FINAL PROJECT
MATLAB Implementation of an N-Channel Vocoder

BEE 235: Lab Session AA (Spring 2020)

Austin Gilbert – 1737454

June 5, 2020

Table of Contents

Table of Figures

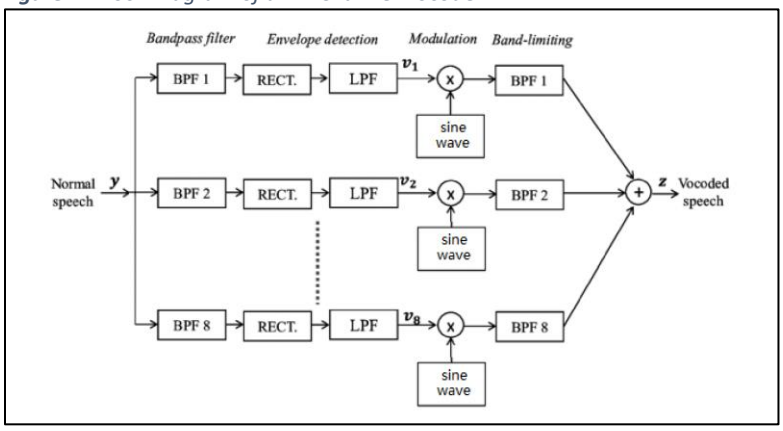
Abstract

With the objective of implementing an n-channel vocoder as a functional program in MATLAB, this final class project integrated a myriad of signal analysis and filtering techniques from labs two and four to briefly explain how to build such a function. Although proven implementations and functions were given to guide this build, much of the project was left for the student to decipher. Additionally, the student was allowed a substantial degree of creative freedom in completing optional portions of the project and overall implementation.

Introduction

First engineered in the early 1940s, the channel vocoder was the first machine to successfully analyze, modulate, and synthesize human speech. Used in modern music, media, and cochlear implants, the vocoder combines a carrier sound source and modulator sound source into a set of specified frequencies. In order to do this, the channel vocoder is implemented as a "bank of filters" that extracts the envelope of the modulator source and modulates it with the carrier source; after completing this modulation for each band, all bands are summed to create a final output signal. This implementation is summarized by the given block diagram in Figure 1.

***Figure 1**: Block Diagram of an N-Channel Vocoder*



** RECT – rectification (abs(signal)); LPF – low-pass filter; BPF – band-pass filter*
*** "band" is a synonym for "channel"*

Program Design Approach

The main approach to designing this N-channel vocoder was taken directly from the block diagram in Figure 1 and the subsequent description. To summarize the diagram, a wav file containing human speech will be band-pass-filtered, rectified, and low-pass-filtered in N separate, logarithmically spaced bands to form N separate envelopes. Each envelope will then be modulated (multiplied after a Fourier Transform) with a given sine wave before summing all N modulated signals into one signal representing our output vocoded speech.

Intending to implement this design into a single top-level program, I began by creating input parameters that could be changed and fed into this function from a separate file. After naming my function `vocoder.m`, I began implementing the following input parameters:

- `inputfile`—the wav file to modulate with the carrier signal
- `outputfile`—the wav file to use for output

- `bandnum`—the number of bands/channels to use in the vocoder
- `soundon`—the Boolean to play sound
- `graphon`—the Boolean to create figures
- `noiseon`—the Boolean to replace the carrier signal with white noise
    - This noise apparently serves to simulate the sounds of a cochlear implant

I then created a simple user interface by designing a progress bar and description which both display to the user what the program is doing while it is running.

***Figure 2**: Initial MATLAB Code*

```
% Austin Gilbert
% B EE 235 Spring 2020
% Final Project: N-Channel Vocoder
% vocoder.m

function y = vocoder(inputfile, outputfile, bandnum, soundon, graphon, noiseon)
% This N-Channel Vocoder modulates a signal into Vocoded Speech
%   inputfile - the signal to modulate with the carrier signal
%   outputfile - file name to use for output
%   bandnum - number of bands/channels
%   soundon - the boolean to play sound
%   graphon - the boolean to create figures
%   noiseon - the boolean to replace carrier signal with white noise

% Description
disp(['Modulating ',inputfile,' to ',outputfile,' through a ',num2str(bandnum),'-Band Vocoder.']);
f = waitbar(0, 'Starting');
```

After setting up the initial code of the equation, I copied the boundary frequency generation function and edited it slightly to fit within my program. Figure 3 shows this snippet of code which serves to logarithmically space each band given a `numband` value of N.

