Lab 8

Lab 8 introduces basic concepts of filtering using the Discrete Fourier Transform (DFT). In particular we will be making a feedback suppression system.

Filtering Using the DFT

Suppose that we have an unwanted sinusoidal component in our sound, such as microphone feedback happening at 4000 Hz. How might we design a system to automatically remove it?

1. Take the DFT of the signal, and separate the magnitude and phase components
2. Find the bin number that has the greatest magnitude*
3. Select a range on either side of that bin according to a user defined parameter
4. Set the magnitude of those bins to 0
5. Reconstruct the complex spectrum using the phasor form $Y=R*e^{j\theta}$, where R is the magnitude vector and θ is the phase vector
6. Take the inverse DFT to convert the signal back to the time domain

*** This will most likely be our offending signal, since continuous sinusoids concentrate a lot of energy into just one or two bins.**

Indexing Arrays with Arrrays

In this lab we will harness some of the real power of Matlab to get things done quickly with a minimum amount of code. Observe the following example:

```
a = [10, 20, 30, 40, 50, 60, 70, 80, 90];
aIndex = [1, 2, 3, 4];
b = a(aIndex);
disp(b);
     10    20    30    40
```

What just happened here? We used the vector aIndex to index data points from the vector a. When I gave the command `b = a(aIndex)` I was essentially saying "fill b with the elements of a that are listed in the vector aIndex." This makes a very powerful tool for selecting portions of arrays. A command that is very useful for this task is the "`find`" command. "`find`" returns an array of indices for which a given condition is true. For instance, in the above example, I could have used `aIndex = find(a < 45)` and would have gotten the same result. You will need this functionality for completing the assignment.

Assignment

Start by downloading **getDirty.m**, and put it in the same folder as your code for this lab. This function takes in a **mono** input as a **row vector** and outputs another row vector which consists of the input plus an added sinusoid. This sinusoid will be in the range of 3 – 5 kHz, and will be different each time the program runs.

1. Apply the getDirty function to a .wav file of your choosing (you might use the song

clip you used in Lab 7). Save this new "dirty" signal to a .wav file. **Don't forget to normalize it before saving it to .wav so that you don't clip the signal.**

2. Implement the algorithm described above in a Matlab script (I recommend a script as opposed to a function, as it will be easier to debug).
3. In your script there **must** be a variable called **"`userWidth`"** that allows for variable width of how many bins are set to 0. It is up to you to determine what range of values this variable can have, and how it is implemented. **Be sure to include comments/description describing to the user (me) how to use it.**
4. Listen to the cleaned signal. Save this new filtered signal to a wav file.
5. Is the cleaned signal exactly the same as the input? If not, how is the sound different? Were you able to completely remove the unwanted noise?
6. **In your submission be sure to include the .m file, the original .wav file, the dirty version, and the clean version.**

Useful Commands:

`fft()`	Take the DFT of a signal
`ifft()`	Take the Inverse DFT of a signal
`abs()`	Take the magnitude of a complex number (or the absolute value, for real numbers)
`angle()`	Take the angle of a complex number
`max()`	Find the maximum value of an array
`real()`	Take only the real part of a complex number
`find()`	Find the indices of an array for which a given condition is true
`round()`	Round to the nearest integer

Tips:

- If we have a vector of magnitudes `mag` and a vector of phases called `phase`, we can convert them into complex form with `myVar = mag.*exp(1i*phase);`
- Remember that the DFT has two components for every frequency. Thus, when you find the bin with the maximum magnitude, you will actually be finding **two** bins (one for the positive frequencies, and one for the negative frequencies. You will need to perform "mirror image" modifications at these locations in order for your results to turn out right.
- Remember that indices must be positive integers. Matlab will get unhappy if you try to do `a(2.5)` or `a(-7)`. The round() function may help you here.

After doing the inverse DFT your results **should** be real (i.e. not complex). It's possible, though, that they will still be in complex form. **If** you're confident that all of your preceding code is correct, simply use the `real()` command to keep only the real part. Note that if you do not handle the DFT bins correctly then the output **will** be complex, so **only** use the real() function once your confident that everything else is right.

