Lab 7: Poles and Zeros – Vowel Synthesis

Compiled by Barukh Rohde at the University of Florida for the Summer 2016 Signals & Systems course
Stolen in parts from: UMich EECS/BME, UDelaware Speech Research Lab, UT-Dallas

Lab Part 1

1.1 Introduction (nothing to submit)

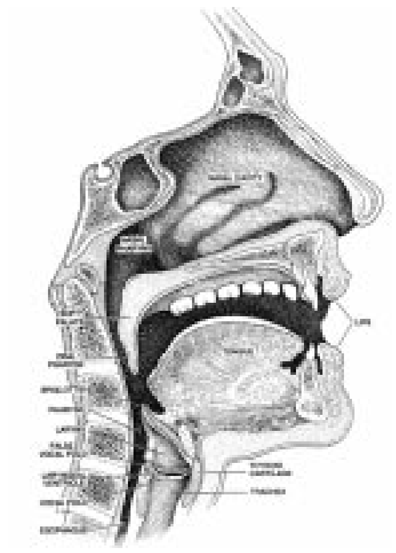
So far, we've been considering filters as systems that we design and then apply to signals to achieve a desired effect. However, filtering is also something that occurs everywhere, without the intervention of a human filter designer. At sunset, the light of the sun is filtered by the atmosphere, often 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 pair 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 to develop a graphical understanding of how poles and zeros affect the frequency response of a system.

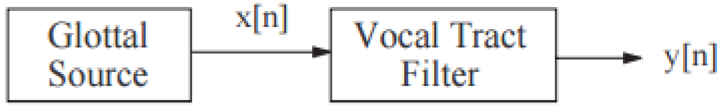
The objective of this lab is to use pole-zero placement design to create filters and to use them to synthesize vocal music!

1.2 Modeling Vowel Production

In order to gain a better understanding of vowel production and how we can model it using discrete-time filters, we need to introduce some theory. Speech production is primarily governed by the larynx (or voice box) and the vocal tract:



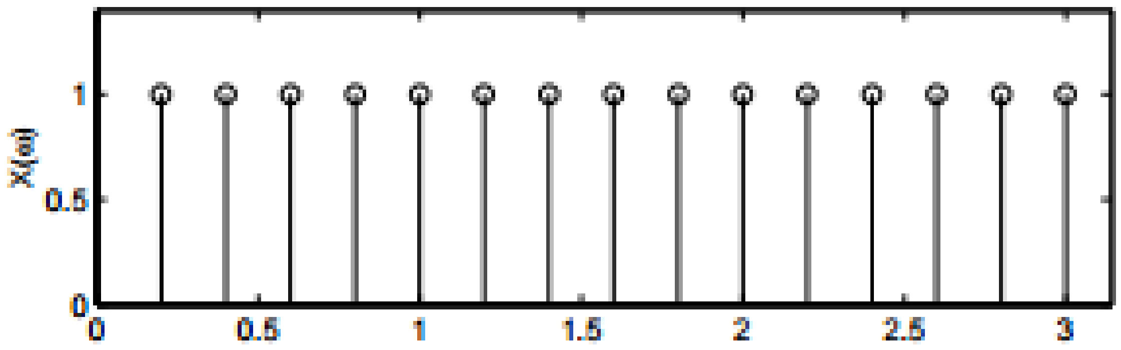
When we speak a vowel, the lungs push a stream of air through the larynx and the vocal folds, or vocal chords. Given the appropriate muscular tension, this stream of air causes the vocal folds to vibrate, creating a nearly periodic fluctuation in air pressure passing through the larynx. The fundamental 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:

$$x(t) = \sum_k A_k cos(\omega_k t + \varphi_k)$$

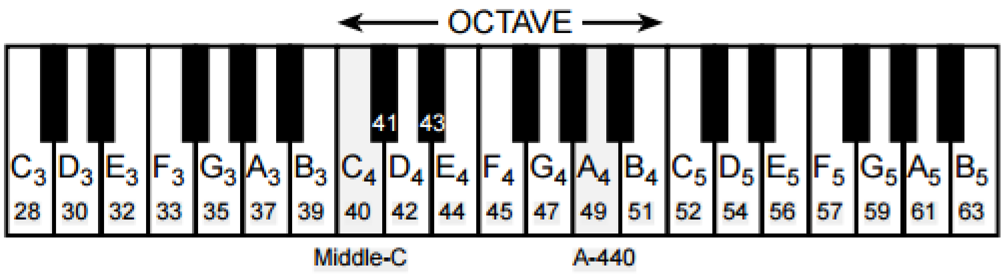
The glottal source signal, the signal formed by the air pressure fluctuations produced by the vibrating vocal cords, is typically modeled as a periodic pulse train. To a first approximation, we can assume that the frequency spectrum of this pulse train is composed of equal amplitude harmonics:



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 60 nonzero harmonics (A_{-30} through A_{30}), with fundamental frequency 150Hz. What is the minimum sampling frequency necessary to avoid aliasing?

1.3.1 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 the 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:

$$f_{piano} = 440(2^{\frac{key\ number\ -49}{12}})$$

In this lab, we'll consider the human-voiced equivalent of a piano note to be half of its frequency, an octave below it, or 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:

$$f_{human} = 220(2^{\frac{key\ number\ -49}{12}})$$

Write a MATLAB function (similar to your key2note function from Lab 3) that takes in a key number and a duration, to produce a glottal source signal of given duration with fundamental frequency corresponding to the desired note:

```
function xx = glottalkey2note(keynum, dur)
```

1.3.2 Synthesize a Song – Mary Had a Bleating Lamb

As you learned in Lab 3, multiple notes can be played in order by concatenating their row vectors as follows:

```
xx = [x1 x2];
```

