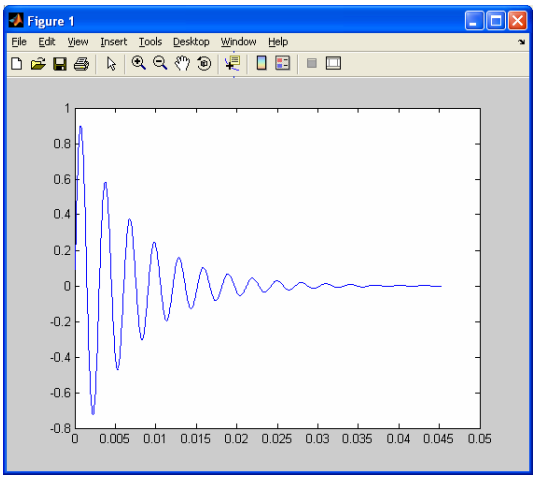
  - You should see a plot as shown below, which displays the decaying exponential function as a function of time. The rate of decay is set by the **time constant** of the exponential, which corresponds to variable **tau** in the above example. Try different values of **tau** in the above example to see how it influences the decay time of the exponential.



  - One important point – the decaying exponential that you are seeing above is *not* the same as the *complex* exponentials that we have been discussing in lecture.
  - While the above plot shows a decaying exponential, the expression **exponential damping** is typically associated with a *non-exponential* waveform whose *amplitude* decays in exponential fashion. A very relevant example is exponential damping of a sine wave, as illustrated by executing the following Matlab commands (with **tau** = 7e-3):

```
y_damped = sin(2*pi*330*t).*exp(-t/tau);
plot(t,y_damped);
```

- After execution of the above command, you should see the plot below. Be sure to note the use of the two characters ".*" in the above example – the ".*" character combination is used when multipling two vectors on an *element-by-element* basis. Try changing **tau** to a few different values to see the resulting waveform changes.



- To see the frequency domain content of a given signal in Matlab, we typically use the function **fft** (which stands for Fast Fourier Transform). Based on the Fourier analysis discussed in lecture, the **fft** operates on non-periodic signals which are sampled in time and have finite length. The output of the **fft** is very similar to what we would expect from doing the Fourier Series of a periodic signal – we essentially obtain **a** and **b** coefficients which can be examined to see the frequency content of a given signal.

  - In Matlab, type

  edit fft_example.m

  - Be sure to answer **yes** when prompted for creation of a new file.
  - Within the new edit window, type in the following commands (note that there is no need to type the comments following the **%** symbols):

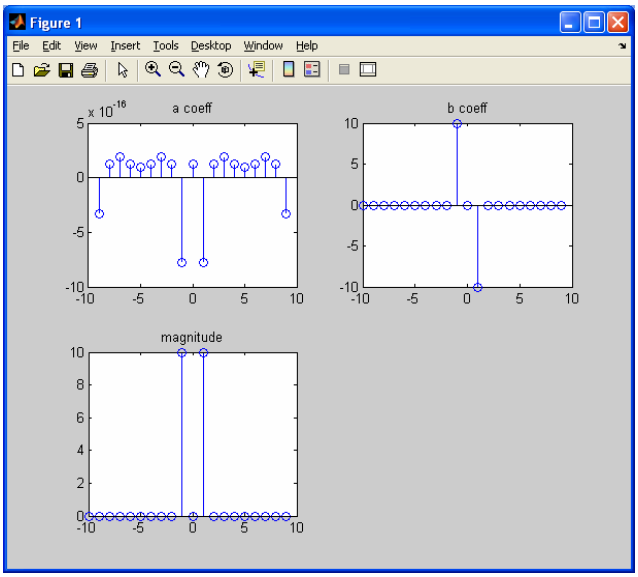
```
t = 0:19;
f = -10:9;   % use index of frequencies for easy viewing in stem plot
y = sin(2*pi*1/20*t);   % sine wave with frequency 1/20
yf = fft(y);
a = real(yf);     % a coefficients of transform
b = imag(yf);   % b coefficients of transform

subplot(221);   % left, upper corner of 2x2 plot
stem(f,fftshift(a));  % make stick figure plot, fftshift used to
                      % properly show positive and negative frequencies
title('a coeff');
```

subplot(222); % right, upper corner of 2x2 plot
stem(f,fftshift(b));
title('b coeff');

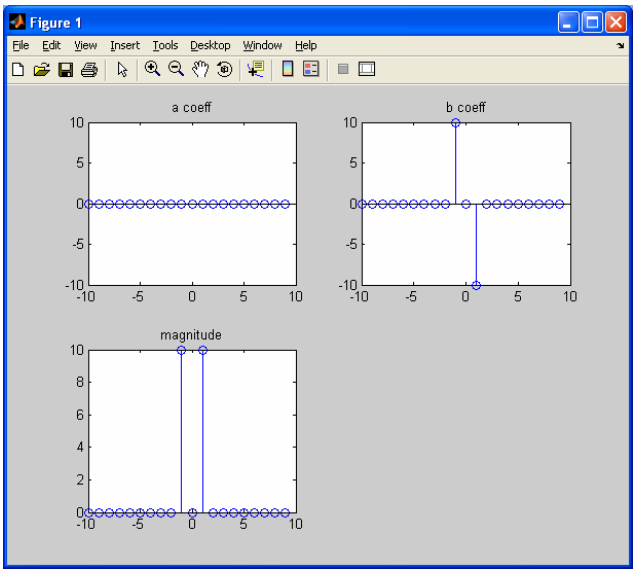
subplot(223); % left, lower corner of 2x2 plot
stem(f,fftshift(abs(yf))); % magnitude of fft coefficients (= sqrt(a^2 + b^2))
title('magnitude');

- You should now see a plot as shown below. Notice that for the sine wave signal, the **a** coefficients are all essentially zero (notice the scale is 1e-15), whereas the **b** coefficients have non-zero values only for index values of +1 and -1 (which correspond to frequencies of +1/20 and -1/20). Notice that the magnitude plot is even symmetric about zero frequency, and the phase plot is odd symmetric about zero frequency.



