DAQ 4:
Aliasing and Frequency Space

4.1 Introduction

For this laboratory, you will be further investigating the use of the DAQ card to sample and store musical information. Specifically, you will be able to see and hear what was happening with the different sampling rates from the previous audio lab and will learn what the term "aliasing" means with respect to sampling data. You will also learn how to manipulate audio information in the frequency domain and convert signals between their time and frequency representations.

4.2 Resources

The additional resources required for this assignment include:

- Books: None
- Pratt Pundit Pages: `MATLAB:CB-68LP Pinout`, `http://pundit.pratt.duke.edu/wiki/EGR 53/DAQ Audio 2`
- Lab Manual Appendices: None

4.3 Getting Started

1. Each lab group should have one person sit at the PC that has the patch of tape (blue, green, or red) on it. This is **not** the same as the strip with the computer's name on it - it will literally be a small patch of tape all by itself.

2. Set the machine to "Log on to" `...(this computer)` and use the "User name" `mrglocal` and the "Password" `p1p2dell` (that is p-ONE-p-TWO-d-e-l-l). The other lab partner can log into the other computer either using the `mrglocal` account or any other valid NET ID (as log as the "Log on to" is set to `Kerberos`). You do *not* need to run X-Win or PuTTY at all on either computer.

3. Start MATLAB on the PC with the tape on it. The first time MATLAB is run, it may take some time to start up.

4. Once MATLAB starts, make sure the `Current Directory` listed at the top of the MATLAB window is

   `C:\Documents and Settings\labuser\My Documents\MATLAB`

5. In the `Current Directory` window on the left, select and delete any files in the `MATLAB` directory.

4.4 Equipment

The lab this week requires essentially the same data acquisition system as the first audio lab week. You will also need headphones and an audio source. You do not have to use the same audio clip nor do you have to be paired up with the same person as last time, however. The network listing is given on page DAQ 4 – 10 in Table 4.1. You will need one red wire and one black wire in addition to a screwdriver.

4.5 Scripts

For this week, some parts of the scripts you need have already been written. You will be making modifications to them in order to understand aliasing and frequency space. The files are at:

`http://www.duke.edu/~mrg/DAQS/DAQaudio2/`

and are called `AliasDemo.m`, `freq1.m`, `freq2.m`, `EqualizerDone.m`, and `MaskMusic.m`. You should also copy the `MyChirp.mat` file, which contains a sound that sweeps through frequencies from 0 Hz to two octaves above middle A (1760 Hz). Open a web browser (IE), point to the URL above, and then right click on the file you want. Choose `Save Target As...` and save the files in

```
C:\Documents and Settings\mrglocal\My Documents\MATLAB
```

Copies of the script files are at the end of this handout so you can take notes regarding how they work.

4.6 Frequency Space

One method of interpolating between points uses polynomial powers to represent the data - this is the polynomial fit that you used. To recap, if there is some data set $x(n)$ that has N known values for evenly spaced, integer values of n, we could use an $N-1$th order polynomial:

$$F(n) = P(1)\ n^{N-1} + P(2)\ n^{N-2} + ... + P(N-1)\ n + P(N) = \sum_{k=1}^{N} P(k)\ n^{N-k}$$

Now there are two fundamentally different ways to represent the data - either as a table of values for a range of n's or as a set of coefficients P. While the former case tells exactly what value the function takes on at a particular time or location (depending on the meaning of n), the latter version may give a better overall impression of the shape of the function that fits the data. For example, given the following three points:

n	1	2	3
$x(n)$	1	3	0

you could use the MATLAB commands:

```
n = [1 2 3];
x = [1 3 0];
P = polyfit(n, x, 2)
```

find the coefficients $P = [-2.5, 9.5, -6]$. This means the data can also be represented by the quadratic equation:

$$F(n) = -2.5n^2 + 9.5n - 6$$

From this equation, it is clear that the data forms a parabola that opens down (negative coefficient on n^2 term) and has an x intercept of -6 (coefficient on n^0 term). If this is the information you were looking for, the polynomial representation is more useful than the time or space based one.

For musical analysis and manipulation, generally it is the *frequency* content that is most interesting. Given that, another method of interpolating points can be used that is based on cosines and sines at different frequencies rather than on different orders of polynomials. The formula for this is decidedly more complex:

$$F(n) = \frac{1}{N} \sum_{k=1}^{N} X(k) e^{\frac{j2\pi(k-1)(n-1)}{N}}$$

where the $X(k)$ are complex numbers. To explain this formula, using Euler's relation, which states that:

$$e^{j\theta} = \cos(\theta) + j\sin(\theta)$$

and you can rewrite the interpolation as

$$F(n) = \frac{1}{N}\sum_{k=1}^{N} X(k) \ \left(\cos\left(\frac{2\pi(k-1)(n-1)}{N}\right) + j\sin\left(\frac{2\pi(k-1)(n-1)}{N}\right)\right)$$