Use `glottalkey2note()` to write a script, `play_glottallamb.m`, that plays a series of notes. Use the following skeleton code to write your script:

```
% --------------play_lamb.m-------------- %
mary.keys = [44 42 40 42 44 44 44 42 42 42 44 47 47];
% NOTES: C D E F G
% Key #40 is middle-C
mary.durations = 0.25 * ones(1,length(mary.keys));
fs = 8000;  % 11025 Hz also works
xx = zeros(1, sum(mary.durations)*fs + length(mary.keys));
n1 = 1;
for kk = 1:length(mary.keys)
     keynum = mary.keys(kk);
     tone =  % <------- Fill in this line
     n2 = n1 + length(tone) - 1;
     xx(n1:n2) = xx(n1:n2) + tone; %<------- Insert the note
     n1 = n2 + 1;
end
soundsc(xx, fs)
```

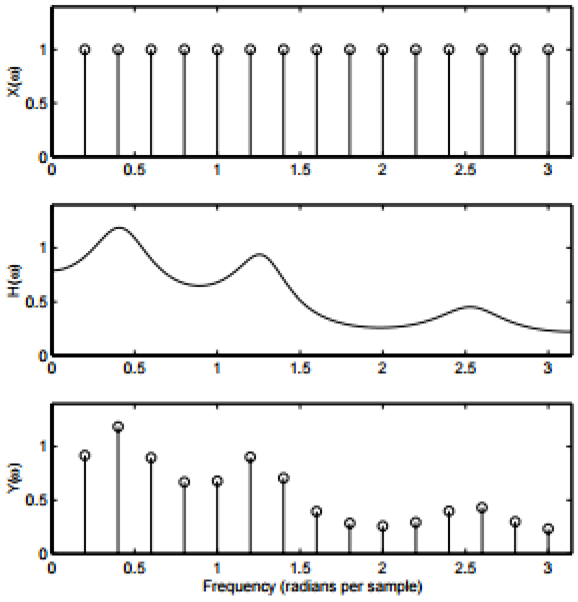
Generate the sound and play it for a TA. Your TA will check you off, and you'll move on to the next section. This check-off isn't for credit – it's just to make sure you're on track.
Plot the frequency-time spectrogram of Mary using one of `plotspec`, `specgram`, or `spectrogram` (or any other Matlab program you can think of).

1.3.3 Synthesize a Song – Baaaaaaaach Taaaalks

Now, **use your `glottalkey2note()` to synthesize your Bach Fugue from Lab 3**.

1.4 Introduction to the Source-Filter Model (nothing to submit)

The second block of the source-filter model is the vocal tract filter. This is a discrete-time filter that mimics the spectrum-shaping properties of the vocal tract. Since we are assuming a source signal with equal-amplitude harmonics, the vocal tract filter provides the spectral envelope for our output signal. That is, when we filter the source signal with fundamental frequency $\widehat{\omega}_0$ radians per sample, the k^{th} harmonic of the output signal will have an amplitude equal to the filter's magnitude frequency response evaluated at $k\widehat{\omega}_0$.



The magnitude spectrum of the glottal source signal is shown on top, the magnitude frequency response of the vocal tract filter is shown in the center, and the magnitude spectrum of the output signal, which is the signal that models the specific vowel signal. One may clearly see that, as desired, the envelope of the spectrum of the vowel signal model matches the spectrum of the vocal tract filter. We will make such source-filter models for particular vowel signals, by measuring the spectrum of the vowel signal and designing an IIR vocal tract filter whose frequency response approximates this spectrum.

1.5: Filters in the Z-Domain

1.5.1 Filters and the ***z***-Transform (nothing to submit)

Previously, we have presented the general time-domain input-output relationship for a causal filter given by the convolution sum:

$$y[n] = x[n] * h[n] = \sum_k h[k]x[n-k] = \sum_k x[k]h[n-k]$$

Where $x[n]$ is the input signal, $y[n]$ is the output signal, and $h[n]$ is the filter's impulse response. Using the z-transform, we can also describe the input/output relationship in the z-domain as

$$Y(z) = H(z)X(z)$$

Where $X(z)$is the z-transform of $x[n]$, which is the complex-valued function defined on the complex plane by

$$X(z) = \sum_n x[n]z^{-n}$$

And where $Y(z)$is the z-transform of $y[n]$, defined in a similar fashion, and where $H(z)$ is the *system function* of the filter, which is a complex-valued function defined on the complex plane by one of the following equivalent definitions:

1. The system function is the z-transform of the filter impulse response $h[n]$, i.e.

$$H(z) = \sum_n h[n]z^{-n}$$

2. For $X(z)$ and $Y(z)$ as defined above, the system function is given by

$$H(z) = \frac{Y(z)}{X(z)}$$

The system function has a very important relationship to the frequency response of a system, $H(e^{j\hat{\omega}})$. The system function evaluated at $e^{j\hat{\omega}}$ is equal to the frequency response evaluated at frequency $\hat{\omega}$. That is,

$$H(z)|_{z=e^{j\hat{\omega}}} = H(e^{j\hat{\omega}})$$

This is simply the definition of a system's frequency given its impulse response $h[n]$.

1.5.2 FIR Filters and the $\boldsymbol{z}$-Transform (nothing to submit)

For a causal FIR filter, one can easily determine the system function using either of the equivalent definitions given above. However, let us highlight the use of the second definition, which will be useful in the next subsection where the first definition is difficult to apply. In particular, for a causal FIR filter with coefficients $\{b_0, b_1, \ldots, b_M\}$, the general time-domain input-output relationship for a causal FIR filter is given by the difference equation

$$y[n] = b_0x[n] + b_1x[n-1] + b_2x[n-2] + \cdots + b_Mx[n-M]$$

Taking the z-transform of both sides of this difference equation yields

$$Y(z) = b_0X(z) + b_1X(z)z^{-1} + b_2X(z)z^{-2} + \cdots + b_MX(z)z^{-M} =$$

$$= X(z)(b_0 + b_1z^{-1} + b_2z^{-2} + \cdots + b_Mz^{-M})$$

Where we have used the fact that the z-transform of $x[n-n_0]$ is $X(z)z^{-n_0}$.
Dividing both sides of the above by $X(z)$ gives the system function:

$$H(z) = \frac{Y(z)}{X(z)} = b_0 + b_1z^{-1} + b_2z^{-2} + \cdots + b_Mz^{-M}$$

Notice that $H(z)$ is a polynomial of order M. We can factor the above complex-valued polynomial as