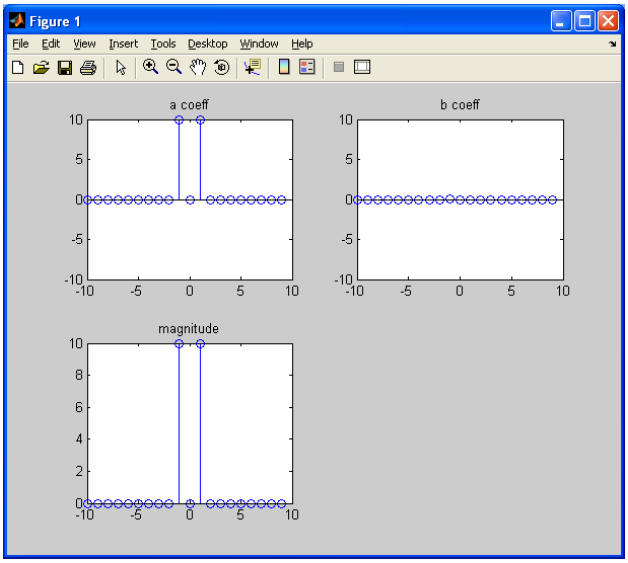
- To better see when the **a** or **b** coefficients are essentially zero, add the following **axis** commands (which are in bold) to the above script. Upon re-running the script, you should see the plot shown below.

….
title('a coeff');
axis([-10 10 -10 10]);
….
title('b coeff');
axis([-10 10 -10 10]);
…..



- Now change y in the above script from a sine to a cosine waveform and then re-run the script to the plot shown below:

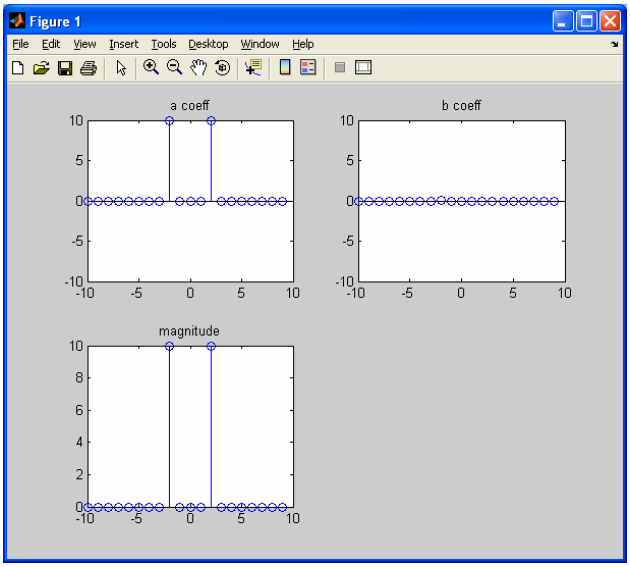
  …..
  y = cos(2*pi*1/20*t); % cosine wave with frequency 1/20
  …..



  - The above plot reveals that the **b** coefficients are now zero.

- Finally, double the frequency of the cosine waveform and re-run the script to get the plot shown below:

  …..
  y = cos(2*pi*1/10*t); % cosine wave with frequency 1/10
  …..



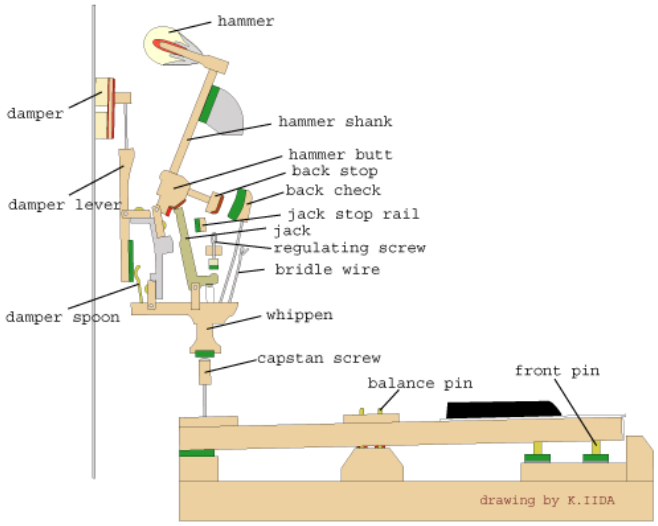
- The above plot shows that the frequency content of the signal has shifted to index values of +2 and -2 (which correspond to frequencies +2/20 and -2/20).

- As we complete our exercise on the **fft** function, we mention a few things that will useful to know in the remaining exercises in this lab:
  - Notice that we did not bother plotting the **phase** of the fft coefficients in the above examples. For many applications, the **magnitude** of the **fft** is of primary importance, and, in such cases, we rarely look at the **phase** or even the **a** and **b** coefficients themselves.
  - Since the **magnitude** is always even symmetric, we often just focus on the *positive* frequency range.