The argument for the trig terms can be re-written as:

$$\frac{2\pi(k-1)(n-1)}{N} = \frac{2\pi}{N}(k-1)(n-1) = \omega_0(k-1)(n-1)$$

where ω_0 is the *fundamental frequency* for the interpolation, making $\omega_0(k-1)$ is the frequency of the kth term in the summation. This leaves an interpolation function of:

$$F(n) = \frac{1}{N}\sum_{k=1}^{N} X(k) \ \left(\cos\left(\omega_0(k-1)(n-1)\right) + j\sin\left(\omega_0(k-1)(n-1)\right)\right)$$

Though this is a complicated formula, the most important part to notice is the frequency of the individual terms, $\omega_0(k-1)$. The magnitudes of $X(k)$ are therefore relative measures of the importance of a particular frequency within the signal. Furthermore, from the summation, it is clear that there is a maximum possible frequency that can be represented. In fact, this is a *little* misleading, in that the highest frequency you will be able to represent with the summation is not $\omega_0(N-1)$ but half that (or slightly less), as will be demonstrated with the first three script files you copied.

Despite the algebraic complexity of this particular interpolation scheme, it gives a method by which some time (or space) series $x(n)$ can be represented by values indicating how much of a particular frequency (the magnitude of $X(k)$) exists in a particular signal. This method of representing the data uses the *Discrete Fourier Transform* (or DFT). As mentioned, the $X(k)$ will generally be complex numbers with the angle of the complex number indicating the phase of that particular frequency. For example, a phase of 0 degrees represents a pure cosine at a particular frequency. The phase information can be extremely important to engineers but for today, only the magnitudes will be described.

4.7 Demonstration of the Influence of the Sampling Rate

To begin the study of sampling rates, run the `AliasDemo` script. In all five cases, the same signal:

$$y = \cos(20\pi t)$$

which has a fundamental frequency f_0 of 10 Hz or fundamental angular frequency $\omega_0 = 2\pi f_0$ of 20π rad/s. On the left side, the signal is shown as a black line for 1 second. To make the line, the signal was sampled 500 times per second - this is called the *sampling frequency* or *sampling rate*, and 500 Hz was chosen because it is more than sufficient to show the true nature of a 10 Hz signal.

4.7.1 Reconstruction Using Samples

The red dots are samples taken at different rates ranging from a maximum of $f_s = 100$ Hz to a minimum of $f_s = 2$ Hz. These samples are then plotted using MATLAB's built-in interpolation scheme for connecting data points and displayed on the right side of the screen. Note that even though each set of data is sampled from the same original equation, the reconstructions look quite different depending on the sample rate.

The first case, where $f_s = 100$ Hz, is a fairly good representation of the signal. We are taking 10 samples for each period of the sinusoid, so there is enough information to reconstruct a reasonable approximation to the signal. In the second case, the sampling rate of 20 Hz means we are only taking two samples per period. For a cosine, this means one at the peak and one at the trough for each period. While we have lost all semblance of curvature, the signal still appears to have the same period. The apparent period changes, however, as the sampling rate is further reduced.

4.7.2 Ramifications of Reducing the Sample Rate

When $f_s = 15$ Hz, we are no longer taking enough samples per cycle to capture the essence of a single period. If you look at the reconstructed graph, not only is it blocky, the apparent period has somehow doubled! Decreasing the sampling rate further, when $f_s = 11$ Hz, or just above the signal frequency, the appearance of the reconstructed signal is of a fairly convincing cosine - at one tenth the original frequency. In the last pairing, when $f_s = 2$ Hz (meaning one sample is taken for every five periods of the original signal), the reconstructed signal appears to be a constant value.

In the last four cases, a phenomenon known as *aliasing* has occurred; it is called aliasing because one frequency is masquerading as another due to the sampling rate. The tipping point for aliasing is when the sampling rate is double or less than the highest frequency component in the signal. Any part of the signal oscillating faster than half the sampling frequency will, in the reconstructed signal, appear at a different, lower frequency. The name given to the lowest frequency that will avoid aliasing for a given signal is the *Nyquist rate* of the signal, which is defined as twice the maximum frequency contained in the signal.[1]

4.7.3 Determining Aliased Frequencies

An interesting part of aliasing is how to determine the frequency to which a particular signal is aliased. When the sampling rate is more than twice the fastest part of a signal, the reconstructed frequencies are the same as the original. For parts of the signal at *exactly* half the rate, there can be deconstructive interference as a result of phase. Note that when $f_s = 20$ Hz, the signal is basically at 0 the entire time (the nonzero magnitude is a result of roundoff error).

As the sampling rate of the system decreases below the Nyquist rate of the signal, the reconstructed frequency begins to *decrease* linearly until the apparent frequency is 0 Hz. For example, when $f_s = 15$ Hz, the maximum frequency signal that can be reconstructed properly is at 7.5 Hz. The signal is at 10 Hz, which is 2.5 Hz faster than this sampling rate can handle. The reconstructed signal will end up resembling a frequency that is 2.5 Hz *slower* than the 7.5 Hz maximum, or (7.5-2.5)=5 Hz. It will also have a different phase - notice that the signal sampled at 15 Hz starts at zero and goes *down* first.