$$H(z) = K(1 - r_1z^{-1})(1 - r_2z^{-1})(1 - r_3z^{-1}) \ldots (1 - r_Mz^{-1})$$

Where K is a real number called the gain, and $\{r_1, \ldots, r_M\}$ are the M roots or zeros of the polynomial, i.e. the values r such that $H(r) = 0$. We typically assume that the filter coefficients b_k are real. In this case, the zeros may be real or complex, and if one is complex, then its complex conjugate is also a zero. That is, complex roots come in conjugate pairs. Note that the system function $H(z)$ of a causal FIR filter is completely determined by its gain and its zeros. Previous ways of describing a filter have included the filter coefficients, the impulse response sequence, the frequency response function, and the system function. We can now think of $\{K, r_1, \dots, r_M\}$ as one more way to describe a filter. We will see that when it comes to designing an FIR filter to have a certain desired frequency response, the description of the filter in terms of its gain and its zeros is by far the most useful. In other words, the best way to design a filter to have a desired frequency response (e.g., a low pass filter) is to appropriately choose its gain and zeros. One may then find the system function by multiplying out the terms of the equation, and then picking off the filter coefficients from the system function. For example, the number multiplying z^{-3} in the system function is the filter coefficient b_3.

Note that the gain of the filter does not affect the shape of the frequency response; it only scales it. The fact that we may design the frequency response of a causal FIR filter by choosing its zeros stems from the following principle:

If a filter has a zero r located on the unit circle, i.e.$|r| = 1$, then $H(r) = 0$, i.e. the frequency response has a null at frequency $\angle r$. Similarly, if a filter has a zero r located close to the unit circle, i.e. $|r| \approx 1$, then $H(r) \approx 0$, i.e. the frequency response has a dip at frequency $\angle r$.

From this fact, we see that we see that we can make a filter block a particular frequency, i.e. create a null or a dip in the frequency response, simply by placing a zero on or near the unit circle at an angle equal to the desired frequency and at its complex conjugate. On the other hand, the frequency response at frequencies corresponding to angles that are not close to these zeros will have large magnitude. The filter will "pass" these frequencies. The specific procedure to design such a filter is the following.

1. Choose frequencies at which the frequency response should contain a null or a dip.
2. Place zeros at those frequencies, with $r = Ae^{j\hat{\omega}}$ with $A = 1$ or Aslightly less than 1 depending on whether a null or dip is desired at that frequency.
3. Form the system function $H(z)$ from these zeros and multiply it out to express $H(z)$as a polynomial with terms that are powers of z^{-1}.
4. Identify the FIR filter coefficients, the coefficients of that polynomial.
5. Use this filter to filter your signal.

1.5.3 IIR Filters and Rational System Functions **(nothing to submit)**

We now consider IIR filters. The general time-domain input-output relationship for a causal IIR filter is given by the difference equation

$$y[n] = b_0x[n] + b_1x[n-1] + b_2x[n-2] + \cdots + b_Mx[n-M] + \\ + a_1y[n-1] + a_2y[n-2] + \cdots + a_Ny[n-N]$$

Here, we have the usual FIR filter coefficients, b_k, but we also have another set of coefficientsa_k, which *multiply past values of the filter's output.* We will call the b_k's the feedforward coefficients and the a_k's the feedback coefficients. If the a_k's are zero, then this filter reduces to a causal FIR filter. As an example, consider the simple IIR filter with difference equation:

$$y[n] = x[n] + \frac{1}{2}y[n-1]$$

Note that in some texts (and in MATLAB), the feedback coefficients are defined as the negatives of the a_k coefficients given here. Because of this, you should always be sure to check which convention is used.

What is the impulse response of this filter? If we assume that $y[n] = 0$for $n < 0$, one can straightforwardly show that the impulse response is

$$h[n] = \left(\frac{1}{2}\right)^n, \qquad n \geq 0$$

which is never zero for any positive n. (Note that the impulse response is generally not so simple to compute; this is an unusual case where the impulse response can be obtained by inspection.) Thus, by introducing feedback terms into our difference equation, we have produced a filter with an infinite impulse response, i.e., an IIR filter. In general, computing the system function by taking the z-transform of the resulting infinite impulse may not be trivial because of the required infinite sum, and also because it may be difficult to find the impulse response. However, we can use the fact that $H(z) = \frac{Y(z)}{X(z)}$to determine the system function. To do this, we first collect the $y[n]$ terms on the left side of the equation and take the z-transform of the result.

$$y[n] - a_1 y[n-1] - \cdots - a_N y[n-N] = b_0 x[n] + b_1 x[n-1] + \cdots + b_M x[n-M]$$

$$Y(z) - a_1 Y(z) z^{-1} - \cdots - a_N Y(z) z^{-N} = X(z) b_0 + b_1 X(z) z^{-1} + \cdots + b_M X(z) z^{-M}$$

$$Y(z)(1 - a_1 z^{-1} - \cdots - a_N z^{-N}) = X(z)(b_0 + b_1 z^{-1} + \cdots + b_M z^{-M})$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}}{1 - a_1 z^{-1} - \cdots - a_N z^{-N}}$$

We can factor this rational polynomial to convert the system function to the form:

$$H(z) = K \frac{(1 - r_1 z^{-1})(1 - r_2 z^{-1})(1 - r_3 z^{-1}) \dots (1 - r_M z^{-1})}{(1 - p_1 z^{-1})(1 - p_2 z^{-1})(1 - p_3 z^{-1}) \dots (1 - p_N z^{-1})}$$

