Lab 4: Poles and Zeros - Vowel Synthesis

Version 1.3

Introduction

Filtering is something that occurs everywhere, even without the intervention of a human filter designer. At sunset, the light of the sun is filtered by the atmosphere, often yielding a spectacular array of colors. A concert hall filters the sound of an orchestra before it reaches your ear, coloring the sound and adding pleasing effects like reverberation. Even our own head, shoulders, and ears form a set of filters that allows us to localize sounds in space.

Quite often, we may wish to recreate these filtering effects so that we can study them or apply them in different situations. One way to do this is to model these "natural" filters using simple discrete-time filters. That is, if we can measure the response of a particular system, we would often like to design a filter that has the same (or a similar) response.

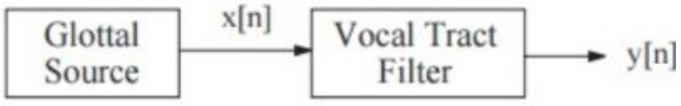
One of the goals for this laboratory is to introduce the use of discrete-time filters as models of real-world filters. In particular, we will examine how to apply a modeling approach to understanding vowel signals. Another goal of this lab is to present a method of filter design called pole-zero placement design. Working with this method of filter design is extremely useful for building an intuition of how the z-plane "works" with respect to the frequency domain that you are already familiar with. The design interface that we use for this task should help you develop a graphical understanding of how poles and zeros affect the frequency response of a system.

The objective of this lab is to use pole-zero placement design to create filters and to use them to synthesize vocal music!

Part 1

1.1 Modeling Vowel Production

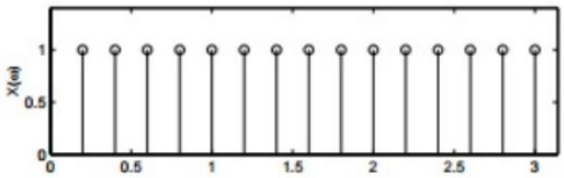
When we speak a vowel, the lungs push a stream of air through the larynx and the vocal cords. Given the appropriate muscular tension, this stream of air causes the vocal folds to vibrate, creating a nearly periodic fluctuation in air pressure passing through the larynx. The fundamental frequency of vocal fold vibration is typically around 100 Hz for males and 200 Hz for females. This fluctuating air stream then passes through the vocal tract, which is the airway leading from the larynx, through the mouth, to the lips. The positions of the tongue, lips, and jaw serve to shape the vocal tract, with different positions creating different vowel sounds. The different sounds are produced as the vocal tract shapes the spectrum of the pressure signal coming from the larynx. Depending upon the vocal tract configuration, different frequencies of the spectrum are emphasized, called formants. Speech production can be modeled using this source-filter model:



The first block is the glottal source, which produces a periodic signal (the glottal source signal) with a given fundamental frequency:

$$x(t) = \sum_{k=-\infty}^{\infty} A_k \cos(\omega_k t + \varphi_k)$$

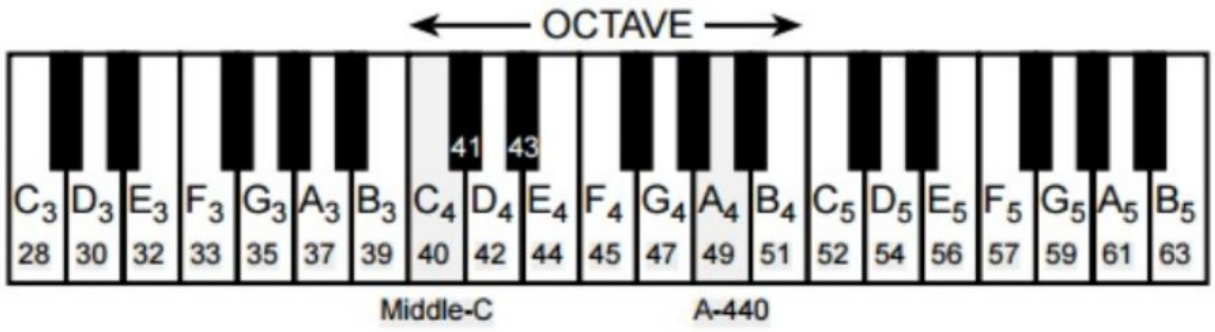
The glottal source signal, the signal formed by the air pressure fluctuations produced by the vibrating vocal cords, is typically modeled as a periodic pulse train. To do a first approximation, we can assume that the frequency spectrum of this pulse train is composed of equal amplitude harmonics:



In other words, all of the A_k are equal in the Fourier series of the glottal source signal. First, ***create a one second long glottal source signal, containing 7 non-zero harmonics (Fourier series k values −7 through 7), with fundamental frequency 150Hz. Use a sampling frequency of 8000 samples/s.*** **Note: this is not to be played for a TA, just for you to hear how the glottal signal sounds.**
Question: What is the minimum sampling frequency necessary to avoid aliasing?

1.2 A Function to Play a Vocal Note

You'll remember, from the Music Synthesis lab, our discussion that piano keyboards are laid out as illustrated by the following image:



- C_4 refers to C-key in the fourth octave. C_4 is also called middle-C.
- A_4 is also called A-440, because its frequency is 440 Hz.
- Every key is given a key number. The key number of middle-C is 40.
- The fundamental frequency of any piano key can be found by substituting its key number into the formula:

$$f_{piano} = 440*(2^{(keynum - 49)/12})$$

In this lab, we'll consider the human-voiced equivalent of a piano note to be half of its frequency, which corresponds to an octave (a difference of twelve notes) below it (this has to do with the way that we perceive human voice). So the human-voiced equivalent of the piano A-440 has a fundamental frequency of half of 440Hz, or 220Hz. Therefore, the fundamental frequency of any human-voiced note can be found by substituting the key number into the formula:

$$f_{piano} = 220*(2^{(keynum - 49)/12})$$

Write a function, named *glottal_key_to_note* (similar to your key_to_note function from Lab 1, **See the Appendix**) that takes in a key number, a duration, and a number of harmonics to produce a glottal source signal of given duration with fundamental frequency corresponding to the desired note. Use a sampling frequency of 8000 Hz (remember that a note with harmonics is a sum of tones with frequencies given by a fundamental frequency f times the harmonic number k). ***Question: If the keynum provided is 49, for 7 harmonics, what is the minimum sampling frequency required to prevent aliasing?***

For the following parts, use a Hann window to make the notes sound better (hanning() in MATLAB)

1.3 Synthesize a Song - Mary had a Bleating Lamb

As you learned in Lab 1, a song can be generated by successively adding notes. Use glottal_key_to_note to write a script that plays **the Mary song with note durations of 0.25 seconds.** Also create a script that plays the notes with seven harmonics (k=7). ***Generate both sounds and play it for a TA for checkoff.*** If you can't finish in time, submit a .wav file on Canvas.
Plot the frequency-time spectrogram of Mary for both one harmonic and seven harmonics and compare**.**

1.4 Scale Creation

See https://en.wikipedia.org/wiki/Scale_(music)
A musical scale is a set of notes ordered by their pitch (fundamental frequency).

1.4.1 Scale Matrix

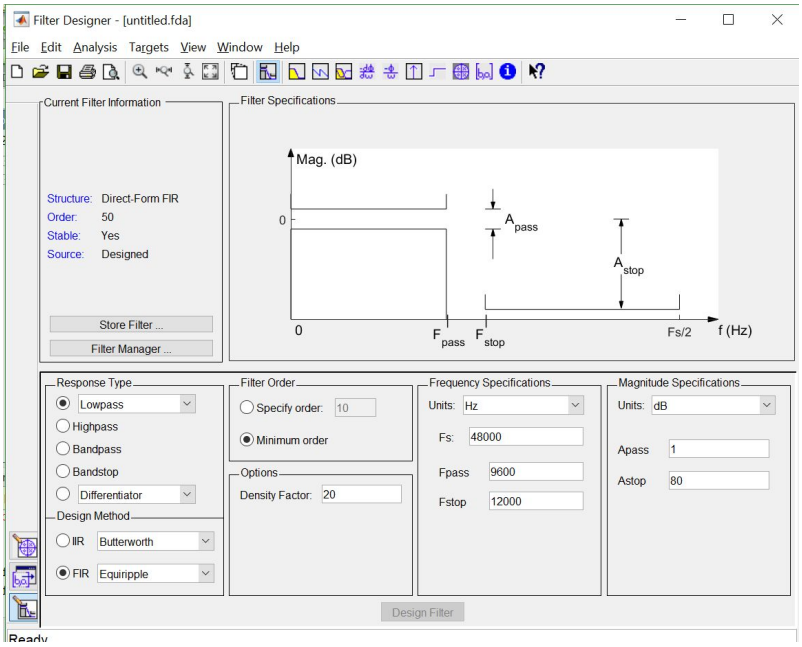
Use the provided code in the Appendix to **create a matrix of scale values**, where each row corresponds to an octave of a C Major scale.