***Figure 3**: Boundary Frequency Generation MATLAB Code*

```
% Boundary Frequency Generation for N Bands
xmin = log10(300/165.4+1)/2.1;                  % min log frequency
xmax = log10(6000/165.4+1)/2.1;                 % max log frequency
x = xmin:(xmax-xmin)/bandnum:xmax;
fco = zeros(1,bandnum+1);                       % skeleton of fco vector
for i = 1:bandnum+1
    fco(i) = 165.4*(10^(x(i)*2.1)-1);           % create fco vector
end
```

Following the generation of all N frequency bands, I began dealing directly with the input or modulator signal. After reading the signal and preparing a couple of vectors, I began a for-loop that iterates over each band.

Using the `butter` and `filter` commands, I then applied a band-pass filter to the input modulator signal for each band (Figure 4). I also began the process of graphing a few plots using the `graphon` Boolean to only graph when the user specifically requests plots. Throughout the next few sections of my program, I have labeled locations in my code where I used the `plot` command with a comment that has the word “GRAPH” in all caps.

***Figure 4**: Bandpass Filter MATLAB Code*

```
% Bandpass Filter
[signal, Fs] = audioread(inputfile);            % read inputfile
t = (0:length(signal)-1)/Fs;                    % time vector creation
y = zeros(length(t), 1);
carrier = rand(size(t));                        % noise creation
waitbar(.33, f, 'Modulating');
for n = 1:length(fco)-1
    [bb, aa] = butter(3,[fco(n), fco(n+1)]/(Fs/2));
    if graphon
        figure(1)                               % GRAPH - Freq Plot
        [H, F] = freqz(bb, aa, 256, Fs);
        plot(F, abs(H), 'DisplayName', ['Band ', num2str(n)]);
        xlabel('Frequency (Hz)');
        ylabel('|H|');
        legend;
        title('Frequency Response (Bandpass)');
        hold on;
    end
    bpsignal = filter(bb, aa, signal);          % bandpass filter
    if graphon
        figure(2)                               % GRAPH - Bandpassed Signal
        plot(t, bpsignal, 'DisplayName', ['Band ', num2str(n)]);
        xlabel('Time (s)');
        ylabel('Amplitude (normalized)');
        legend
        title('Bandpassed Signals');
        hold on;
    end
```

Further in the for-loop, I used the `abs()` command to perform a full-wave rectification on the filtered signal. I then used the `butter` and `filter` commands again to put this rectified signal through a low-pass filter, extracting the envelope of the original modulator signal.

***Figure 5**: Envelope Rectification MATLAB Code*

```
% Envelope Rectification
    rectsignal = abs(bpsignal);                 % full-wave rectification
    [dd, cc] = butter(2, 400/(Fs/2));           % lowpass filter
    if graphon
        waitbar(.67, f, 'Graphing Intermediate Waveforms');
        figure(3)                               % GRAPH - Freq Plot
        [H, F] = freqz(dd, cc, 256, Fs);
        plot(F, abs(H), 'DisplayName', ['Band ', num2str(n)]);
        xlabel('Frequency (Hz)');
        ylabel('|H|');
        legend;
        title('Frequency Response (Lowpass)');
        hold on;
    end
    envelope = filter(dd, cc, rectsignal);
    if graphon
        figure(4)                               % GRAPH - Extracted Envelopes
        subplot(bandnum/2, 2, n)
        plot(t, envelope);
        xlabel('Time (s)');
        ylabel('Amplitude');
        title(['Envelope ', num2str(n)]);
        hold on;
    end
```

At the end of the for-loop, I created two carrier signals depending on if the Boolean `noiseon` was initially set to `true` or `false`: one sine wave signal and one from random noise. I then modulated both signals by multiplying them. For the noise carrier signal, I had to perform the extra step of putting it through a second bandpass filter before summing. Finally, all signals were summed together, normalized, and written to a final file.