The roots of the polynomial in the numerator, $\{r_1, \dots, r_M\}$, are again called the zeros of the system function. The roots of the polynomial in the denominator, $\{p_1, \dots, p_N\}$, are called the poles of the system function. $K$is again a gain factor that determines the overall amplitude of the system's output. As before, the zeros are complex values where $H(z)$ goes to zero. The poles, on the other hand, are complex values where the denominator goes to zero and thus the system function goes to infinity.$H(z)$is undefined at the location of a pole, but is large in the neighborhood of a pole. Again, we typically assume that the filter coefficients b_k and a_k are real, so both the poles and zeros of the system function must be either purely real or must appear in complex conjugate pairs. Just as we could completely characterize an FIR filter by its gain and its zeros, we can completely characterize an IIR filter by its gain, its zeros, and its poles. As in the FIR case, this is typically the most useful characterization when designing IIR filters. As before, if the system function has zeros near the unit circle, then the filter magnitude frequency response will be small at frequencies near the angles of these zeros. On the other hand, if there are poles near the unit circle, then the magnitude frequency response will be large at frequencies near the angles of these poles. With FIR filters we could directly design filters to have nulls or dips at desired frequencies. Now, with IIR filters, we can design peaks in the frequency response, as well as nulls. The specific procedure is the following:

1. Choose frequencies at which the frequency response should contain a null, a dip, or a peak.
2. Place zeros$r = Ae^{j\hat{\omega}}$at the frequencies at which a null or dip should occur.
3. Place poles $p = Ae^{j\hat{\omega}}$at the frequencies at which a peak should occur.
4. Form the system function $H(z)$. The numerator is formed from the zeros and multiplied out. The denominator is formed from the poles, multiplied out. Express $H(z)$as a ratio of polynomials with terms that are powers of z^{-1}.
5. Identify the IIR filter coefficients$b_k$and a_k, the coefficients of those polynomials.
6. Use this filter to filter your signal.

1.5.4 Poles and Zeros at the Origin and at Infinity **(nothing to submit)**

Here, we have defined our system functions in terms of negative powers of z. This is because our general forms for FIR and IIR filters are defined in terms of time delays, and multiplication of the z-transform of some signal $X(z)$ by z^{-1} is equivalent to a time delay of one sample. However, there are may be "hidden" poles and zeros when we express a system function in this matter. Consider first the system function for an FIR filter, e.g.

$$H(z) = \frac{Y(z)}{X(z)} = b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_M z^{-M}$$

If we try to evaluate this system function at$z = 0$, we will immediately find that we are dividing by zero. Thus, there is actually a pole at the origin of this system function. To reveal such "hidden" poles and zeros, we express the system function in terms of positive powers of z, as

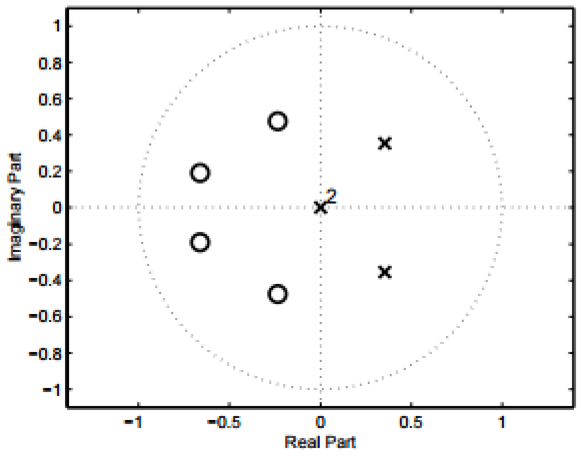
$$H(z) = \frac{b_0 z^M + b_1 z^{M-1} + b_2 z^{M-2} + \cdots + b_M}{z^M}$$

By the Fundamental Theorem of Algebra, we know that the numerator polynomial has M roots, and thus the system has M zeros. However, the denominator, z^M, has M roots as well, all at $z = 0$. This means that our causal FIR system function has M poles at the origin. In some cases, like the previous example, we find extra poles at the origin. In other cases, we find extra zeros at the origin. For example, the filter $y[n] = y[n-1] + x[n]$, has $H(z) = \frac{1}{1-z^{-1}} = \frac{z}{z-1}$, from which we see there is one zero at the origin. In still other cases we find zeros at infinity. For example, the filter $y[n] = x[n-1]$, has $H(z) = z^{-1} = \frac{1}{z}$, from which we see that there is a zero at infinity. In still other cases, we find combinations of the previous cases. We will call poles and zeros located at the origin, or at infinity, "*trivial poles and zeros*" because they do not affect the system's magnitude frequency response, though they affect the phase response. In this laboratory, we will primarily be concerned with nontrivial poles and zeros (those not at the origin or at infinity). Note that there will always be the same number of poles and zeros in a linear time-invariant system, including both trivial and nontrivial poles and zeros. The total number equals the Mor

N, whichever is larger. Such facts are useful for checking to make sure that you have accounted for all poles and zeros in a system. Note also that if one chooses filter coefficients such that the numerator and denominator contain an identical factor, then these factors "cancel" each other; i.e. the filter is equivalent to a filter whose system function has neither factor.

1.5.5 Pole-Zero Plots

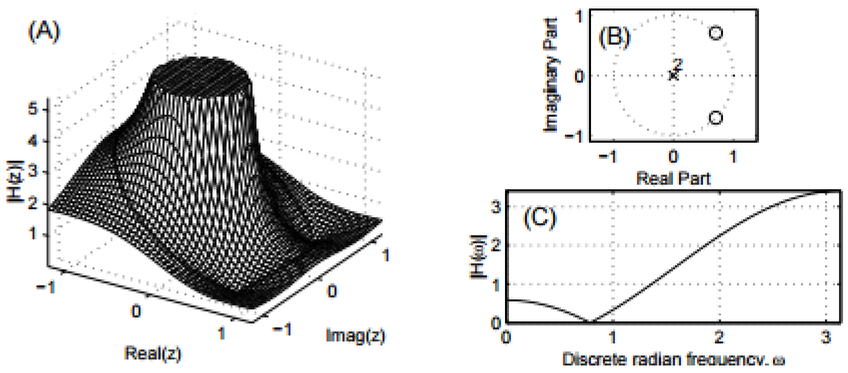
It is often very useful to graphically display the locations of a system's poles and zeros. The standard method for this is the *pole-zero plot*:



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), two "trivial" poles at the origin, and one other conjugate pair of poles. Where in the complex plane can zeros and poles be placed to have the strongest influence on the magnitude response of the filter?