1.4.2 Creating the Sounds

Using the scale matrix that you just made, **create a musical scale** using octaves 3 through 7 (keep in mind that row 1 is octave 1, row 2 is octave 2, etc.), where each note lasts for 0.25 seconds. **Make one scale with 1 harmonic, and another set with seven harmonics. Be sure to use fs = 8000 for one harmonic, and fs = 56000 for the seven harmonics. Play both of the scales for a TA.**

1.5 Intro to MATLAB Filter Designer

Matlab contains a GUI for simplifying the process of designing filters. To access this GUI, use the command filterDesigner in the command window. The GUI that appears should look like this:



The options that this lab will have you use are the response type, frequency specifications, and magnitude specifications. Once your filter is designed, use File -> Export to load your coefficients into MATLAB. If you design an IIR filter (see TAs/professors for explanation on why one filter may be preferable over another), the exported matrices are a Second-Order Section (SOS) matrix and a Gain matrix, which can be converted to b and a coefficients using [b,a] = sos2tf(SOS,G). If you use an FIR filter, Num is the output of the filter designer, with b = Num and a = 1. Then, filter your input signal with your filter using filter(b,a,xx) to create the output signal.

For all of the following filters, submit a screenshot of the magnitude response and the pole/zero plot (the latter can be accessed by clicking the image that looks like a pole/zero plot in the bottom left of the above photo).

1.5.1

Design a lowpass filter with the following specifications: fs = 8000, $f_{passband}$ = 1000Hz, $f_{stopband}$ = 1200Hz, do not change the default gain values . Run the scale with one harmonic through the filter and play for a TA to be checked off.

1.5.2

Design a highpass filter with the following specifications: fs = 8000, $f_{passband}$ = 700Hz, $f_{stopband}$ = 600Hz, do not change the default gain values . Run the scale with one harmonic through the filter and play for a TA to be checked off.

1.5.3

Design a bandpass filter with the following specifications: fs = 8000, $f_{stopband1}$ = 200Hz, $f_{passband1}$ = 300Hz, $f_{passband2}$ = 1000Hz $f_{stopband2}$ = 1100Hz, do not change the default gain values . Run the scale with one harmonic through the filter and play for a TA to be checked off.

1.5.4

Comment on how the pole/zero plots differ between the above filters.

1.5.5

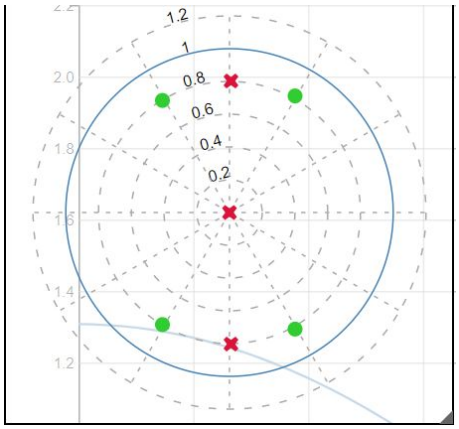
Generate the scale vector with k = 7. **Run the scale through each of the above filters, and comment on the differences.**

Part 2

2.1 Pole-Zero Plots

2.1.1

It is often very useful to graphically display the locations of a system's poles and zeros. The standard method for this is the pole- zero plot:



This is a two-dimensional plot of the z-plane that shows the unit circle, the real and imaginary axes, and the position of the system's poles and zeros. Zeros are typically marked with an 'o', while poles are indicated with an 'x'. Sometimes, a location has multiple poles and zeros. In this case, a number is marked next to that location to indicate how many poles or zeros exist there. The above figure, for instance, shows four zeros (two conjugate pairs), one"trivial" pole at the origin, and one other conjugate pair of poles.