Continuing to slow down the sampling rate, when $f_s = 11$ Hz, the maximum frequency this can capture properly is 5.5 Hz. The signal is 4.5 Hz faster, so the reconstructed signal is 4.5 Hz slower than the maximum possible - or (5.5-4.5)=1 Hz. If the 10 Hz signal were sampled at 10 Hz, the apparent frequency of the signal would be 0 Hz.

If the sampling rate is further decreased, the apparent frequency of the reconstructed signal *speeds up again*. Essentially, frequency space folds in on itself, bouncing back and forth between the maximum possible reconstructed frequency ($f_s/2$) and 0 Hz. For the signal sampled at 7 Hz, then, the maximum properly reconstructed frequency is 3.5 Hz. The apparent reconstructed frequency can be calculated by first going out to 3.5 Hz and noting there is 6.5 Hz remaining. Coming back into 0 Hz eliminates another 3.5 Hz leaving 3 Hz to go. The reconstructed frequency is thus 3 Hz and it will have the same phase as the original.

[1] Simon Haykin and Barry Van Veen, *Signals and Systems* (New Jersey: John Wiley and Sons, 2005), p. 374

4.8 Visual Aliasing and Frequency Space Demonstration

What this all means is that by looking at the relative magnitudes of the values of $X(k)$ we can see which frequencies are more prevalent and which are less, but we must also keep in mind some information may be aliased. You will be using the `freq1.m` script to examine this. The code for this script is at the end of this lab. The script takes information about a sampling rate as well as up to three cosine amplitude and frequency pairs. It then plots the signal for two seconds as well as the frequency content for those two seconds.

You will be running the following ten different argument combinations to further your understanding of frequency space as well as aliasing. The instructor will tell you which case to run when. Note that in the ten primary cases, the sampling frequency is 100 Hz, meaning two hundred samples are collected over the two second interval. The sampling rate is changed for the three alternate cases, denoted by an "a."

Case	Code	$x(t)$
1	freq1(100, 1, 0)	1
2	freq1(100, 1, 10)	$\cos(20 * \pi * t)$
3	freq1(100, 1, 50)	$\cos(100 * \pi * t)$
4	freq1(100, 1, 90)	$\cos(180 * \pi * t)$
4a	freq1(200, 1, 90)	$\cos(180 * \pi * t)$
5	freq1(100, 1, 10, 2, 50)	$\cos(20 * \pi * t) + 2\cos(100 * \pi * t)$
6	freq1(100, 1, 10, 2, 80)	$\cos(20 * \pi * t) + 2\cos(160 * \pi * t)$
7	freq1(100, 1, 10, 2, 90)	$\cos(20 * \pi * t) + 2\cos(180 * \pi * t)$
7a	freq1(200, 1, 10, 2, 90)	$\cos(20 * \pi * t) + 2\cos(180 * \pi * t)$
8	freq1(100, 1, 10, 2, 30, 4, 40)	$\cos(20 * \pi * t) + 2\cos(60 * \pi * t) + 4\cos(80 * \pi * t)$
8a	freq1(???, 1, 10, 2, 30, 4, 40)	$\cos(20 * \pi * t) + 2\cos(60 * \pi * t) + 4\cos(80 * \pi * t)$
9	freq1(100, 1, 10, 2, 30, 4, 80)	$\cos(20 * \pi * t) + 2\cos(60 * \pi * t) + 4\cos(160 * \pi * t)$
10	freq1(100, 1, 10, 2, 30, 4, 90)	$\cos(20 * \pi * t) + 2\cos(60 * \pi * t) + 4\cos(180 * \pi * t)$

In the figure window, the top graph shows the sampled and reconstructed signal while the bottom bars represent the magnitudes of the $X(k)$ values given in the DFT equation. What you will see in the ten cases is as follows:

(1) In the first case, the signal in time is a constant value of 1. In the lower plot, there is a single bar at a frequency $f = 0$ Hz. The height of this bar is determined both by the amplitude of the signal (1) and the total number of samples (200).

(2) Running the second case, you will see a cosine at a frequency of 10 Hz in the top window - this relates to 20 periods across the screen since the period T is 0.1 sec. In the bottom window, you will see two bars, one at positive 10 Hz and one at negative 10 Hz. DFTs of real signals generally have matched positive and negative pairs with two notable exceptions. One exception is a DC signal, such as what you saw in Case 1. Note that the bars here in Case 2 are half the height of the single bar from Case 1 - think of Case 1 as having bars at positive0) Hz and negative 0 Hz which therefore happen to be on top of each other at 0 Hz.