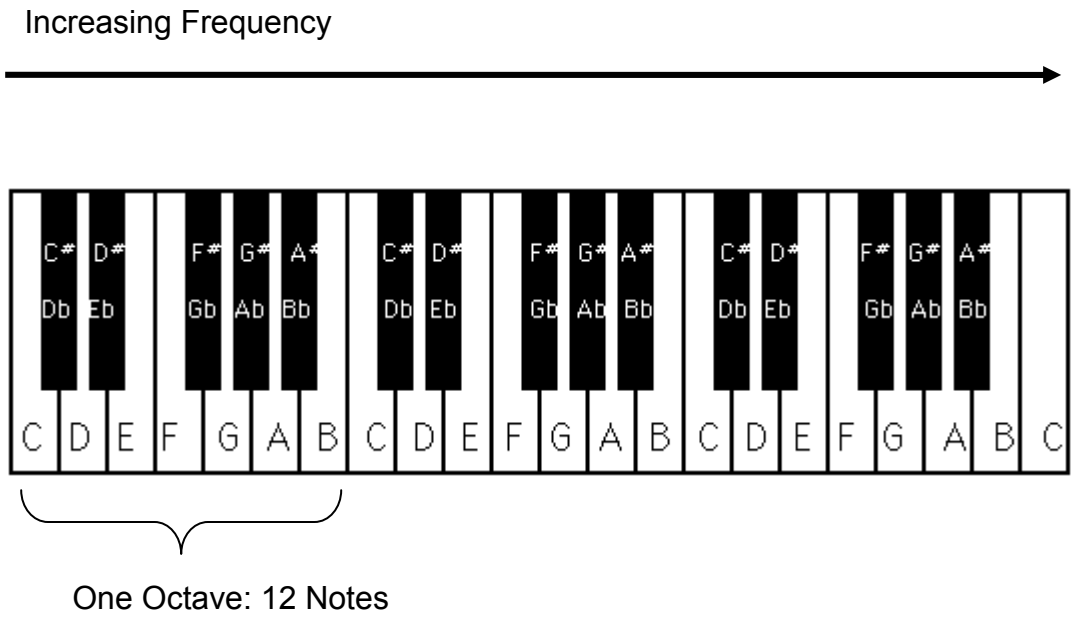
Lab Exercises (2pm – 5pm, Wed., February 14, 2007):

Background on Piano Music

Pressing a piano's key causes a hammer to strike a set of taut strings; the hammer then falls away from the strings so as not to dampen their vibration. The vibration of the strings produces the sound that we hear and recognize as the strike of a piano's key. The quality of the sound we hear is determined by the *amplitude* and *frequency* of the string's vibration, which in turn are related to the length, diameter, tension and density of the string.

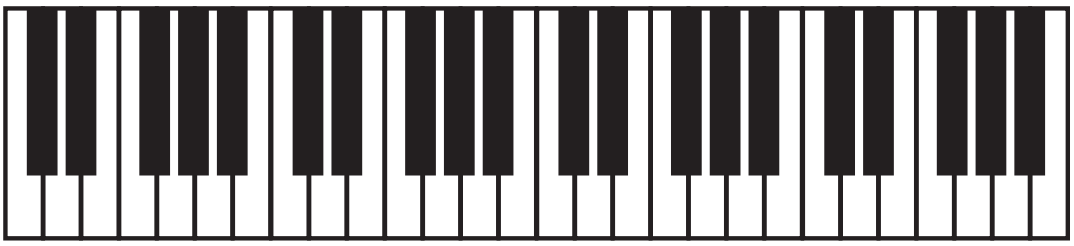


Most pianos have 88 keys with the leftmost key producing the lowest frequency sound. The keys on a piano are arranged in a repeating pattern of 12 keys (7 white and 5 black) as shown below.



Each of the 12 keys in a pattern corresponds to a note in the twelve-note Western Music Scale. The seven white keys correspond to the seven *natural* notes (C, D, E, F, G, A, B); and the remaining 5 black keys correspond to the *sharp* and *flat* notes ($C^{\#}$/Db , $D^{\#}$/Eb, $F^{\#}$/Gb, $G^{\#}$/Ab, $A^{\#}$/Bb). In this lab, we will focus only on the seven natural notes.

A note that is one *octave* above another has a frequency that is *2* times the frequency of the lower note. To distinguish between the different octaves, we often add a subscript to each note to indicate the octave it belongs to. For example, the leftmost C_1 note on a piano has a frequency of 32.7 Hz, whereas the C_2 note is one octave above C_1 and has a frequency of 65.4 Hz. The figure below lists the frequencies of the keys that will be of interest in this lab.



C_2: 65.41 Hz	C_3: 130.81 Hz	C_4: 261.63 Hz	C_5: 523.25 Hz
D_2: 73.42 Hz	D_3: 146.83 Hz	D_4: 293.66 Hz	D_5: 587.33 Hz
E_2: 82.41 Hz	E_3: 164.81 Hz	E_4: 329.63 Hz	E_5: 659.26 Hz
F_2: 87.31 Hz	F_3: 174.61 Hz	F_4: 349.23 Hz	F_5: 698.46 Hz
G_2: 98.00 Hz	G_3: 196.00 Hz	G_4: 392.00 Hz	G_5: 783.99 Hz
A_2: 110.00 Hz	A_3: 220.00 Hz	A_4: 440.00 Hz	A_5: 880.00 Hz
B_2: 123.47 Hz	B_3: 246.94 Hz	B_4: 493.88 Hz	B_5: 987.77 Hz

Starting Matlab and creating a Lab2 Directory

- Choose a lab partner and a PC to work on.
- Log in to your Athena account and then type the following commands within the *same* shell window to start Matlab:

  setup 6.082
  cd
  cp –rf /mit/6.082/Labs/Lab2 .
  cd 6.082
  su (password: 4$jk88*)
  matlab &

- Within the Matlab execution window, type the command:

  cd Lab2

Time-Domain Analysis of Piano Notes

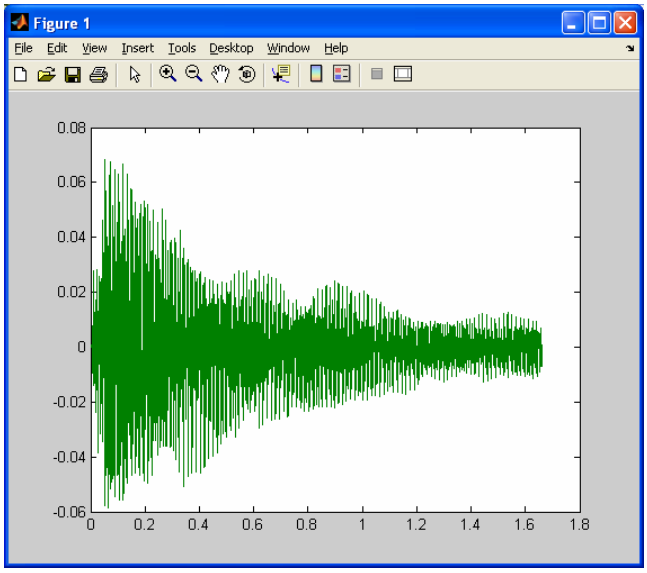
We begin our analysis of piano notes by looking at their time-domain waveforms. To do so, we'll load a snippet from a song played from a piano and examine its basic structure.