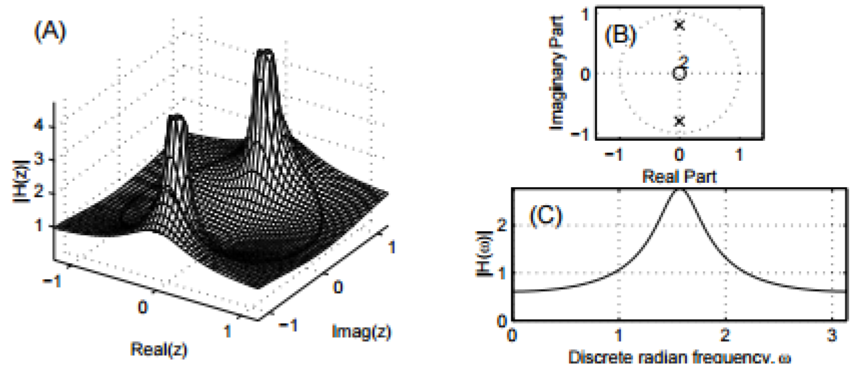
1.5.6 Graphical Interpretation of the System Function **(nothing to submit)**

If we take the magnitude of $H(z)$, we can think of $|H(z)|$as defining a (strictly positive) *surface* over the z-plane for which the *height* of the surface is given as a function of the complex number z. Here's an example of such a surface:

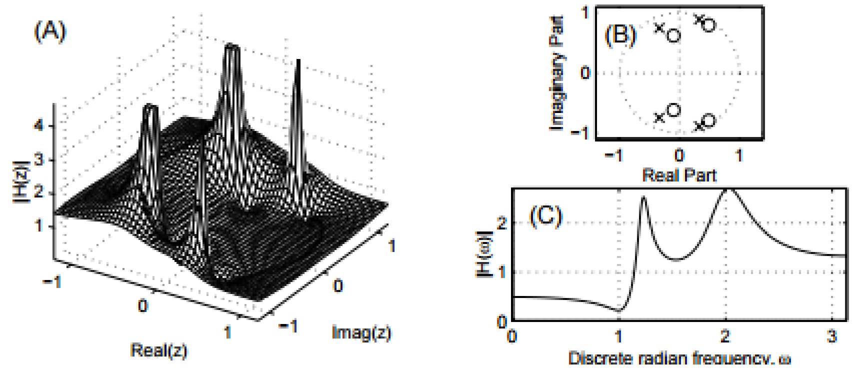


This system function has two zeros (which form a complex conjugate pair) and two poles at the origin. Notice that the unit circle is outlined on the surface $|H(z)|$. The height of the surface at

$z = e^{j\hat{\omega}}$ (i.e., on the unit circle) defines the magnitude of the frequency response, $|H(e^{j\hat{\omega}})|$, which is shown to the right of the surface. On the figure above, we can see two points where the surface $|H(z)|$ goes to zero; these are the zeros of the system function. Notice how the surface is "pulled down" in the vicinity of these zeros, as though it has been "tacked to the ground" at the location of the zeros. Near the system's zeros, the magnitude frequency response has a low point because of the influence of the nearby zero. Also notice how the surface is "pushed up" at points far from the zeros; this is another common characteristic of system function zeros. Thus, the magnitude frequency response has higher gain at points far away from the zeros. Since the two poles in this figure are at the origin, they have no effect on the system's magnitude frequency response. This will be the case for all FIR filters.



This figure shows the surface $|H(z)|$as defined by a different system function. This system function has two poles (which form a complex conjugate pair) and two zeros at the origin. Notice how the poles "push up" the surface near them, like poles under a tent. The surface then typically "drapes" down away from the poles, getting lower at points further from them. The magnitude frequency response here has a point of high gain in the vicinity of the poles. (Again, the zeros in this system function are located at the origin, and thus do not affect the magnitude frequency response.)



This figure shows the surface for a system function which has poles and zeros interacting on the surface. This system function has four poles and four zeros. Notice the tendency of the poles and zeros to cancel the effects of one another. If a pole and a zero coincide exactly, they will completely cancel. If, however, a pole and a zero are very near one another but do not have exactly the same position, the z-plane surface must decrease in height from infinity to zero quite rapidly. This behavior allows the design of filters with rapid transitions between high gain and low gain.

1.5.7 Poles and Stability

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:

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

What are the poles and zeros of this filter?
What is this filter's impulse response?

If the input is $x[n] = \delta[n]$ and $y[n] = 0 \text{for} n < 0$, 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 to maintain stability.

1.5.8 Filter Design Using Manuel Pole-Zero Placement

In this lab, we will explore a method of filter design in which we place poles and zeros on the z-plane in order to match some target frequency response. You will be using a MATLAB graphical user interface, called `pole_zero_place3d()`, to do this. The interface allows you to place, delete, and move poles and zeros around the z-plane. The frequency response will be displayed in another figure and will change dynamically as you move poles and zeros. To keep the filter's coefficients real, you will design by placing a pair of poles and zeros on the z-plane simultaneously.

The approach for this method depends somewhat on the type of filter that we wish to design. If we want an FIR filter (i.e., a filter that has no poles), we need to use zeros to "pin down" the frequency response where it is low, and allow the frequency response to be pushed upwards in regions where there are no zeros. Note that if we put a zero right on the unit circle, we introduce null in the frequency response at that point. Conversely, the closer to the origin that we place a zero, the less effect it will have on the frequency response (since it will begin to affect all points on the unit circle roughly equally). You might use the example of the running average filter and bandpass filters (given in Chapter 7 of DSP First) as a prototype of how to use zeros to design FIR filters using zero placement. If we wish to design an IIR filter (with both poles and zeros), it usually makes sense to start with the poles since they typically affect the frequency response to a greater extent. If the frequency response that we are trying to match has peaks on it, this suggests that we should place a pole somewhere near that peak (inside the unit circle). Then, use zeros to try to pull down the frequency response where it is too high. As with zeros, poles near the origin have relatively little effect on the system's frequency response. Regardless of which type of filter we are designing, there are a couple of methodological points that should be mentioned. First, moving a pole or zero affects the frequency response of the entire system. This means that we cannot simply optimize the position of each pole-pair and zero-pair individually and expect to have a system which is optimized overall. Instead, after adjusting the position of any pole-pair or zero-pair, we generally need to move many of the remaining pairs to compensate for the changes. This means that filter design using manual pole-zero placement is fundamentally an iterative design process. Additionally, it is important that you consider the filter's gain. Often we cannot adjust the overall magnitude of the frequency response using just poles and zeros. Thus, to match the frequency response properly, you may need to adjust the filter's gain up or down. The pole-zero design interface that you will use in this lab includes an edit box where you can change the gain parameter. Alternately, by dragging the frequency response curve, you can change the gain graphically.

