Laboratory # 9

Filter Design, Modeling, and the z-Plane

9.1 Introduction

So far, we've been considering filters as systems that we design and then apply to signals to achieve a desired affect. However, filtering is 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 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. We will see that such filters are very useful for modeling speech signals and improving the results of our vowel classifier from Lab #8.

In this lab, we will consider a graphical filter design method that allows us to attempt to match a desired frequency response. This method is called *pole-zero placement design*, and it 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.

9.1.1 "The Question"

- How can we *design* filters for certain purposes?

- How can we model vowel production using discrete-time filters?

9.2 Background

9.2.1 Filters and the z-transform

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

$$y[n] = x[n] * h[n] = \sum_k h[k]x[n-k] = \sum_k x[k]h[n-k] \ , \tag{9.1}$$

where $x[n]$ is the input signal, $y[n]$ is the output signal and $h[n]$ is the filter impulse response. Using the z-transform techniques described in Chapter 7 of the textbook, we can also describe the input/output relationship in the z-domain as

$$Y(z) = H(z)X(z) \ , \tag{9.2}$$

where $X(z)$ is the z-transform of $x[n]$, which is the complex-valued function, defined on the complex plane[1] by

$$X(z) = \sum_n x[n]z^{-n} \ , \tag{9.3}$$

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 any of the following equivalent definitions:

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

$$H(z) = \sum_n h[n]z^{-n} \ . \tag{9.4}$$

2. When the filter input is the complex-valued signal $x[n] = z^n$, then the output $y[n]$ is the input z^n multiplied by a complex number, and this complex number is $H(z)$.

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

$$H(z) = \frac{Y(z)}{X(z)} \ . \tag{9.5}$$

The second bullet above is a generalization of the fact that when the input is the complex exponential $x[n] = e^{j\hat{\omega}n} = (e^{j\hat{\omega}})^n$, then the output is that same exponential, multiplied by a complex number, namely, the frequency response function evaluated at frequency $\hat{\omega}$, i.e. $y[n] = \mathcal{H}(\hat{\omega})e^{j\hat{\omega}n}$. In the more general case, the input is the more general complex exponential z^n, where z is an arbitrary complex number, and the output is z^n times a complex number, namely, the system function evaluated at z, i.e. $y[n] = H(z)z^n$. If $z = e^{j\hat{\omega}}$, then the general case reduces to the original case. From this, we see the relationship between the frequency response function $\mathcal{H}$ and the system function H:

$$\mathcal{H}(\hat{\omega}) = H(e^{j\hat{\omega}}) \ . \tag{9.6}$$

More generally, z might be something like $re^{j\theta}$, in which case $z^n = r^n e^{j\theta n}$, which is a complex exponential $e^{j\theta n}$, multiplied by real-valued exponential r^n, which grows exponentially if $|r| > 1$ and shrinks exponentially if $|r| < 1$.

[1] The *complex plane* is simply the set of all complex numbers.

9.2.2 FIR Filters and the Z-transform

For a causal FIR filter one can easily determine the system function using any of the three equivalent definitions given above. However, let us highlight the third method, as it will be useful in the next subsection where the first two are very difficult. In particular, for a causal FIR filter with coefficients $\{b_0, \ldots, b_M\}$, the general time-domain input-output relationship for a causal FIR filter is given by the difference equation

$$y[n] = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + \cdots + b_M x[n-M] \ . \tag{9.7}$$

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

$$\begin{aligned} Y(z) &= b_0 X(z) + b_1 X(z) z^{-1} + b_2 X(z) z^{-2} + \cdots + b_M X(z) z^{-M} \\ &= X(z)(b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + \cdots + b_M z^{-M}) \ , \end{aligned} \tag{9.8}$$

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

$$H(z) = \frac{Y(z)}{X(z)} = b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + \cdots + b_M z^{-M}. \tag{9.9}$$

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

$$H(z) = K(1 - r_1 z^{-1})(1 - r_2 z^{-1})(1 - r_3 z^{-1}) \cdots (1 - r_M z^{-1}) \ , \tag{9.10}$$

where K is a real number, often 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 $H(r) = 0$. (The Fundamental Theorem of Algebra guarantees that $H(z)$ factors in this way.) We assume that the filter coefficients b_k are real. Thus, the zeros may be real or complex, and if one is complex, then its complex conjugate is also a zero.