- In Matlab, execute the following commands (again, no need to type the comments after the **%** symbols):

  Fs = 22050; % sample frequency of music snippet
  load Sample_1; % load in music snippet file
  soundsc_linux(Sample_1,Fs); % play the music snippet on the headphones

t = (1:length(Sample_1))/Fs;
plot(t, Sample_1); % plot the waveform

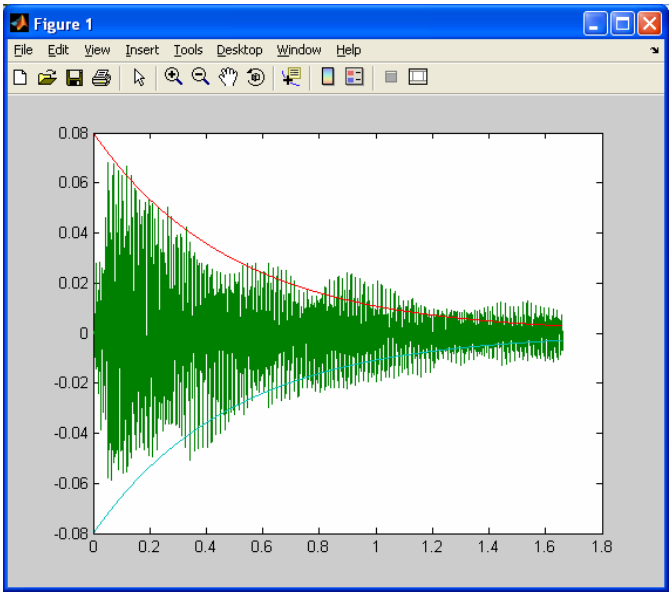
- Upon executing the above commands, you should hear the note on the PC headphones and then see the waveform as shown below:



- We see from the plot above that the amplitude of the signal decays with time. Let us now consider modeling this decay as exponential damping. To do so, we need to figure out the time constant of the exponential decay, **tau**. In this case, we will simply tell you that **tau = 0.5** seconds ends up being a pretty good fit. To verify that this value works pretty well, type in the following Matlab commands:

tau = 0.5; % the time constant value we have given you
r = 0.08*exp(-t/tau); % create exponential decay waveform
plot(t, Sample_1, t, r, t, -r);

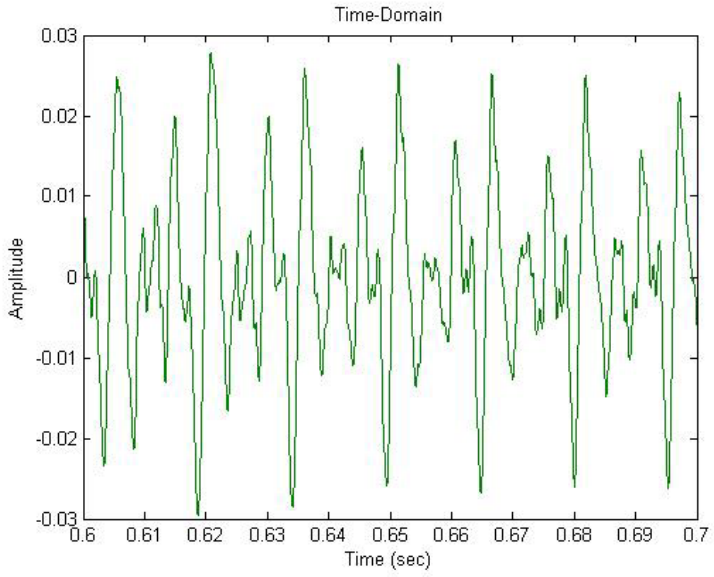
- Upon executing the above commands, you should see the plot below. Note that while the assumption of exponential damping is not a perfect fit, it is a nice engineering approximation that we will assume for the rest of this lab.



- Now let us look more closely at the above signal by zooming in on the signal segment between 0.6 and 0.7 seconds. At the Matlab prompt, type

  axis([0.6 0.7 -0.03 0.03])

  which yields the plot below:



- From the zoomed-in plot above, we observe that the signal is periodic in nature. Unfortunately, it is difficult to figure out the frequencies associated with the above signal based on its time-domain plot. That is a task better suited for frequency-domain analysis, as we will now explore.

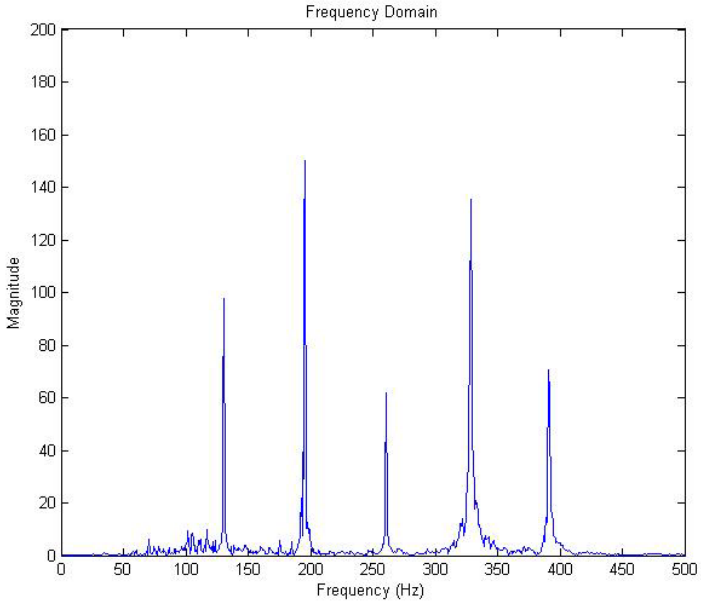
Frequency-Domain Analysis of Piano Notes

- We will now make use of the Matlab **fft** function to more directly examine the frequency content of the piano note examined in the previous section. In particular, we will be looking at the magnitude of the **fft** output, and will focus on positive frequencies when doing plotting.