***Figure 6**: Carrier Modulation & Summation MATLAB Code*

```
% Carrier Modulation / Summation
    if ~noiseon                                 % modulation with carrier
        carrier = sin(2*pi*round((fco(n) + fco(n+1))/2)*t);
        if graphon
            figure(5)                           % GRAPH - Modulated Signals
            subplot(bandnum/2, 2, n)
            plot(t, envelope .* carrier');
            xlabel('Time (s)');
            ylabel('Amplitude');
            title(['Mod Signal ', num2str(n)]);
            hold on;

        end
        y = y + (envelope .* carrier');         % summation & modulation
    else
        carriermod = envelope .* carrier';
        if graphon
            figure(5)                           % GRAPH - Modulated Signals
            subplot(bandnum/2, 2, n)
            plot(t, filter(bb, aa, carriermod));
            xlabel('Time (s)');
            ylabel('Amplitude');
            title(['Mod Signal ', num2str(n)]);
            hold on;
        end
        y = y + filter(bb, aa, carriermod);     % summation & bandpass
    end
end
y = y * (max(abs(signal))/max(abs(y)));
audiowrite(outputfile, y, Fs);
```

The final section of code following the consists mostly of code that plots different graphs as well as sound to play at the end of the function. Figure 7 contains the code I used to play and display the original sound in comparison with the modulated sound.

***Figure 7**: Final MATLAB Code*

```
if soundon
    linelength = fprintf('The Original Sound'); % SOUND - Original Sound
    sound(signal, Fs)
    pause(2)
    fprintf(repmat('\b',1,linelength))

    linelength = fprintf('The Modulated Sound');% SOUND - Modulated Sound
    sound(y, Fs)
    pause(2)
    fprintf(repmat('\b',1,linelength))
end
end
```

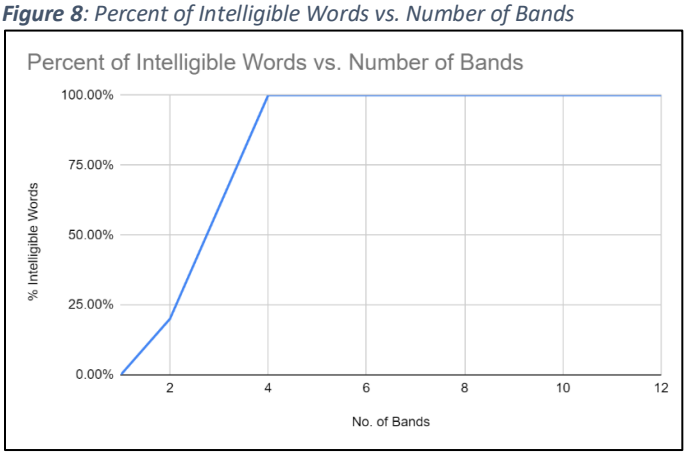
Program Results & Analysis

After creating and designing this program, I began to test its functionality. Testing different sounds under default parameters, I found that my program outputted a robotic version of the input modulator sound that was recognizable as distorted, yet human, speech. Finding no bugs or obvious problems with my code, I quickly moved on to testing multiple sounds and variables. One result that immediately captured my attention was the difference in sound when the `noiseon` parameter was turned from `false` to `true`. Sounding far more static and almost creepier than the previous robotic-sounding signal, I was able to simulate what it would sound like to have a cochlear implant. Sounds such as voices were heard very clearly, while music was very difficult to decipher.

Varying the Number of Bands

After varying the number of bands, I found that overall, the modulated sound became increasingly clearer with more frequency bands. From Figure 8, we need at least four bands to fully understand a vocoded sentence. I attributed this to the fact that speech contains many different sounds at multiple frequencies. Additionally, I discovered that if you input `bandnum` as 400 or greater, the voice also becomes unintelligible as the bands become so small that they create weird alien-sounding noises.

***Figure 8**: Percent of Intelligible Words vs. Number of Bands*



Intermediate Waveforms

The following waveforms are those calculated in each intermediate step of vocoder processing and represent each description of the block-diagram-inspired design.