(3) The other exception to having matched pairs of bars involves signals at a frequency just on the edge of how fast a particular sampling rate can pick things up. For a sampling rate of 100 Hz, the fastest frequency that can be uniquely identified happens to be 50 Hz. Note that there is a single bar of height 200 at a frequency of -50 Hz. This is an instance where phase would matter - if we were plotting sin instead of cos, one bar would be positive and one would be negative, meaning their overlap would cancel out and you would get no signal, just as in `AliasDemo` when the 10 Hz signal was sampled at 20 Hz.

(4) Case 4 shows what happens when you try to sample a signal going faster than half the sampling rate. In this case, the signal "looks like" a signal going at a slower rate. Given a sampling rate of 100 Hz, a signal at 90 Hz is 40 Hz faster than the fastest signal that can be truly captured by that particular sampling rate. It ends up looking like a signal going 40 Hz away from the 50 Hz maximum, or 10 Hz.

As mentioned earlier, as a signal's frequency increases, its energy would simply bounces back and forth between 0 and 50 Hz (for a sampling rate of 100 Hz).

This case also demonstrates what happened in the previous DAQ lab when sound was sampled at progressively lower rates. Higher frequency sounds started sounding like lower frequency sounds because they were aliased. Furthermore, because aliasing reverses the order of some frequencies, the sound was garbled. Beyond even *that* problem, sound is not a linearly perceived stimulus, so at very low sampling rates the sound came out as bass-only, utterly mis-tuned garbage.

To get rid of aliasing, you would need to sample this signal at 180 Hz or faster. Case 4a demonstrates that, sampled at 200 Hz instead of 100 Hz, the signal looks like a proper 90 Hz signal and its energy is properly located at 90 Hz.

The next cases examine what happens when multiple frequencies are present in a single signal. These more closely represent what is happening to music, since music is truly a combination of a several frequencies.

(5) Case 5 shows two frequencies - 10 Hz and 50 Hz - at two different amplitudes - 1 and 2 respectively. Notice that they do not interfere with each other (the bar heights are what we would expect for the individual components). As long as frequencies are not the same (or aliased to look the same), the energies of those frequencies do not interfere with other frequencies.

(6) Case 6 again shows two frequencies - this time 10 Hz and 80 Hz. Even though the 80 Hz signal is aliased to look like 20 Hz, its energy does not interfere with the 10 Hz signal and vice versa. Looking at the time signal, you can count 20 periods, each having a wobble that seems to be going twice as fast within it.

(7) Case 7 shows two frequencies - 10 Hz and 90 Hz. In this case, the signals do interfere because at a sampling rate of 100 Hz, 90 Hz "looks like" 10 Hz. If you were to run this case with a sampling rate of 200 Hz, the frequencies would be captured independently. Go ahead and run case 7a at that sampling rate to see how different the time signal appears without aliasing.

(8) Case 8 shows three frequencies - 10 Hz, 30 Hz, and 40 Hz - which can all be captured uniquely by a 100 Hz sampling rate. What sampling rates would cause the 10 Hz and 30 Hz signals to overlap? Run the function with that rate.

(9) Case 9 again shows three frequencies - 10 Hz, 30 Hz, and 80 Hz. Even though the 80 Hz is now aliased, because it is aliased to 20 Hz it does not interfere with the energies of the other two frequencies.

(10) Case 10 shows three frequencies - 10 Hz, 30 Hz, and 90 Hz - where the aliased signal does interfere.

Note in all these cases that we did not examine what happens when cosines and sines come together. The issue of *phase* is very important, especially in fields like acoustics. If the `freq1.m` program had been using sines instead of cosines, for example, the interference would have been destructive rather than constructive. For this lab, however, it is with the location of the energy itself rather than the phase that we will be concerned.

4.9 Audio Demonstration

In order to properly capture the frequency information you have to sample at at least *twice* the rate of the highest frequency you want to capture. This fact can be used to explain why CDs sample slightly above 44,000 times per second - human hearing generally only goes up to about 20 kHz, so 44.1 kHz captures everything we can hear (and up to 22.05 kHz) without wasting storage. For example, the sounds with the 200 kHz sampling rates probably did not sound different from the ones sampled at 44.1 kHz. Aliasing also explains why the sounds you sampled last week at rates lower than 44.1 kHz sounded wrong - you were aliasing high frequency information into lower frequencies which made the sounds seem muffled and confused. Sampling at too low a rate does not eliminate the energy in those higher frequencies - it moves the energy to different frequencies.

To *hear* this problem, we are going to use the `freq2.m` program, which is basically the same as the `freq1.m` program, only it adds the capability to play x as a sound. The cases in the table below will be played in lab. Note that the frequencies are increased by a factor of 20. The `freq1` program is good for looking at the signal and seeing the changes in it, but those frequencies are too low to hear well. On the other hand, audio signals oscillate too fast to see what is going on for any appreciable duration, so this set of code is mainly for aural observation.