- Assuming you just ran the Matlab commands from the previous section, continue by typing in the following commands:

```
yf = fft(Sample_1);                  % perform fft on music snippet
f = (0:length(yf)-1)*Fs/length(yf);  % We will not be using fftshift() to view
                                     % negative frequencies, so frequency
                                     % should span 0 to Fs Hz with a little
                                     % adjustment to keep length same as yf.
plot(f,abs(yf));
axis([0 500 0 200]);                 % look at frequency range from 0 to 500 Hz
```

  - The resulting frequency-domain plot shown below reveals the frequencies of the periodic waveform displayed in the previous section. In particular, we see that there are sine/cosine components present at several different frequencies, and that the relative magnitude at those different frequencies varies significantly from each other. Without having phase information, we cannot tell what the *relative* contribution of *sine* versus *cosine* components are in making up the overall magnitude at each frequency value. The good news is that the phase information is essentially irrelevant in terms of how you will hear the music on the headphones. Therefore, for the analysis to follow, we will simply assume that only sine waveforms are present and ignore any cosine contribution. This decision is fairly arbitrary – we could just as easily have chosen cosine components instead.



- Looking at the above plot in more detail, we see that the key frequency components are at 130Hz, 195Hz, 260Hz, 330Hz, and 390 Hz (we'll make this fact more clear in the section below). Using the list of key frequencies shown in the figure on page 9, we determine that the musical notes corresponding to the observed frequency components are C3, G3, C4, E4, and G4. Observe that the note pairs $\{C_3\ C_4\}$ and $\{G_3\ G_4\}$ differ in frequency by a factor of two, and that the amplitude of $C_3 > C_4$ and the amplitude of $G_3 > G_4$. This implies that C_3 is a *fundamental note* and C_4 is its *second harmonic* (or overtone); and that G_3 is another fundamental note and G_4 is its second harmonic.

- One can see from the above exercise that frequency-domain analysis offers a very powerful tool in examining the *structure* of a given waveform. As an example, knowledge of the various frequencies shown in the above plot allowed us to figure out *which* piano keys were pressed. Such a task would be very difficult when looking at the time-domain view of the signals. However, the time-domain view offers a different advantage – one readily sees the exponential damping of the signal (at least in an approximate manner), and we gain a sense of *how* the piano note decays in time. Therefore, one should consider time and frequency domain analysis as being *complementary* – each offers its own advantages when trying to observe the structure of a signal.

Signal Modeling and Synthesis of Piano Notes

The power of understanding the *structure* of a signal is that we can use such knowledge to build a *model* of the signal, which in turn can be used to *synthesize* similar signals. In this lab, we will seek to build a simple model of the waveforms produced when various piano keys are played, and then use that model to construct our own version of a piano song. We start with a model based on Fourier concepts, and then simplify it further in order to make the synthesis task a bit easier.

Fourier concepts tell us that we can model the piano note we examined above as the sum of weighted sine and cosine components. However, to accurately represent the decaying amplitude of the piano notes with *non-decaying* sine and cosine waveforms, we would need to have a large number of those sine and cosine components. Therefore, let us instead consider directly modeling the decaying amplitude by assuming that the piano notes consist of a small number of sine waveforms with *exponential damping* (as discussed in Prelab). As mentioned earlier, the irrelevance of phase for this application means that we could pick either sine or cosine waveforms to work with – our choice of using sine waveforms is fairly arbitrary. In any case, we now express our signal model mathematically as:

$$y(t) = \sum_{i=1}^{N} b_i \sin(2\pi f_i t) e^{-t/\tau}$$
$$= b_1 \sin(2\pi f_1 t) e^{-t/\tau} + b_2 \sin(2\pi f_2 t) e^{-t/\tau} + \bullet\bullet\bullet + b_N \sin(2\pi f_N t) e^{-t/\tau}$$

where y(t) is assumed to be the piano note signal of interest. We call this a *parametric signal model* since we need to determine the values of its corresponding parameters in order to represent the given signal. In the above case, the parameters of the model are the number of sinusoids N; the amplitude of each sinusoid b_i; the frequency of each sinusoid f_i; and the rate τ at which sinusoids decay.

Let us use this model to reconstruct the piano note we have been investigating. To help us with this task, we will use a Matlab function called **ginput**. You may want to type **help ginput** at the Matlab prompt to get more information on this command.

- Assuming that you just ran the Matlab commands from the previous section so that the **Figure 1 plot** appears as shown on the previous page, continue by typing in the following command:

  [f,b] = ginput(5)

  - After executing the above command, place the cursor within the **Figure 1 plot** window and then left-click on each peak. After you have clicked on each of the 5 peaks, you should see values for the **f** and **b** vectors appear which are similar in value to:

    f = [130 195 261 328 390]
    b = [98 150 61 135 70]