Lab 8 – Filtering using the FFT

Nate Paternoster

```
%---- Lab 8 - Filtering using the DFT ----%


[y,fs,NBITS] = wavread('wavefile.wav',[10000,500000]);
y = y(:,1);                                 % Converts the signal to mono
y = y';                                     % Creates a ROW vector
y = getDirty(y,fs);
y = y/max(abs(y));                          % Normalizes the signal

wavwrite(y,fs,NBITS,'Lab 8 dirty signal.wav');
sound(y,fs);                                % The dirty signal

z = cleanup(y,100);
sound(z,fs);                                % The clean signal
wavwrite(z,fs,NBITS,'Lab 8 clean signal.wav');



%---- My cleanup function ----%


function [ output ] = cleanup(x, userwidth)

fftLen = 2^nextpow2(length(x));
y = fft(x,fftLen);
dft_Mag = abs(y);                       % The magnitude of the dft
dft_Pha = angle(y);                     % The phase of the dft

maxvalue = max(dft_Mag);                % Finds the max value
maxbin = find(dft_Mag==maxvalue);       % Finds the 2 indices (bins) of
                                        that value
halfwidth = round(userwidth/2);

y1index = maxbin(1)-halfwidth:maxbin(1)+halfwidth;
y2index = maxbin(2)-halfwidth:maxbin(2)+halfwidth;
      % Setting the two vectors according to the user width

dft_Mag(y1index) = zeros(size(y1index)); % Those values in the two bin
dft_Mag(y2index) = zeros(size(y2index)); % vectors are set to 0

z = dft_Mag.*exp(1i*dft_Pha);    % Reconstruction of the new magnitude
                                 % vector with its phase to get a
                                 % complex vector
o = ifft(z);

output = real(o);
end
```

7. *Apply the getDirty function to a .wav file of your choosing (you might use the song clip you used in Lab 7). Save this new "dirty" signal to a .wav file.* ***Don't forget to normalize it before saving it to .wav so that you don't clip the signal.***

   The *getDirty* function will take a .wav file and insert a sinusoid varying between 3k and 5kHz. This will mimic the affect of an unwanted frequency in a recording that we want to remove. My function, called *cleanup*, will use the DFT to find the offending frequency and remove it from the signal.

8. *Implement the algorithm described above in a Matlab script (I recommend a script as opposed to a function, as it will be easier to debug).*

   This algorithm is:

   - Take the DFT of the signal, and separate the magnitude and phase components
   - Find the bin number that has the greatest magnitude*
   - Select a range on either side of that bin according to a user defined parameter
   - Set the magnitude of those bins to 0
   - Reconstruct the complex spectrum using the phasor form $Y=R*e^{j\theta}$, where R is the magnitude vector and θ is the phase vector
   - Take the inverse DFT to convert the signal back to the time domain

9. *In your script there* ***must*** *be a variable called* ***"userWidth"*** *that allows for variable width of how many bins are set to 0. It is up to you to determine what range of values this variable can have, and how it is implemented.* ***Be sure to include comments/description describing to the user (me) how to use it.***

The cleanup function takes the input signal and a userwidth. The userwidth will set how many of the adjacent bins to the offending frequency will also be set to zero. We can think of the userwidth to be the Q of a filter where the offending frequency is the center frequency to be attenuated and the Q determines how many neighboring frequencies will also be attenuated.

```
fftLen = 2^nextpow2(length(x));
y = fft(x,fftLen);
dft_Mag = abs(y);
dft_Pha = angle(y);
```

This part of the code will find the most efficient DFT of the input waveform and separate it into its magnitude and phase vector components.

```
maxvalue = max(dft_Mag);
maxbin = find(dft_Mag==maxvalue);
```

This will search the magnitude vector of the DFT and find the maximum value reached. This will be our offensive frequency. It will then find the two bins that are the locations of that frequency (the offensive frequency bin and its complex conjugate).

```
halfwidth = round(userwidth/2);
y1index = maxbin(1)-halfwidth:maxbin(1)+halfwidth;
y2index = maxbin(2)-halfwidth:maxbin(2)+halfwidth;
dft_Mag(y1index) = zeros(size(y1index));
dft_Mag(y2index) = zeros(size(y2index));
```

This will define the two vectors of bins we want to remove centered around the two locations of the offensive frequency. The width is defined by the user. The values of the magnitude vector at these positions are set to zero.

```
z = dft_Mag.*exp(1i*dft_Pha);
o = ifft(z);
output = real(o);
```

The new magnitude vector is recombined with the untouched phase vector to put the DFT signal back in its complex form. The signal then undergoes the inverse DFT to transform back into the time domain. Finally, we take only the real values of the signal to playback.

10. Listen to the cleaned signal. Save this new filtered signal to a wav file.

11. Is the cleaned signal exactly the same as the input? If not, how is the sound different? Were you able to completely remove the unwanted noise?

The cleaned signal is different depending on the size of the userwidth. With a low userwidth the noticeable difference is that the offending frequency has been removed. With a much larger userwidth, like 30000 or 50000 bins, the output begins to sound like it has been put through a HPF. This is because so much of the mid and high frequencies are being attenuated to zero. A userwidth of 100 was a satisfactory amount to completely remove the unwanted noise.