Case	Code	$x(t)$
1m	freq2(2000, 1, 0)	1
2m	freq2(2000, 1, 200)	$\cos(400 * \pi * t)$
3m	freq2(2000, 1, 1000)	$\cos(2000 * \pi * t)$
4m	freq2(2000, 1, 1800)	$\cos(3600 * \pi * t)$
4ma	freq2(4000, 1, 1800)	$\cos(3600 * \pi * t)$
5m	freq2(2000, 1, 200, 2, 1000)	$\cos(400 * \pi * t) + 2\cos(2000 * \pi * t)$
6m	freq2(2000, 1, 200, 2, 1600)	$\cos(400 * \pi * t) + 2\cos(3200 * \pi * t)$
7m	freq2(2000, 1, 200, 2, 1800)	$\cos(400 * \pi * t) + 2\cos(3600 * \pi * t)$
7ma	freq2(4000, 1, 200, 2, 1800)	$\cos(400 * \pi * t) + 2\cos(3600 * \pi * t)$
8m	freq2(2000, 1, 200, 2, 600, 4, 800)	$\cos(400 * \pi * t) + 2\cos(1200 * \pi * t) + 4\cos(1600 * \pi * t)$
9m	freq2(2000, 1, 200, 2, 600, 4, 1600)	$\cos(400 * \pi * t) + 2\cos(1200 * \pi * t) + 4\cos(3200 * \pi * t)$
10m	freq2(2000, 1, 200, 2, 600, 4, 1800)	$\cos(400 * \pi * t) + 2\cos(1200 * \pi * t) + 4\cos(3600 * \pi * t)$

4.10 Sound Equalization

The workstation should already have the audio cabling in place from the previous DAQ lab. Use the net listing in Table 4.1 on page DAQ 4 – 10 which is a highly simplified version of the circuit from the previous lab. Level the input using the `softscope` oscilloscope emulator, only set your volume such that the voltage peaks at ±250 mV instead of ±400 mV. The reason for this is that later in the lab you are going to be amplifying the signal by up to a factor of 4. MATLAB's `sound` program takes values with values between -1 and 1 only (anything beyond that is chopped to ±1) so the input voltage needs to be limited to ±250 mV.

Once that is done, **close the oscilloscope completely** and then open the `EqualizerDone.m` in MATLAB. This particular function simply adds the ability to look at the frequency content of your sounds to the `MirrorInDone.m` program used last time. The code at the bottom of this script is almost exactly what was used in the `freq1.m` script to see the frequency content of those simpler signals. Examine the file and confirm that it will acquire 10 seconds of your song at a sampling rate of 50 kHz.

Start your audio device, run the program, save the file as `MySound`, and then type `whos`. You should have 500,000 samples to work with. Now let's say you want to modify the influence that a particular band of frequencies - for example, the bass - has on the overall sound. Using a graphic equalizer, you generally have a slider at certain frequencies that you move up and down to change the amplitude of those frequencies. Using the DFT information, we can do the same thing. To eliminate all sounds below 880 Hz, for example, we can take the DFT, use logical masks to manipulate the amplitudes of certain frequencies, then take the inverse DFT (given by the `ifft` command) and play the result. To make things a bit easier, we will also use the `fftshift` function which moves DC to the center of the frequency spectrum. Furthermore, we will use the `real` command on the resulting sound since the `fft` and `ifft` functions generally produce small round-off errors that allow complex parts to creep into the final signal that should not be there (and that the `sound` command doesn't know how to handle.

Open the `MaskMusic.m` code and run it with mask 0.1 while choosing to hear the original sound. The command for this in MATLAB is:

```
MaskMusic(0.1, 1)
```

Use the `MySound` file that you just created when asked. Having done that, you can hear that the bass is reduced and the sound is "hollow" somehow. To hear the part that was eliminated all by itself, run the code using mask 0.2:

```
MaskMusic(0.2, 1)
```

This time, only signal components below 880 Hz are sent to the speakers. You can probably recognize the sound, but it is a bit like listening to it through a wall. Because you sampled at a high enough frequency, there is no aliasing - just a complete loss of the higher frequencies that provide a crispness to the sound.

To prove that this really is doing what we claim, re-run those two masks on the `MyChirp` sound file. You will note that only certain sections of the chirp are audible (with the exception of some tones produces primarily through roundoff).

To hear a single octave, you can create a mask that only allows a band of frequencies through. Run the code using mask 0.3 to hear the octave starting at middle A and ending one octave above middle A. Finally, you may want to hear a very narrow range of frequencies - for example, those between middle A and middle C. Run the code using mask 0.4 to hear those sounds.

Next, you will calculate some other masks and play your sound after having applied them. You should put these in the `if` tree in the `MaskMusic` function. You will be showing your completed functions as well as the figures to your lab TA. You should also be discussing how each mask changes the sound - be sure to take notes while running them so you can describe them to the TA if asked. Each member of the team should listen to the filtered sound each time. You will likely choose to not listen to the original recording after the first few masks - simply put in a 0 for the second argument of the function to keep it from repeating the original sound.