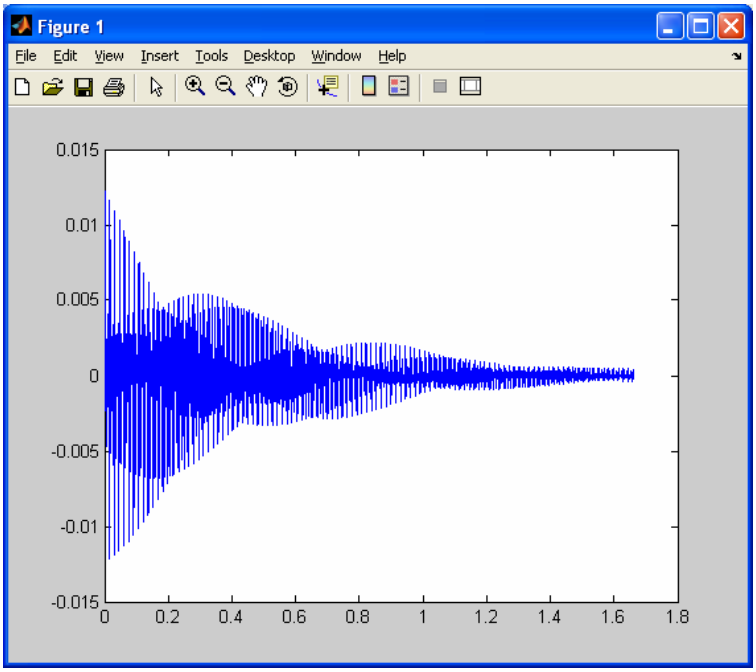
  - Note that the **f** vector values are in Hz, and the **b** vector values have significance only with respect to their *relative* values for our purposes here.
  - We can relate the above frequency values to those associated with the various piano keys. Using the frequency table shown on page 10, we obtain:

    130 Hz: C_3, 195 Hz: G_3, 261 Hz: C_4, 328 Hz: E_4, 390 Hz: G_4

- Let us now plug in the above parameter values to the parametric model discussed above. Within Matlab, continue the previous set of commands by executing:

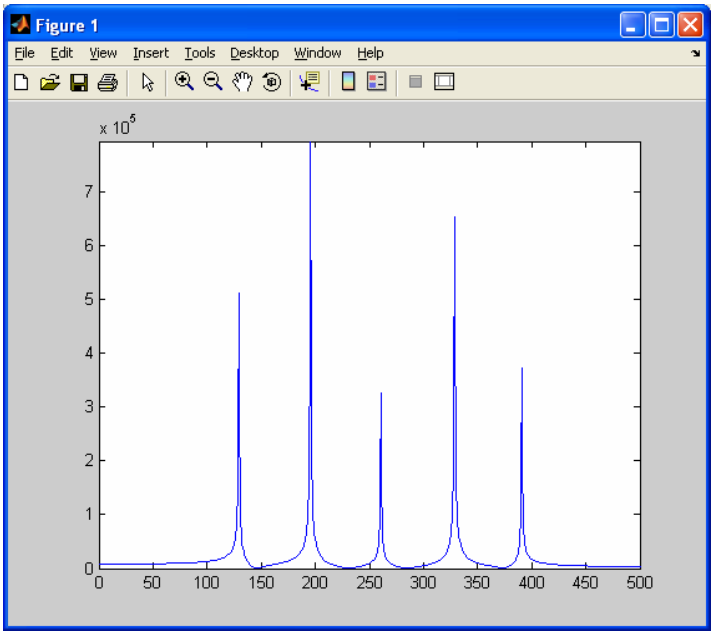
```
tau = 0.5;   % note that we provided you with this value of tau earlier
ysin = b(1)*sin(2*pi*f(1)*t) + b(2)*sin(2*pi*f(2)*t) + b(3)*sin(2*pi*f(3)*t) +
       b(4)*sin(2*pi*f(4)*t) + b(5)*sin(2*pi*f(5)*t);
y = ysin.*exp(-t/tau);
soundsc_linux(y,Fs);
plot(t,y);
```

  - You should hear the synthesized piano note in the headphones, and you should see a plot similar to what is shown below:



  - To see the frequency content of the above waveform, type the following command to see the plot shown below. Note that the absolute magnitudes are significantly different than the actual piano music, but the relative magnitudes are closely matched. In the interest of time, we're not going to get into the issue of why the absolute magnitudes are different since it will not affect the key results we seek.

```
yf = abs(fft(y));
f = (0:length(yf)-1)*Fs/length(yf);
plot(f,yf);
axis([0 500 0 max(yf)]);
```

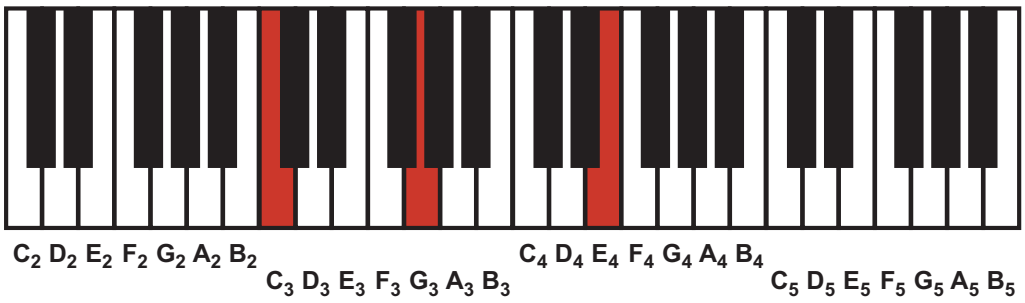


- One should note that the sound is lacking the richness of the actual piano note. To hear the difference, type **soundsc_linux(Sample_1,Fs)** at the Matlab prompt to hear the actual piano noted again. The difference is due to the fact that we are approximating the piano note with only 5 sine waves and are ignoring frequency content above 500 Hz. However, for the purposes of this lab, we'll assume our approximation is close enough to get the basic ideas across.

An Even Simpler Model for our Synthesized Piano Notes

While we could directly use the parameterized model from the previous section to analyze the piano song we are about to examine, it would be a bit dull to simply pick off frequency and amplitude values using **ginput()**. Instead, we seek a more physically motivated parameterization that will allow us to connect the waveforms we see to the keys actually pressed on the piano.