Question: Where in the complex plane can zeros and poles be placed to have the strongest influence on the magnitude response of the filter?

2.1.2

System poles cause the system function to go to infinity at certain values of z because we are dividing by zero. On the one hand, this can have the desirable effect of raising the magnitude frequency response at certain frequencies. On the other hand, this can have some undesirable side effects. One somewhat significant problem is introduced if we have a pole outside the unit circle.

Consider the filter:

$$y[n] = x[n] + 2y[n-1]$$

Question: What are the poles and zeros of this filter?

Question: What is this filter's impulse response?

If the input is $x[n] = \delta[n]$ and $y[n] = 0$ for $n < 0$ then at $n = 0$, $y[n] = 1$. Then, every $y[n]$ after that is equal to twice the value of $y[n - 1]$. The value of this impulse response grows as time goes on. This system is termed 'unstable'. Unstable filters cause severe problems, and so we wish to avoid them at all costs. As a general rule of thumb, you can keep your filters from being unstable by keeping their poles strictly inside the unit circle. Note that the system's zeros do not need to be inside the unit circle to maintain stability.

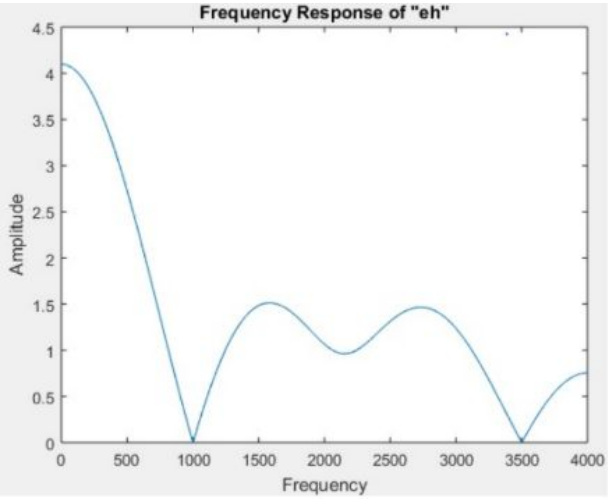
2.2 Vowel Creation

Now, it's your turn to use all this information to create vowels!

Note: For help on this GUI, please look at our demo video posted on Canvas. The GUI can be run in your browser by running index.html.

2.2.1

The magnitude response plot below has sampling frequency of 8000Hz. There are six non-trivial zeros (including conjugates). *Submit a table of the zeros location in normalized angular frequency.*



2.2.2

This filter represents the eh sound. Using the GUI, *create an FIR filter with six nontrivial zeros that matches the above magnitude response.*

Question: What are the filter coefficients b and a?

2.2.3

Submit a screenshot of your GUI (magnitude response + poles/zeros plots of the filter that you create).

Note: your magnitude response plot must be in normalized angular frequency.

2.2.4

Filter a glottal source signal, with fundamental frequency 150Hz and sampling frequency 8000Hz, through the eh filter. *Play this sound for your TA for checkoff*. If you can't finish in time, submit as a .wav file on Canvas.

2.3 Four More Vowel Sounds

See the filters.txt file in the data files!!!!!!!!!!!!!!!!!!

Now, using the **provided B and A filter coefficients,** use the GUI to *mimic the following frequency responses to create the vowels* ee , ah , oh , and oo .

Note that there is an option in the pole-zero GUI to import poles and zeros.

All the modeled magnitude response plots below have **sampling frequency 10000Hz**. You need to convert the important "peaks" and "dips" location to normalized angular frequency to put into your GUI.