A related idea is that of spectral slope. By having a pair of poles or zeros inside the unit circle and near the real axis, we can adjust the overall "tilt" of the frequency response. As we move the pair to the right and left on the z-plane, we can adjust the slope of the system's frequency response up and down.

1.6 Some MATLAB Commands for this Lab

1.6.1 Calculating the Frequency Response of IIR Filters **(nothing to submit)**

Previously, we have used `freqz()` to compute the frequency response of FIR filters. We can use the same command to compute the frequency response of an IIR filter. If our filter is defined by feedforward coefficients b_k stored in a vector `B` and feedback coefficients a_k stored in a vector `A`, we compute the frequency response at 256 points using the command:

```
>>[H,w]=freqz(B,A,256);
```

`H` contains the frequency response and `w` contains the corresponding discrete-time frequencies. Alternatively, we can compute the frequency response only at a desired set of frequencies. For example, the command

```
>>[H,w]=freqz(B,A,[pi/4,pi/2,3*pi/4]);
```

returns the frequency response of the filter at the frequencies $\frac{\pi}{4}$, $\frac{\pi}{2}$, and $\frac{3\pi}{4}$.

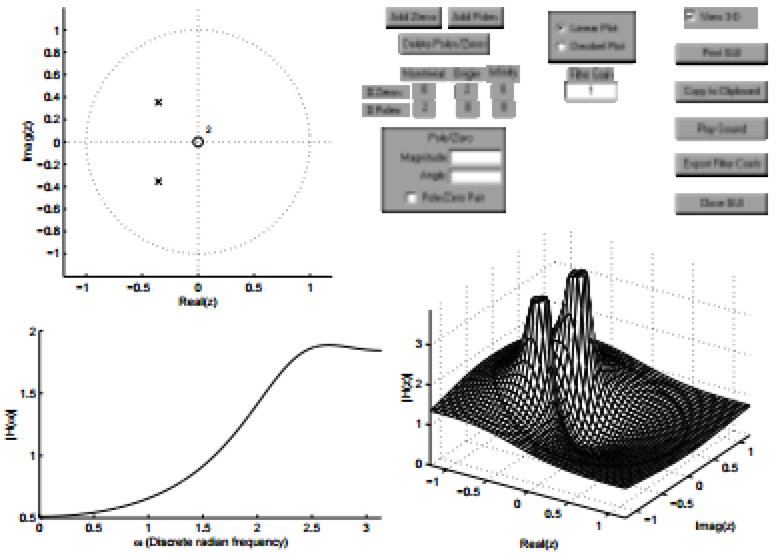
1.6.2 Pole-Zero Place 3-D **(nothing to submit)**

In this laboratory, we will primarily be exploring filter design using manual pole-zero placement. To help us do this, we will be using a MATLAB graphical user interface (GUI) called **Pole-Zero Place 3-D**. Pole-Zero Place 3-D allows you to place, move, and delete poles and zeros on the z-plane, and provides immediate feedback by displaying the filter's frequency response and the $|H(z)|$ surface. Additionally, it calculates some useful statistics for assessing the quality of a particular filter design. To run this program you need to download two different files from Canvas: *pole_zero_place3d.m* and*pole_zero_place3d.fig*.

Make sure they're both in a folder within MATLAB's search path, and run the following command:

```
>>pole_zero_place3d;
```

Note: This has been designed for a Windows computer.
Once the program starts, the following GUI window will appear:



The axis in the upper left of the window shows a portion of the z-plane with the unit circle. In the lower left is an axis that displays the frequency response of the system. In the lower right is a 3D axis which displays a 3D graph of the $|H(z)|$ surface.

1. To add poles or zeros to the z-plane, click the *Add Zeros* or *Add Poles* button and then click on the z-plane plot in the upper left of the GUI. Note that the program also adds the hidden poles and zeros that accompany nontrivial poles and zeros. Specifically, for each zero that is added a pole is added at the origin, or if there is already at least one zero at the origin, instead of adding a pole at the origin, one zero at the origin is removed, i.e. cancelled. Moreover, for each pole that is added, a zero is added at the origin, or if there are already poles at the origin, one pole is removed, i.e. cancelled. The system does not allow one to place zeros at infinity, and it can be shown that zeros at infinity will not be induced by any other choices of poles or zeros.

2. To move a real pole or zero (or a conjugate pair of complex poles or zeros), you must first select the pole/zero by clicking on one member of the pair. Then, you can drag it around the z-plane, use the arrow keys to move it, or move it to a particular location by inputting the magnitude and angle (in radians) in the*Magnitude* and*Angle* editboxes.

3. To delete a pole or zero (or pair), select it and hit the *Delete Poles/Zeros* button. Again, the system will maintain an equal number of poles and zeros by also removing poles or zeros from the origin as necessary. This may also have the effect of no longer cancelling other poles and zeros, and thus the total number of poles and zeros that appear at the origin will change.

4. To change the filter's gain, you can either use the *Filter Gain* edit box or you can click-and-drag the blue frequency response curve in the lower left.

5. To toggle between linear amplitude and decibel displays in the lower two plots, select the

desired radio button above the *Filter Gain* edit box, though at present time I'm having trouble with this feature.

6. To begin with an initial filter configuration defined by the feedforward coefficients, `B`, and the feedback coefficients, `A`, start the program with the command

```
>>pole_zero_place3d(1, 200, B,A);
```

   This is useful if you wish to start continue working on a design that you had previously saved.