In the previous section, we determined that the frequencies from **Sample_1** correspond to piano notes C_3, G_3, C_4, E_4, and G_4. However, it is important to note that C_4 and G_4 are second harmonics of C_3 and G_3, respectively. Therefore, we can think of C_3, G_3 and E_4 as *fundamental notes* and C_4, G_4 as second harmonics or overtones. The musician very likely only played the fundamental notes C_3, G_3, and E_4, as indicated in the figure below. We know this because it is common to play notes in groups of three, which are known as *chords* or *triads*. The harmonics of each fundamental note are heard because striking the piano string associated with a fundamental note sets up string vibrations at the fundamental note frequency and at integer multiples of that frequency.



C_2: 65.41 Hz	C_3: 130.81 Hz	C_4: 261.63 Hz	C_5: 523.25 Hz
D_2: 73.42 Hz	D_3: 146.83 Hz	D_4: 293.66 Hz	D_5: 587.33 Hz
E_2: 82.41 Hz	E_3: 164.81 Hz	E_4: 329.63 Hz	E_5: 659.26 Hz
F_2: 87.31 Hz	F_3: 174.61 Hz	F_4: 349.23 Hz	F_5: 698.46 Hz
G_2: 98.00 Hz	G_3: 196.00 Hz	G_4: 392.00 Hz	G_5: 783.99 Hz
A_2: 110.00 Hz	A_3: 220.00 Hz	A_4: 440.00 Hz	A_5: 880.00 Hz
B_2: 123.47 Hz	B_3: 246.94 Hz	B_4: 493.88 Hz	B_5: 987.77 Hz

- Continuing our chain of Matlab commands, let us now change our parameterized model for the **Sample_1** snippet to be based on three fundamental keys and their second harmonics. As a crude approximation, we'll simply assume that the fundamentals have the same amplitude, and that the harmonics have 1/2 the amplitude of their respective fundamentals. Within Matlab, type:

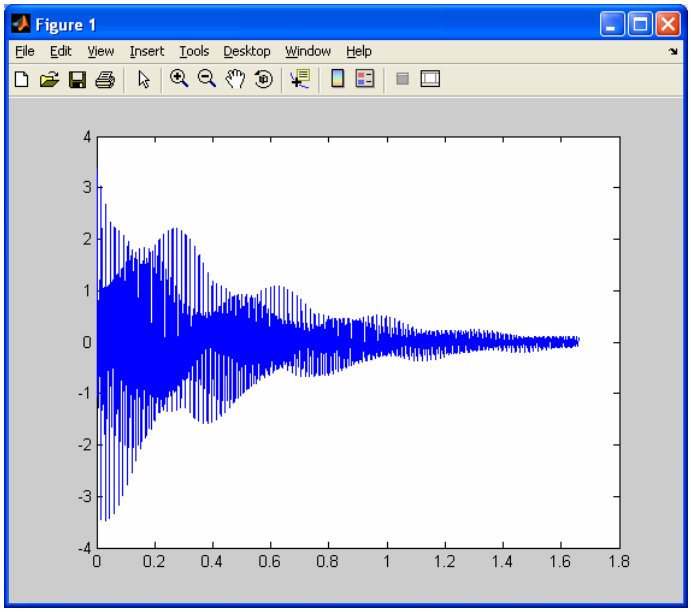
```
c3f = 130.81;   % frequency of note c3
g3f = 196;      % frequency of note g3
e4f = 329.63;   % frequency of note e4
hm = 0.5;     % relative magnitude of harmonic

c3h = sin(2*pi*c3f*t) + hm*sin(2*pi*2*c3f*t); % sine wave: c3 & harmonic
g3h = sin(2*pi*g3f*t) + hm*sin(2*pi*2*g3f*t); % sine wave: g3 & harmonic
e4h = sin(2*pi*e4f*t) + hm*sin(2*pi*2*e4f*t);  % sine wave: e4 & harmonic

c3 = c3h.*exp(-t/tau);  % include exponential damping
g3 = g3h.*exp(-t/tau);
e4 = e4h.*exp(-t/tau);

y = c3 + g3 + e4;
soundsc_linux(y,Fs);
plot(t,y);
```

  - Execution of the above commands should allow you to hear the synthesized notes on the headphones. You should also see a plot similar to what is shown below.



A Composition Script for Matlab

For the exercises to follow, you will be filling into a Matlab script which provides you with variables for all of the piano notes of interest. In particular, you will be using the skills you have learned thus far in this lab to analyze a short piano song and then synthesize it using the simplified signal model introduced in the previous section. Before we describe the exercises below, let us first look at this script so that you get a feeling of what it provides.

- In Matlab, type:

edit compose_song.m

  - The first portion of this file contains same basic parameters which include the time duration of note, the exponential damping time constant, and the relative harmonic magnitudes. As stated in the file, you'll initially keep these values unchanged as you complete the first exercise. Note that **duration** simply corresponds to the approximate time that a note is played before a new note occurs – examination of the above plot shows that 1.6 seconds is a reasonable estimate of this value.

```
duration = 1.6;    % time duration of each note (in seconds)
tau = .5;          % damping time constant
hm = 0.5;          % relative magnitude of harmonic
```

  - The second section provides variables for each of the relevant piano keys which are composed of their fundamental and second harmonic waveforms

and are labeled **c2**, **d2**, **e3**, etc. These variables will be used to construct the synthesized song that we will seek.
    - The third section provides placeholders for the chords that you will be figuring out in the exercises below. The value for **chord1** is already provided, and corresponds to the **Sample_1** snippet that we examined earlier.
    - The final section simply strings together the chords, plays them on the headphones, and plots the time-domain waveform of the song.

Exercises

For the exercises below, please fill in the requested values on the check-off sheet located at the end of this document. Note that the last exercise is open-ended and subjective in nature, and therefore requires a check-off directly by the TA before you leave.

Exercise 1

Here you will listen to eight notes from a piano song, determine which notes are being played, and then synthesize the song using the simplified parameterized model from Section F. To do so, you will be running one script to hear and analyze the song (**play_song.m**), and filling in a different script (**compose_song.m**) to synthesize the song. You will be limited to three notes per chord – note that these notes may be harmonically related to each other in some cases.