4.11 Your Turn

Pick	Mask Description
1	Hear only sounds below 440 Hz
2	Hear only sounds above 1760 Hz
3	Hear only sounds between 440 Hz and 1760 Hz
4	Hear only sounds either below 440 Hz or above 1760 Hz
5	Hear only sounds below 4000 Hz (telephone quality)
6	Hear only sounds below 5000 Hz (AM quality)
7	Hear only sounds above 4000 Hz (sounds missing from telephone)
8	Hear only sounds above 5000 Hz (sounds missing from AM)

Note that case 6 will sound different from sampling the signal at 10 kHz. This is because, for this program, you have sampled at a high enough frequency that none of the information is aliased, then you have applied a filter to cut out the frequency information beyond 5 kHz. This is what a radio station must due before sending the audio signals. If they were to simply sample the audio at 10 kHz, they would get aliasing and most likely a lower-quality sound.

While the above only pass through particular frequency bands, you could also choose to *accentuate* certain sounds while still playing the rest of the frequencies at their original magnitudes. This more closely resembles what happens with a graphic equalizer - you do not completely eliminate a range of frequencies but rather accentuate the ones you like. For example, run the function using mask 0.5 to accentuate sounds between two and one octaves below middle A. Notice how the mask is built - it starts off with a 1 since you want *every* frequency to come through, then 3 more is added for a particular range of frequencies (in this case, frequencies between 110 and 220 Hz). In other words, you hear everything, but you have quadrupled sounds between 110 and 220 Hz (the original one plus three more). Now you should create the following masks as well. Note they can be easily built by looking at your first four masks as well as mask 0.5.

Pick	Mask Description
9	Hear everything, but quadruple sounds below 440 Hz
10	Hear everything, but quadruple sounds above 1760 Hz
11	Hear everything, but quadruple sounds between 440 Hz and 1760 Hz
12	Hear everything, but quadruple sounds either below 440 Hz or above 1760 Hz
13	Your first choice:
14	Your second choice:

These masks should have an amplitude of 1 except in the specified frequencies, where the amplitude should be 4. For this lab, you will also need to come up with your own masks and mask descriptions. Put these in Picks 13 and 14. You should think about what you want to do with ranges of frequencies. Note that your masks should never have a magnitude above 4 since the maximum voltage of your signal is 250 mV and the maximum value used by the `sound` function is 1.

4.12 Checkout and Clean-Up

This lab is to be completed before the end of the laboratory period. You will need to get checked off for the required masks as well as the two you choose to create. You can have the TA check off several masks at once by running your program and selecting the masks you want credit for. Before leaving the lab, please carefully disconnect both wires from the CB-68LP, take the wires and return them to the front, then disconnect your headphones but leave the rest of the audio wiring intact. On the computer, remove all files from the MATLAB directory, but only after your lab group is sure that you have been checked off for the masks. Once you have finished all this, you can have a TA come over, verify that the MATLAB directory is clear and that your circuit has been taken apart, and collect your worksheet.

4.13 Overall Network Listing

Item	First	Second	Note
AudioRedClip	Splitter	RED1	Clip red lead to red wire
RED1	AudioRedClip	Line 33	ACH1
AudioBlackClip	Splitter	BLK1	Clip black lead to black wire
BLK1	AudioBlackClip	Line 66	ACH9

Table 4.1: Highly Simplified Network Listing for Synchronous I/O

4.14 Scripts

4.14.1 `AliasDemo.m` Code

```
% AliasDemo.m
% Written by Michael R. Gustafson II (mrg@duke.edu)
% Visual demonstration of aliasing

SR1 = 500; SR2 = 100; SR3 = 20; SR4 = 15;
SR5 =  11; SR6 =   7; SR7 =  2;

t1 = linspace(0, 1, SR1+1); t2 = linspace(0, 1, SR2+1); t3 = linspace(0, 1, SR3+1);
t4 = linspace(0, 1, SR4+1); t5 = linspace(0, 1, SR5+1); t6 = linspace(0, 1, SR6+1);
t7 = linspace(0, 1, SR7+1);

f = @(t) sin(10*2*pi*t);

figure(1);
subplot(6,2,1)
plot(t1, f(t1), 'k-', t2, f(t2), 'ro')
subplot(6,2,2)
plot(t2, f(t2), 'ro-', t1, f(t1), 'b--')
title('fs=100 Hz');

subplot(6,2,3)
plot(t1, f(t1), 'k-', t3, f(t3), 'ro')
subplot(6,2,4)
plot(t3, f(t3), 'ro-', t1, f(t1), 'b--')
title('fs=20 Hz');

subplot(6,2,5)
plot(t1, f(t1), 'k-', t4, f(t4), 'ro')
subplot(6,2,6)
plot(t4, f(t4), 'ro-', t1, f(-t1/2), 'b--')
title('fs=15 Hz');

subplot(6,2,7)
plot(t1, f(t1), 'k-', t5, f(t5), 'ro')
subplot(6,2,8)
plot(t5, f(t5), 'ro-', t1, f(-t1/10), 'b--')
title('fs=11 Hz');

subplot(6,2,9)
plot(t1, f(t1), 'k-', t6, f(t6), 'ro')
subplot(6,2,10)
plot(t6, f(t6), 'ro-', t1, f(t1*3/10), 'b--')
title('fs=7 Hz');

subplot(6,2,11)
plot(t1, f(t1), 'k-', t7, f(t7), 'ro')
subplot(6,2,12)
plot(t7, f(t7), 'ro-', t1, f(t1*0/10), 'b--')
title('fs=2 Hz');
```