7. To print the GUI window, Windows users can either use the *Copy to Clipboard* button to copy an image of the figure into the clipboard, or you can print the figure using the *Print GUI* button. (Or just take a screenshot.)

8. To save your current design, use the *Export Filter Coefs* button. The feedforward and feedback coefficients will be stored in the variables `B_pz` and`A_pz`,respectively.

9. To hear your filter's response to periodic signal with equal-amplitude harmonics, press the *Play Sound* button. This is particularly useful when using the GUI to design vocal tract filters for vowel synthesis.

Note that there is a similar GUI, `pezdemo`, contained in the SPFirst toolbox. Use whichever one you prefer.

1.6.3 Converting Between Filter Coefficients and Zeros/Poles **(nothing to submit)**

Given a set of filter coefficients, we often need to determine the set of poles and zeros defined by those coefficients. Similarly, we often need to take a set of poles and zeros and compute the corresponding filter coefficients. There are two MATLAB commands that help us do this. First, if we have our filter coefficients stored in the vectors `B` and `A`, we compute the poles and zeros using the commands

```
>>zeros=roots(B);
>>poles=roots(A);
```

This is because the system zeros are simply the roots of the numerator polynomial whose coefficients are the numbers in `B`, while the system zeros are simply the roots of the denominator polynomial whose coefficients are the numbers in `A`. To convert back, use the commands

```
>>B=poly(zeros);
>>A=poly(poles);
```

Notethatwe losethefilter'sgaincoefficient,K,withbothoftheseconversions.

1.6.4 Automatic All-Pole Modeling (nothing to submit)

Using the MATLAB command `aryule()`,we can compute an all-pole filter model for a discrete-time signal.That is, `aryule()` automatically finds an all-pole filter whose magnitude frequency response tries to match the magnitude frequency response of the signal. The command

```
>>A=aryule(signal,N);
```

returns the filter feedback coefficients a_k as a vector `A`. The parameter `N` indicates how many poles we wish to use in our filter model. Once we have `A`, we can compute the filter's frequency response at 256 points using `freqz()` as

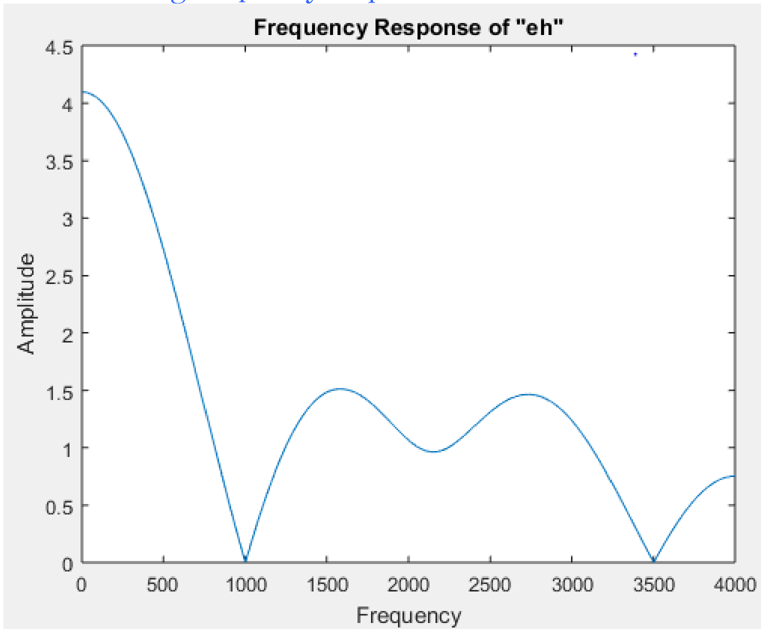
```
>>[H,w]=freqz(1,A,256);
```

1.7 Vowel Creation

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

For each vowel, submit screenshots of your GUI, submit the filter coefficients a_k and b_k and the pole and zero locations, and submit a plot of the frequency response of your vowel filter, as well as either a *.wav* file or a password.

Using the `pole_zero_place3d` or `pezdemo` GUI, create an FIR filter with six nontrivial zeros that matches the following frequency response:



This filter represents the 'eh' sound.
What are the filter coefficients b and a?
Filter a glottal source signal, with fundamental frequency 150Hz, **through the 'eh' filter**.
Submit the frequency response of the filter that you create.
Play this sound for your TA, who will check you off (alternately, submit as a *.wav* file).
Note that the `filter(b,a,xx)` command will prove useful here.

Lab Part 2

2.1 Four More Vowel Sounds

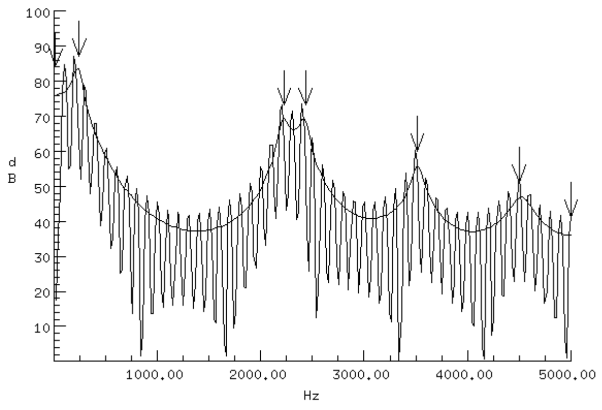
Now, using as many poles and zeros as you'd like, use the GUI and the poles-and-zeros fundamentals that you've learned to mimic the following frequency responses to create the vowels "ee", "ah", "oh", and "oo". For each vowel, submit screenshots of your GUI, submit the filter coefficients a_k and b_k and the pole and zero locations, and submit a plot of the frequency response of your vowel filter, as well as either a *.wav* file or a password.