For each vowel:

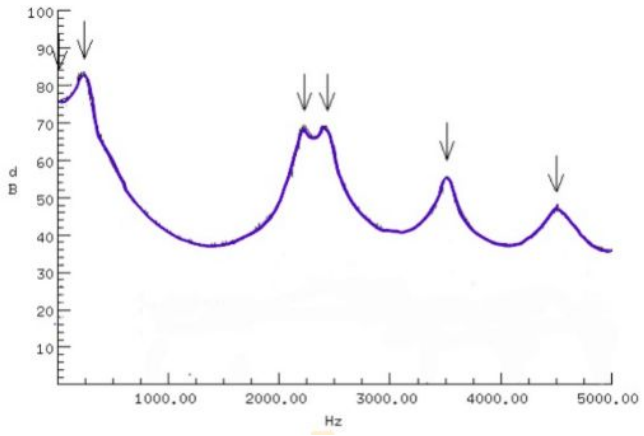
- Submit screenshots of your GUI (magnitude response and poles/zeros plot)
- **Submit the poles and zeros.**
- Make sure to either get each vowel sound checked off by a TA or submit .wav files.

Note that these amplitudes may be in dB, or decibels. The relation between and absolute value is:

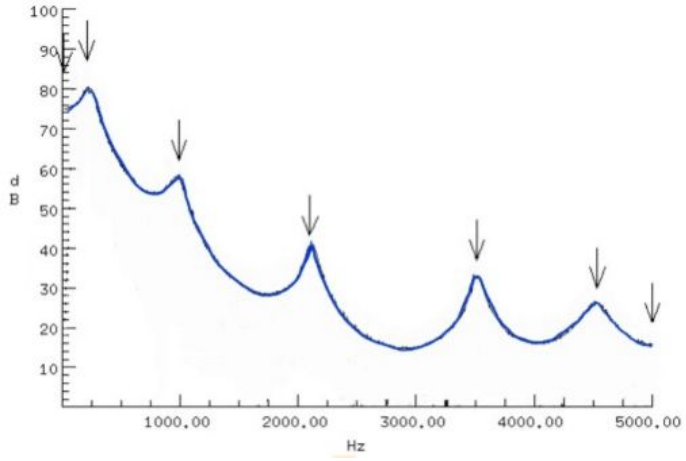
$$dB = 20\log(A)$$

Also note that changing the gain does not change the shape of the frequency response, nor does it change anything in the vowel produced, other than the volume. For instance, in the "ee" response, the minimum is at 40dB = 100 and the maximum is at 80dB = 100000 . Instead, you can create your vowel in the GUI by scaling the minimum to 1dB and the maximum to 100dB.

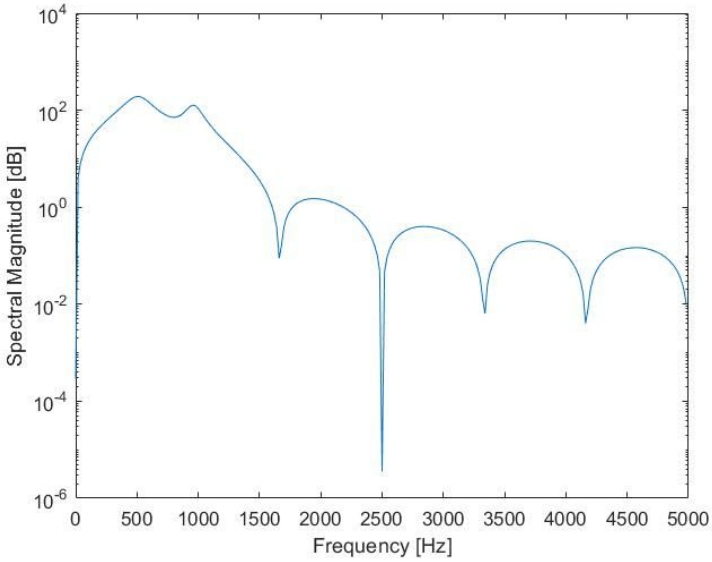
Hint: if vowel sounds are off, consider changing the number of harmonics when creating the glottal source signal. The vowels sound more distinguishable if they are played one after another.