The very important point to observe now is that the system function $H(z)$ of a causal FIR filter is completely determined by its gain and its zeros. Therefore, we can think of $\{K, r_1, \ldots, r_M\}$ as one more way to describe a filter[2]. 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 zeros. The procedure for this will be described below. Once the gain and zeros are picked, one may find the system function from equation (9.10), and then find its coefficients by inspection of the system function. For example, the number multiplying z^{-3} in the system function is the filter coefficient b_3.

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 close to the unit circle, i.e. $|r| \approx 1$, then $\mathcal{H}(\hat{\omega}) \approx 0$ for all $\hat{\omega}$ near $\angle r$. If the root is located right on the unit circle, i.e. $|r| = 1$, then $\mathcal{H}(\angle r) = 0$.

[2]Previous ways of describing a filter have included the filter coefficients, the impulse response sequence, the frequency response function, and the system function.

This follows from the fact that if $\hat{\omega} \approx \angle r$, then $e^{j\hat{\omega}} \approx r$, and so

$$\mathcal{H}(\hat{\omega}) = H(e^{j\hat{\omega}}) \approx H(r) = 0 \ . \tag{9.11}$$

From this fact, we see that we see that we can make a filter block particular frequencies, i.e. make the magnitude of the frequency response small at these frequencies, simply by placing zeros on or near the unit circle at angles whose values are near these frequencies. 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.

Note that by the Fundamental Theorem of Algebra, a zero r must either be real, or it's complex conjugate r^* must also be a zero of the system function. That is, non-real zeros will always come in complex conjugate pairs.

9.2.3 IIR filters and rational system functions

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

$$\begin{aligned} y[n] \quad = b_0 x[n] \quad & + b_1 x[n-1] + b_2 x[n-2] + \cdots + b_M x[n-M] \\ & + a_1 y[n-1] + a_2 y[n-2] + \cdots + a_N y[n-N] \ . \end{aligned} \tag{9.12}$$

Here, we have the usual FIR filter coefficients, b_k, but we also have another set of coefficients a_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:

$$y[n] = x[n] + \frac{1}{2} y[n-1] \tag{9.13}$$

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

$$h[n] = \left(\frac{1}{2}\right)^n \ , \quad n \geq 0, \tag{9.14}$$

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

$$y[n] - a_1 y[n-1] - \cdots - a_N y[n-N] = b_0 x[n] + b_1 x[n-1] + \cdots + b_M x[n-M] \quad (9.15)$$

$$Y(z) - a_1 Y(z) z^{-1} - \cdots - a_N Y(z) z^{-N} = b_0 X(z) + b_1 X(z) z^{-1} + \cdots + b_M X(z) z^{-M} \quad (9.16)$$

$$Y(z)(1 - a_1 z^{-1} - \cdots - a_N z^{-N}) = X(z)(b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}) \quad (9.17)$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}}{1 - a_1 z^{-1} - \cdots - a_N z^{-N}} \quad (9.18)$$

Equation (9.18) shows the general form of the system function of an IIR filters. Since it is the ratio of two polynomials, it is called a *rational function*[3].

Just as we could factor the polynomial in equation (9.9), we can do the same with equation (9.18) to yield

$$H(z) = K \frac{(1 - r_1 z^{-1})(1 - r_2 z^{-1})(1 - r_3 z^{-1}) \cdots (1 - r_M z^{-1})}{(1 - p_1 z^{-1})(1 - p_2 z^{-2})(1 - p_3 z^{-1}) \cdots (1 - p_N z^{-1})}. \quad (9.19)$$

The roots of the polynomial in the numerator, $\{r_1, \ldots, r_M\}$, are again called the *zeros* of the system function. The roots of the polynomial in the denominator, $\{p_1, \ldots, p_N\}$ 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[4]. 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 be large at frequencies near the angles of these poles. With FIR filters we could directly design filters to have "nulls" at desired frequencies. Now, with IIR filters, we can design peaks in the frequency response, as well as nulls.