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

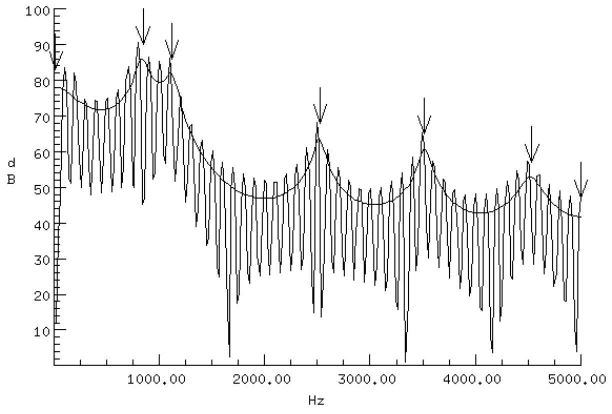
$$dB = 20\log(A)$$

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 $80dB = 10{,}000$. Instead, you can create your vowel in the GUI by scaling the minimum to 1 and the maximum to 100.

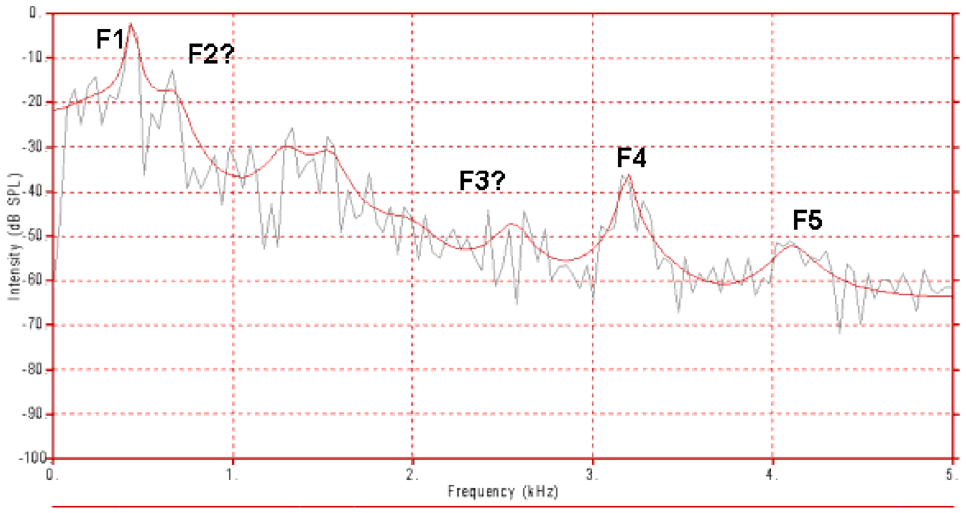
"EE"



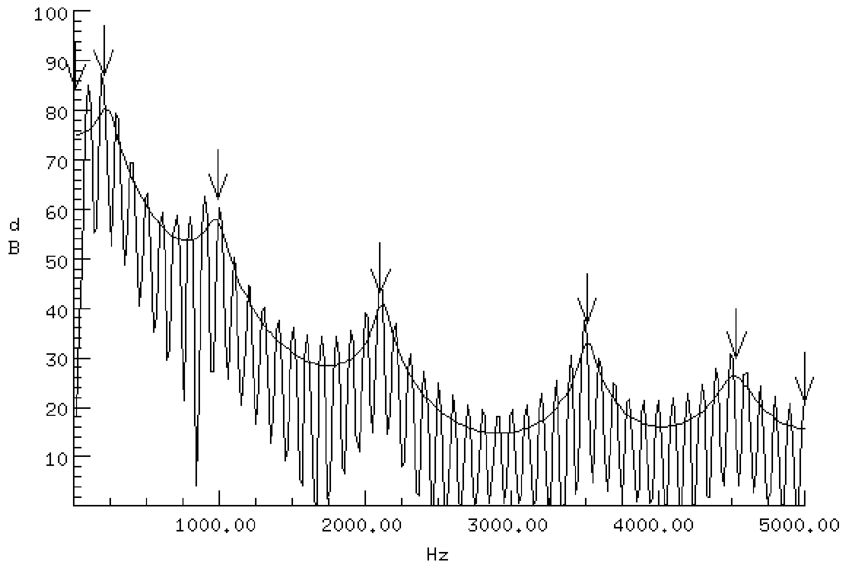
"AH"



“OH”



“OO”



2.2 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 MATLAB’s `rand()` command, 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.

<u>2.3 Reconstruction of a Voweled Fugue</u>

You'll remember from Lab 3 that MATLAB allows for structures, which group information together. A structure, which is a data type, is used to represent information about something more complicated than what can be held by a single number, character, or boolean or an array of any one of them. For example, a Student can be defined by his or her name (an array of characters), GPA (a double), age (an integer), UFID (a long integer), and more. Each of these pieces of information can be labeled with an easily understood descriptive title, and then combined to form a whole (the structure).

Structures give us a way to "combine" multiple types of information under a single variable. The nice thing about structures is that they allow us to use human-readable descriptions for our data. In Lab 3, every note in a melody was characterized as a list of notes, each one of which starts at a specified pulse, and is played for a specified duration. We could define three separate arrays for one melody, stored as part of the structure `melody`:

```
      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 for that melody:

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

The vowels array is an array of characters representing the vowels to be played at each note, two characters at a time. It'll actually be stored in this form:

```
      melody.vowels         = ['eheeahohooohahee'];
```

The vowel matching a particular note n, will be specified by `melody.vowels((2*n-1):(2*n))`.

MATLAB has a few functions, such as the `strcmp()` function, that can be used to determine the current vowel.

Load the `barukh_fugue.mat` file, containing the `theVoices` structure. You'll notice that it now has four fields: `noteNumbers`, `durations`, `startPulses`, and `vowels`.

Play the correctly-voweled, correctly-timed, all-three-voices-added-together version of the Barukh Fugue. Submit this as a .wav file.

<u>2.4 Extra Credit – Synthesize a Musical Piece of your own!</u>

For up to twenty points of extra credit, out of 100 for the entire lab, synthesize a piece of voweled music into a structure of `theVoices` form, calling it `theVoices_yourname`. Save it as `yourname_fugue.mat`. Submit the *.mat* file, your *.m* code, your PDF explaining it, and the *.wav* of your musical composition labeled *yourname_extracredit.wav.* A simple composition might get 5 points. A more complex one might get 10. A really well-done one might earn the maximum 20 points. Again, be creative!