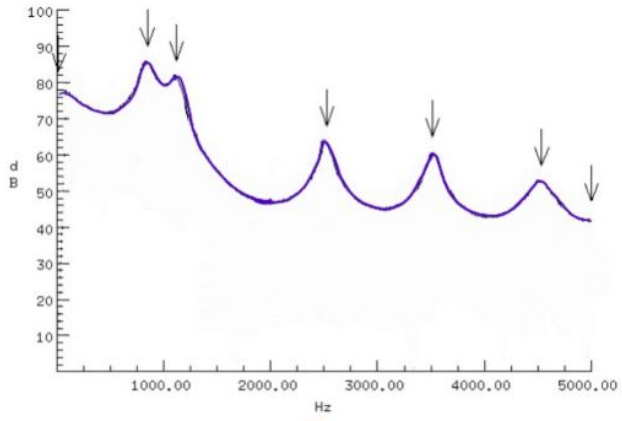
Ee


100
90
80
70
60
50
40
30
20
10
dB
1000.00
2000.00
3000.00
4000.00
5000.00
Hz

oo


10^4
10^2
10^0
10^{-2}
10^{-4}
10^{-6}
Spectral Magnitude [dB]
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
Frequency [Hz]

oh



Ah

2.4 Whispering Vowels

Whispering is done by filtering air through our vocal tract filter without allowing our vocal cords to vibrate. A whispered vowel is white noise filtered through the vocal tract. Use the randn() function (see help randn), which creates random numbers between 0 and 1, to create a vector with the same length as your glottal source signal vector, containing random numbers between zero and one. This signal is called "white noise", because it is broadband in frequency.

Filter this white noise through your five vowel filters (eh , ee , ah , oh , and oo) to create five whispered vowel sounds. Play them for your TA to get checked off.

You should have been checked off six times thus far in this lab: One for each of eh (in lab part 1), ee , ah , oh , and oo , and a sixth for whispering them.

2.5 Reconstruction of a Voweled Fugue

You'll remember from Lab 1 that MATLAB allows for structure, which group information together.

The nice thing about objects is that they allow us to use human-readable descriptions for our data.

```
melody.noteNumbers= [40 42 44 45 47 49 51 52];
melody.durations= [1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5];
melody.startPulses= [1 3 5 7 9 11 13 15];
```

We now introduce a new array:

```
melody.vowels= ['eh' 'ee' 'ah' 'oh' 'oo' 'oh' 'ah' 'ee'];
```

The vowel property is a two-character string representing the vowel to be played at each note.

Load chris_fugue from the chris_fugue.mat file. You'll notice that it four fields: noteNumbers , durations , startPulses , and vowels .

Hint: when finding the fundamental frequency corresponding to a note number, consider modifying the formula for f_{human}

and using 110 instead of 220.

Another hint: You might need to normalize the vowels so that the gains are uniform, as different filters may have different gains associated with them.

Play the correctly-voweled, correctly-timed, all-three-voices-added-together version of the Chris Fugue.

Play it for a TA for checkoff. If you can't finish in time, submit a .wav file on Canvas.

2.6 Extra Credit - Synthesize a Musical Piece of your own!

For up to fifteen points of extra credit, synthesize a piece of voweled music into a song, calling it yourname_song . Save it as yourname_song.mat . Submit your .m code, your PDF explaining it, and the .wav of your musical composition. A simple composition might get 5 points. A more complex one might get 10. A really well - done one might earn the maximum 15 points. Again, be creative!

Just a couple ideas for the full 15 points:

- Add consonants to make understandable words!
- Transition between different vowels!
- Adjust vowels slightly at different pitches

Appendix

glottal_key_to_note Skeleton Code:

```
function xx = key_to_note(k, keynum, dur,fs)
%{
KEY_TO_NOTE: Produce a sinusoidal waveform corresponding to agiven piano key number
Input Args:
k: number of harmonics (default = 1)
keynum: number of the note on piano keyboard
```

```
dur: duration of the note (in seconds)

Output:

xx: sinusoidal waveform of the note

%}

tt = 0:(1/fs):dur-1/fs;

for ii = 1:k

        f =  ; %FILL THIS LINE IN

        xx = ; %FILL THIS LINE IN

end
```

Scale matrix code:

```
%% Script to generate vector of octaves

%Using the code provided, generate a 7x8 matrix of the 8 notes in octaves

%1-7

baseKey = 4;

offsets = [0,2,4,5,7,9,11,12];

keynums = zeros(7,length(offsets));


for i = 1:7 %For loop for octaves 1-7

  keynums(i,:) = baseKey + 12*(i-1) + offsets;

end
```