Poles and zeros at the origin

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 the system function for our FIR filter given by (9.9). 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.

[3] This is a generalization of the terminology that the ratio of two integers is called a *rational number*.

[4] Technically, because of a division by zero, $H(z)$ is undefined at the location of a pole. However, the magnitude of the system function becomes very large in the neighborhood of a pole.

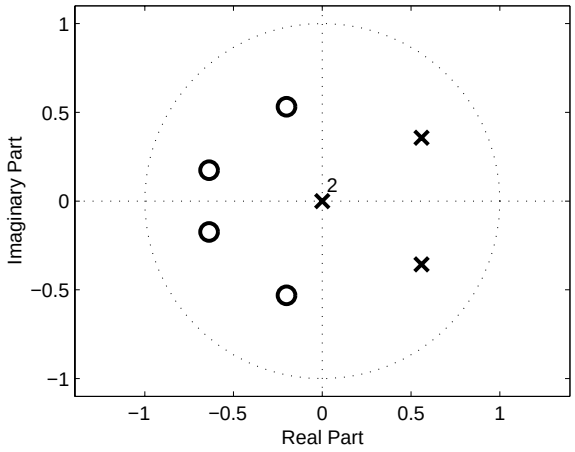

Figure 9.1: A pole-zero plot of an IIR filter.

To reveal such "hidden" poles and zeros, we express the system function in terms of positive powers of z. To do so, we multiply by z^M/z^M, which yields

$$H(z) = \frac{b_0 z^M + b_1 z^{M-1} + b_2 z^{M-2} + b_3 z^{M-3} + \cdots + b_M}{z^M}. \tag{9.20}$$

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. It is mathematically important to account for poles and zeros at the origin. These poles and zeros, though, *only* affect the phase (and thus the delay or time shift) of a system; they do *not* affect the system's magnitude frequency response. Because of this, we will generally not count poles and zeros at the origin towards the total number of poles and zeros in a system.

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*. Figure 9.1 shows an example of a 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. Figure 9.1, for instance, shows four zeros (two conjugate pairs), two "trivial" poles at the origin, and one other conjugate pair of poles. Recall that zeros and poles near the unit circle can be expected to have a strong influence on the magnitude frequency response of the filter.

9.2.4 Graphical interpretation of the system function

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 z. Figure

9.2 shows an example of just 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, $|\mathcal{H}(\hat{\omega})|$, which is shown to the right of the surface.

On Figure 9.2, 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. (Since the two poles in this figure are at the origin, they have no effect on the system's magnitude frequency response.) Thus, the magnitude frequency response has higher gain at points far away from the zeros.

Figure 9.3 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.)

Figure 9.4 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.

9.2.5 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 the other hand, this can have some undesirable side effects. The introduction of poles means that the z-transform may not be defined at certain points because the z-transform sum does not converge, i.e. is not well defined. This leads to the introduction of so-called *regions of convergence* on the z-plane, i.e. regions where the z-transform sum converges to a well defined value. We will not consider regions of convergence here, however.

One somewhat significant problem is introduced if we have a pole outside the unit circle. Consider the following filter, for instance:

$$y[n] = x[n] + 2y[n-1] \tag{9.21}$$

This filter has a single pole at $z = 2$. What is this system's impulse response? 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 *unstable*[5]. Unstable filters cause

[5]Technically, it is *bounded input, bounded output* (BIBO) unstable because the input has a limited magnitude but the output does not.