***Figure 9**: Band-pass Frequency Response*

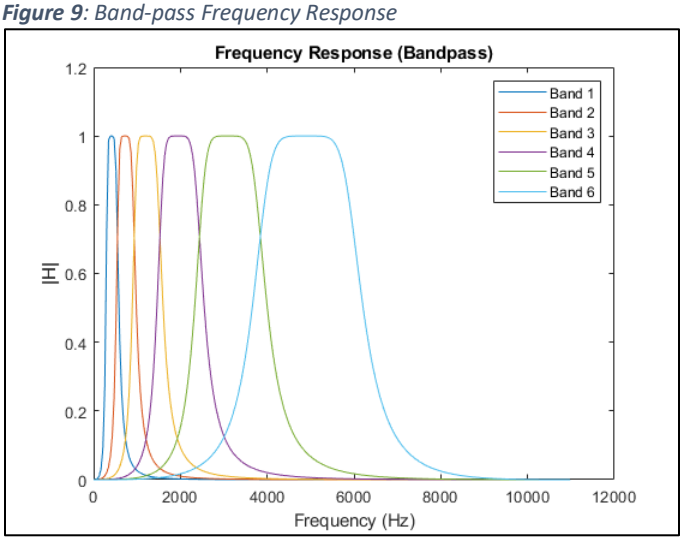


***Figure 10**: Low-pass Frequency Response*

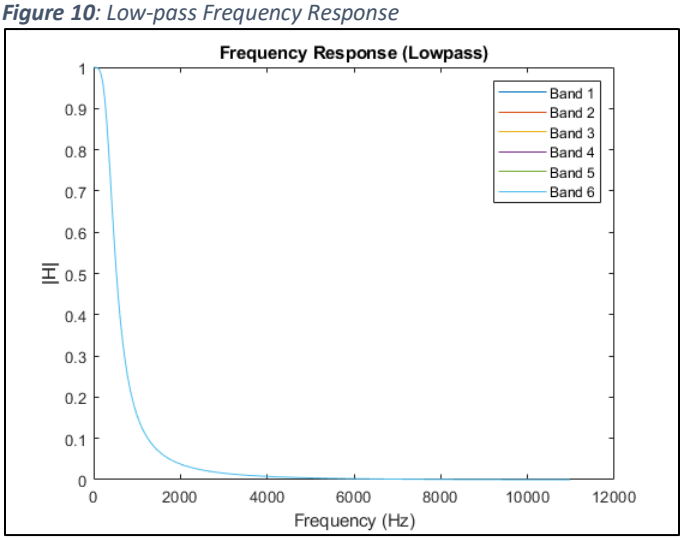


***Figure 11**: Band-passed Signals*

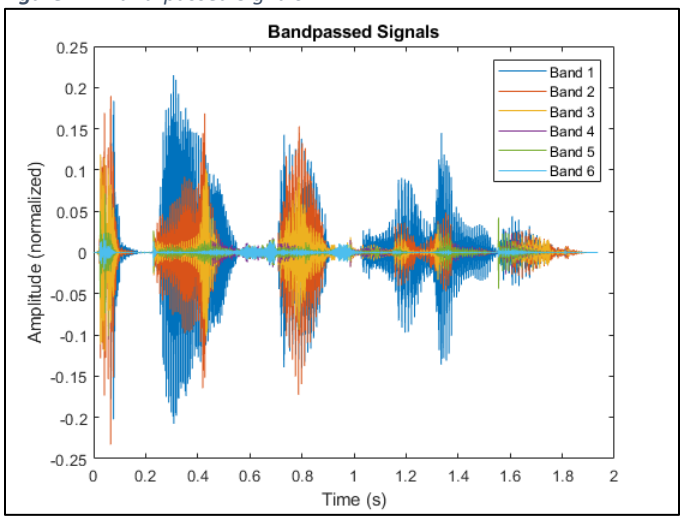
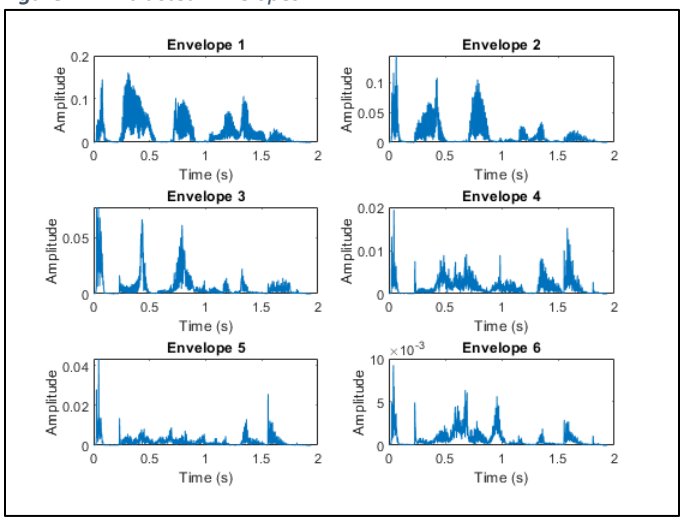


***Figure 12**: Extracted Envelopes*



Final Modulated Waveforms

The following waveforms are those calculated in the final step of vocoder processing and represent the comparison of the modulated and original signals.