4.14.2 `freq1.m` Code

```
function freq1(fs, a, fa, b, fb, c, fc)
% freq1.m
% Written by Michael R. Gustafson II (mrg@duke.edu)
% Visual demonstration of aliasing
% fs: sampling rate
% a, fa: amplitude and frequency of first cosine
% b, fb: amplitude and frequency of second cosine (optional)
% c, fc: amplitude and frequency of third cosine (optional)


%% Set up time and frequency bases
tmax=2;
N=fs*tmax;
t=linspace(0,tmax-1/fs, N)';
f=linspace(0, (N-1)*fs/N, N)';
f=f-f(fftshift(f)==0);

%% Open or make figure 1 active
figure(1)

%% Validate the number of inputs - must be 3, 5, or 7
if nargin<3
    error('Must have at least three arguments\n')
elseif mod(nargin,2)==0
    fprintf('Must have complete amplitude and frequency pairs\n');
end

%% Build the signal based on the number of arguments
x = a*cos(2*pi*fa*t);
if nargin>3
    x = x + b*cos(2*pi*fb*t);
end

if nargin>5
    x = x + c*cos(2*pi*fc*t);
end

%% Calculate the FFT and plot both the signal and
%% its frequency information
X=fft(x);
subplot(2,1,1)
plot(t,x)
xlabel('t, sec')
ylabel('x(t)')
title('Time Representation');
subplot(2,1,2)
X=fft(x);
bar(f,fftshift(abs(X)));
xlabel('f, Hz')
ylabel('|X(f)|')
title('Frequency Representation');
```

4.14.3 `freq2.m` Code

```
function freq2(fs, a, fa, b, fb, c, fc)
% freq2.m
% Written by Michael R. Gustafson II (mrg@duke.edu)
% Visual demonstration of aliasing
% fs: sampling rate
% a, fa: amplitude and frequency of first cosine
% b, fb: amplitude and frequency of second cosine (optional)
% c, fc: amplitude and frequency of third cosine (optional)
% Will also play signal
% NOTE - sounds are always rescaled for maximum volume
% freq2(2000, 1, 400, -1, 1600) for example

%% Set up time and frequency bases
tmax=2;
N=fs*tmax;
t=linspace(0,tmax-1/fs, N)';
f=linspace(0, (N-1)*fs/N, N)';
f=f-f(fftshift(f)==0);

%% Open or make figure 1 active
figure(1)

%% Validate the number of inputs - must be 3, 5, or 7
if nargin<3
    error('Must have at least three arguments\n')
elseif mod(nargin,2)==0
    fprintf('Must have complete amplitude and frequency pairs\n');
end

%% Build the signal based on the number of arguments
x = a*cos(2*pi*fa*t);
if nargin>3
    x = x + b*cos(2*pi*fb*t);
end

if nargin>5
    x = x + c*cos(2*pi*fc*t);
end

%% Calculate the FFT and plot both the signal and
%% its frequency information
X=fft(x);
subplot(2,1,1)
plot(t,x)
xlabel('t, sec'); ylabel('x(t)'); title('Time Representation');
subplot(2,1,2)
X=fft(x);
bar(f,fftshift(abs(X)));
xlabel('f, Hz'); ylabel('|X(f)|'); title('Frequency Representation');

pause
sound(x./max(abs(x)), fs)
```