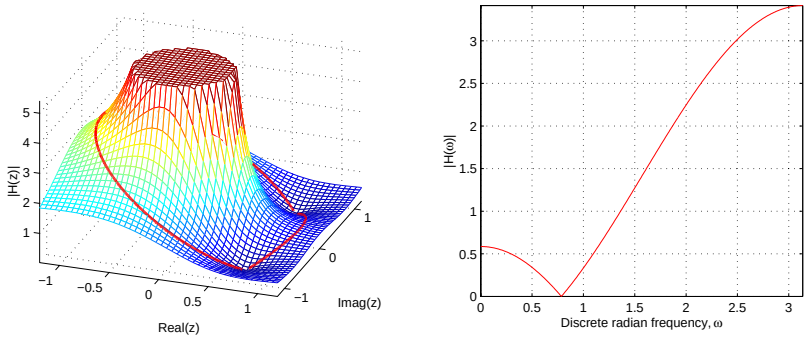

Figure 9.2: (Left) The z-plane surface defined by the system function $H(z) = (1 - e^{j\pi/4}z^{-1})(1 - e^{-j\pi/4}z^{-1})$. Note that this system has two poles at the origin. (Right) The corresponding magnitude frequency response.

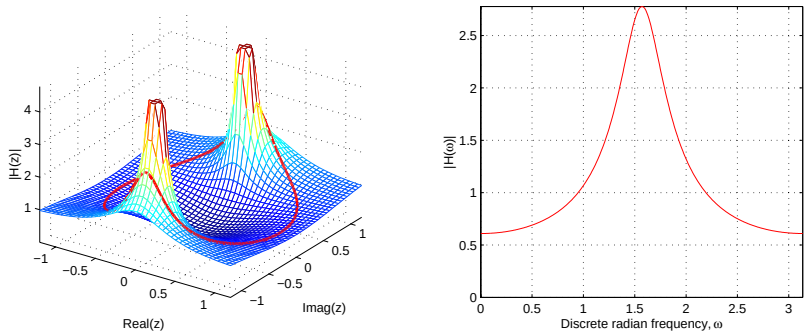

Figure 9.3: (Left) The z-plane surface defined by the system function $H(z) = \frac{1}{(1-0.8e^{j\pi/2}z^{-1})(1-0.8e^{-j\pi/2}z^{-1})}$. Note that this system has two zeros at the origin. (Right) The corresponding magnitude frequency response.

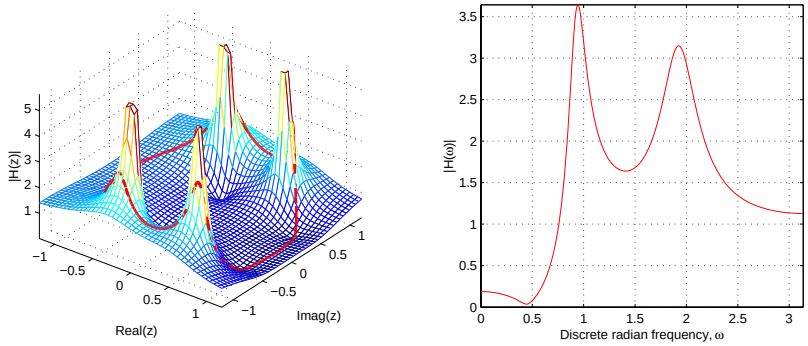

Figure 9.4: (Left) The z-plane surface for a complicated system function with four poles and four zeros. (Right) The corresponding magnitude frequency response.

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.

9.2.6 Filter design using pole-zero placement

In this laboratory, 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 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 the textbook in Chapter 7) 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 filter 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 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.

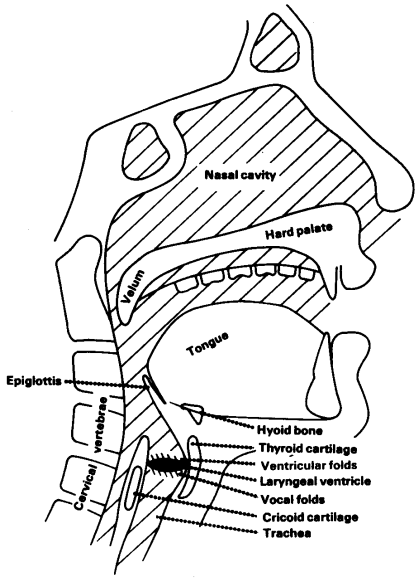

Figure 9.5: A diagram showing the larynx and vocal tract.