- In Matlab, type:

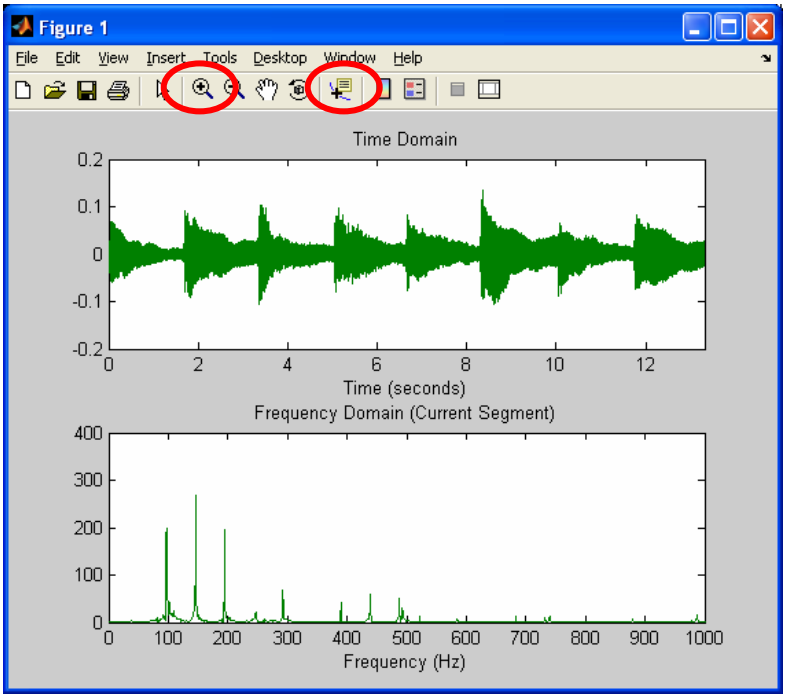
    play_song

    - You should hear eight chords of a song, and the **Figure 1 plot** window should show both the time and frequency domain views of the song. The time-domain view progresses along and keeps a record of the entire song as it plays. The frequency-domain view shows the **fft** results one chord at a time. Re-run the above script a few times to hear the song and see the information that it provides.
- Now in Matlab, type:

    edit play_song.m

    - Within the edit window, uncomment the **pause** statement that is close to the end of the script. This will allow you to step through the chords one at a time. Be sure to save the file when you are done.
- Re-run the **play_song** script in Matlab. You will notice that the script now pauses after each note. In order to progress to the next note, you must push a key on the PC keyboard (the **spacebar** is a good choice for this).
- You have three choices for picking off the frequency values from the Matlab plot. One is to use the zoom button within the **Figure 1 plot** window (which appears as a magnifying glass with a plus symbol inside of it, as circled on the left below).

The next is to hit Ctrl-C in order to stop the script at a given place, and then use **ginput** as discussed earlier. The third is to click on the button circled below on the right within the **Figure 1 plot** window, which acts in similar fashion to **ginput**.



1. Given the information that you see in the **Figure 1 plot** window for each chord, determine **3** appropriate piano keys for each chord played. Assume that each key has equal magnitude as assumed for **chord1** in Section F. Fill in your answers in the check-off sheet at the end of this document.
2. Given the piano key keys determined above, edit the **compose_song.m** file to synthesize your song. Once you feel confident that your synthesized song sounds OK given your limited modeling constraints, call over a TA to get checked off on this exercise.

Exercise 2

Here we make the task more open-ended, and simply ask you to try to improve the signal modeling within the **compose_song.m** file in order to make the synthesized song sound truer to the actual song. However, you must limit yourself to 3 piano notes per chord. Therefore, the parameters you have available to change are as follows:

- **duration**, **tau**, and **hm**: these allow you to change the duration of the notes, the time constant of the exponential damping, and the relative magnitude of the second harmonic.
- The relative scaling of notes: we previously constrained you to chords being composed of equal amplitude notes (i.e., **c3 + g3 + e4**), but you may now weight them differently (i.e., **0.7*c3 + g3 + 0.7*e4**).

- The number of harmonics associated with each note: we have thus far only included the second harmonic – you may also add additional harmonics if you like and scale them by whatever factor you like. However, all harmonics must be consistently scaled relative to their fundamental (i.e.., all second harmonics must be scaled relative to their fundamental by factor **hm**, and all third harmonics would need to be scaled relative to their fundamental by factor **hm3,** etc**.**).

- When you are satisfied with your improved song, call over a TA to get checked off on this exercise before leaving lab.

PostLab (Due on Friday, February 16, 2007)

_____
Student Name

1. Is it easier to look at the issue of exponential damping in the time or frequency domain? Why?
2. Is it easier to determine *which* note was played in the time or frequency domain? Why?
3. If the time constant of exponential damping is reduced, does that imply that the note lasts for a longer or shorter period of time?
4. How are the decaying exponentials discussed in this lab different from complex exponentials used in the definition of Fourier Tranforms?
5. Define what a parametric model is and explain why it is useful.
6. How might you have improved the parametric model of musical notes further to achieve an even closer match to the actual musical notes?
7. How might you apply what you learned in this lab to looking into the areas of speech recognition and automatic note detection for arbitrary musical pieces? Are these difficult problems? Why?

We invite you to have more fun with this lab by continuing to improve your song on Athena.

Check-off for Lab 2

_____ _____

Student Name Partner Name

Piano notes for Exercise 1 (only 3 per chord are allowed):

Chord 1: c3 g3 e4

Chord 2: _____

Chord 3: _____

Chord 4: _____

Chord 5: _____

Chord 6: _____

Chord 7: _____

Chord 8: _____

Check-off for synthesized song in Exercise 1

_____

TA Signature

Check-off for synthesized song in Exercise 2

_____ _____

TA Signature Score