4.14.4 `EqualizerDone.m` Code

```
%% Initialize variables and objects
clear; format short e; figure(1); clf;
delete(daqfind);

%% Prepare Input
% Create handle to input
AI=analoginput('nidaq',1);

% Add channel 1 to input
addchannel(AI, 1);

%% Prepare DAQ Card
% set variables for sample rate and duration
samplerate=50000;
duration=10;

% calculate number of samples
samples=duration*samplerate;

% set sample rate, trigger type, and samples per trigger
set(AI, 'SampleRate', samplerate);
set(AI, 'TriggerType', 'Manual');
set(AI, 'SamplesPerTrigger', samples);

% set input, sensor, and units range
set(AI.Channel(1), 'InputRange', [-0.5, 0.5]);
set(AI.Channel(1), 'SensorRange', [-0.5, 0.5]);
set(AI.Channel(1), 'UnitsRange', [-0.5, 0.5]);

%% Use DAQ Card
% examine AI, start AI, and trigger AI
AI
start(AI);
fprintf('Press return to take data\n');
pause
trigger(AI);
fprintf('Taking data...\n')

% take data from AI
[data time] = getdata(AI);

%% Plot and Play Data
plot(time, data)
fprintf('Press return to play sound\n');
pause
sound(data, samplerate)
fprintf('Playing sound...\n');

%% Save Data
FileName = input('Enter filename for sound data: ', 's');
FileSave = sprintf('save %s data time samplerate', FileName);
eval(FileSave)
```

```
53
54  %% Calculate and plot frequency information
55  tmax = duration;
56  fs = samplerate;
57  N = fs*tmax;
58  t = linspace(0, tmax-1/fs, N)';
59  f = linspace(0, (N-1)*fs/N, N)';
60  f = f - f(fftshift(f)==0);
61
62  figure(1)
63
64  Incr = floor(samplerate/5000);
65
66  x=data;
67  X = fft(x);
68  subplot(2,1,1)
69  plot(t(1:Incr:end),x(1:Incr:end))
70  subplot(2,1,2)
71  X = fft(x);
72  bar(f(1:Incr:end), fftshift(abs(X(1:Incr:end))));
```

4.14.5 `MaskMusic.m` Code

```
%% MaskMusic.m
%% Written by Michael R. Gustafson II (mrg)
%% Masks for Picks 1-10 written by:
%%%
%%%

function Mask(Pick, PlayOriginal)
%% Default case is not to play original
if nargin==1, PlayOriginal=0; end

%% Load data
FileName = input('Enter filename for sound data: ', 's');
FileLoad = sprintf('load %s', FileName);
eval(FileLoad)

%% Set up time and frequency bases depending on original song
fs = samplerate;
N = length(time)
f = linspace(0, (N-1)*fs/N, N)';
f = f - f(fftshift(f)==0);

%% Take DFT of song
x=data;
X = fft(x);

%% Determine mask to use
if Pick==0.1
    Mask = (abs(f)>=880);
elseif Pick==0.2
    Mask = (abs(f)<=880);
elseif Pick==0.3
    Mask = (abs(f)>=440) & (abs(f)<=880);
elseif Pick==0.4
    Mask = (abs(f)>=440) & (abs(f)<=440*2^(3/12));
elseif Pick==0.5
    Mask = 1 + 3*((abs(f)>=110) & (abs(f)<=220));
elseif Pick==1
elseif Pick==2
elseif Pick==3
elseif Pick==4
elseif Pick==5
elseif Pick==6
elseif Pick==7
elseif Pick==8
elseif Pick==9
elseif Pick==10
elseif Pick==11
elseif Pick==12
elseif Pick==13
elseif Pick==14
end

```

```
53  %% Use mask to remove corresponding components in X
54  %% and create a shifted FFT called Y
55  Yshift = fftshift(X).*Mask;
56
57  %% Shift Y back, then convert it to time space
58  Y = fftshift(Yshift);
59  y = real(ifft(Y));
60
61  %% Plot the original signal and its DFT,
62  %% the filtered signal and its DFT, and the mask
63  Incr = floor(samplerate/500);
64
65  figure(1)
66  subplot(3,2,1)
67  plot(time(1:Incr:end), real(x(1:Incr:end)));
68  axis([0 max(time) -1 1])
69  title('Original Sound')
70  xlabel('t, sec'); ylabel('x(t)')
71  subplot(3,2,3);
72  bar(f(1:Incr:end), fftshift(abs(X(1:Incr:end))));
73  title('DFT of Original Sound')
74  xlabel('f, Hz'); ylabel('|X(f)|')
75
76  subplot(3, 2, 2)
77  plot(time(1:Incr:end), real(y(1:Incr:end)))
78  axis([0 max(time) -1 1])
79  title(sprintf('Sound After Mask %0.0f', Pick))
80  xlabel('t, sec')
81  ylabel('x(t)')
82  subplot(3, 2, 4)
83  bar(f(1:Incr:end), fftshift(abs(Y(1:Incr:end))));
84  title(sprintf('DFT of Sound After Mask %0.0f', Pick))
85  xlabel('f, Hz'); ylabel('|X(f)|')
86
87  subplot(3, 1, 3);
88  bar(f(1:Incr:end), Mask(1:Incr:end));
89  title(sprintf('Mask %0.0f', Pick))
90  xlabel('f, Hz'); ylabel('|M(f)|')
91
92  %% Play the original sound after hitting return
93  %% only if requested
94  if PlayOriginal
95      fprintf('Hit return to play original.\n');
96      pause;
97      sound(x, fs)
98  end
99
100 %% Play new sound after hitting return
101 fprintf('Hit return to play filtered sound.\n');
102 pause;
103 sound(real(y), fs)
```