***Figure 13**: Waveforms of Modulated Signals*

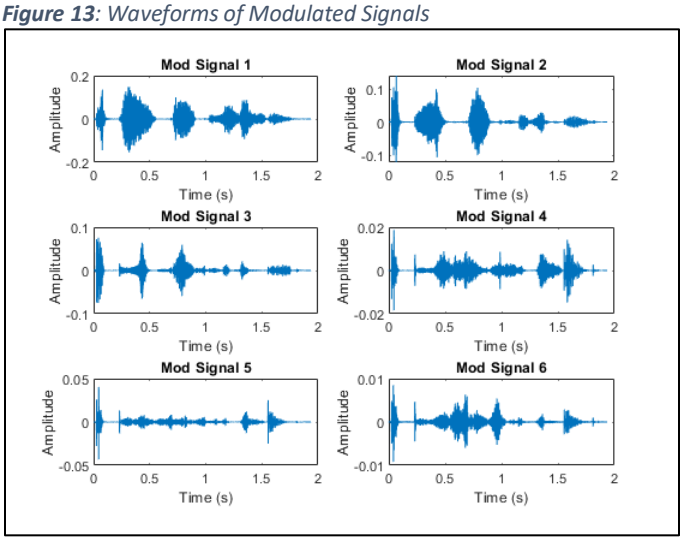


***Figure 14**: Modulated and Original Signal*

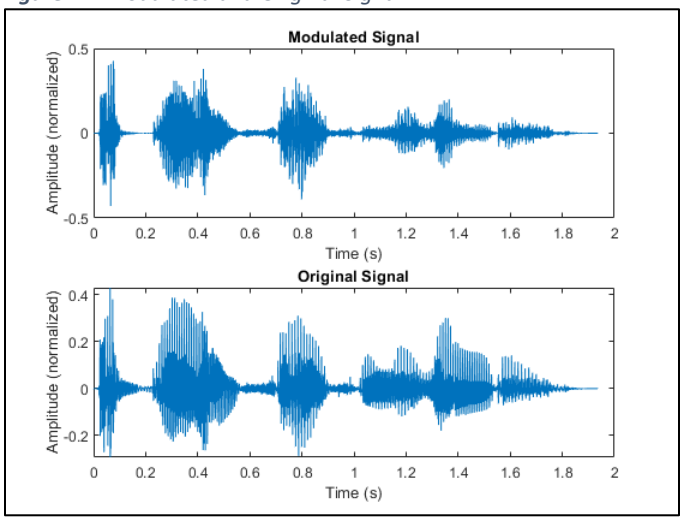
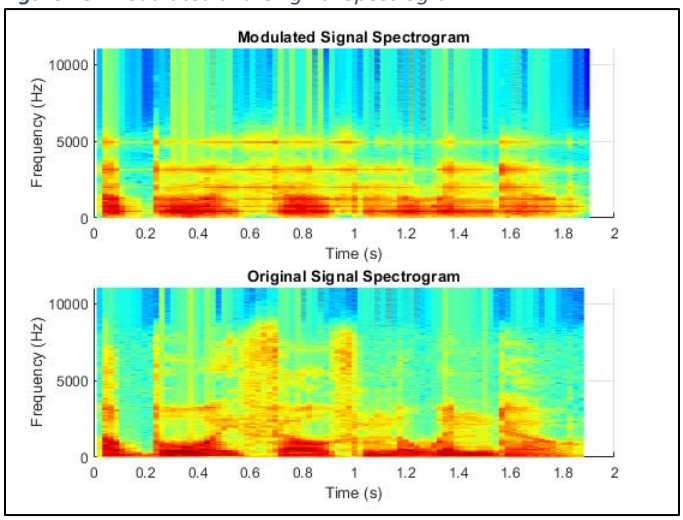


***Figure 15**: Modulated and Original Spectrogram*



Conclusions

Overall, this project served as a fun and interesting way to cap off everything we learned in the various labs throughout the quarter. Additionally, the element of creativity present in this project allowed me to design in whatever way I thought the code would best function. During the design phase of this project, I learned how to implement Booleans, progress bars, file creation, and modulation. Conceptually, I also learned how a vocoder functions and how the many filters are used to result in such a unique sound. Therefore, I feel the overall purpose of this project was achieved very well after checking my results with different values and learning the fundamentals of MATLAB.