9.2.7 Modeling Vowel Production

In Lab #8, we discussed some of the properties of vowel production in speech, but we did not examine the mechanisms behind 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*. Figure 9.5 shows a diagram of the larynx and vocal tract. When we speak a vowel, the lungs push a stream of air through larynx and the *vocal chords*. Given the appropriate muscular tension, this stream of air causes the vocal cords to vibrate[6]. This in turn creates a nearly periodic fluctuation in air pressure passing through the larynx. The fundamental frequency of vocal chord 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 and 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; from Lab #8, we know these frequencies as *formants*. When whispering a vowel, the lungs push air through the larynx, but the vocal chords do not vibrate. In this case, the air pressure fluctuation is quite noise-like and generally is not periodic. Nevertheless, the tongue, lips and jaw shape the vocal tract just as before to make the various vowel sounds.

Note that the above description is only accurate for vowels and so-called *voiced consonants* like "m" and "n." Most consonant sounds are produced using the tongue, lips, and

[6]In fact, during normal production the vocal cords open and close completely on each cycle of the vibration.

Figure 9.6: A block diagram of the source-filter model of speech production.

teeth rather than the vocal cords. We will not consider consonants in this lab.

It is traditional to model speech production using a *source-filter model*. Figure 9.6 shows a block diagram of the source-filter model. The first block is the *glottal source*, which takes as input a fundamental frequency and produces a periodic signal (the *glottal source signal*) with the given fundamental frequency. The signal produced is typically modeled as a periodic pulse train. To a first approximation, we can assume that spectrum of this pulse train is composed of equal amplitude harmonics. The glottal source signal is meant to be analogous to the signal formed by the air pressure fluctuations produced by the vibrating vocal cords. Note that to model whispering, the glottal source signal can be modeled using random noise rather than a pulse train.

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. Typically, such a filter can have relatively low order (i.e., approximately 10-20 coefficients). Further, the acoustics of the vocal tract suggest that this filter should be IIR. Often, the vocal tract is modeled using an *all-pole filter* which has no zeros. This is because an acoustic passageway like the vocal tract primarily affects a sound through *resonances*. A resonance is a part of a system that tends to vibrate at a certain *resonant frequency*, thus amplifying that frequency in signals passed through them. The feedback form of an IIR filter is a direct implementation of resonance; this is how IIR filters are able to produce high gain at certain frequencies. Using this simple model of speech production, it is possible to synthesize artificial vowels.

All-pole analysis and vowel classification

In the last lab, we explored some features for vowel classification that were based on two measures of spectral energy in a vowel signal. The development of the source-filter model, however, suggests an acoustically motivated feature for vowel classification. If we assume that the vocal tract can be modeled with a low-order discrete-time filter, then the vocal tract filter captures all of the relevant information about which vowel has been produced. Variations such as fundamental frequency and type of vowel production (i.e., voiced or whispered) are restricted to the glottal source and can be neglected. Using samples of the frequency response of the vocal tract filter as features for vowel classification has been shown to produce good classification results.

Fortunately, there are nice mathematical tools for deriving all-pole filter models automatically from a time-domain waveform. These tools, fit the spectrum of a time-domain signal with poles in a least-squares sense. Note that these tools work directly with the time-domain waveform rather than its spectrum; typically, they return the resulting a_k feedback coefficients for a filter with those poles, rather than the locations of the poles themselves. We will explore these tools for all-pole analysis in the laboratory assignment, and we will compare classification performance using features based on these models to the performance we achieved with our other feature sets from Lab #8.

9.3 Some MATLAB commands for this lab

- ***Pole-Zero Place*:** In this laboratory, we will primarily be exploring filter design using pole-zero placement. To help us do this, we will be using a MATLAB graphical user interface program called *Pole-Zero Place*. This program was designed to run using Windows systems running MATLAB 6.0; it will not work with previous versions of MATLAB. To run this program you need to download two different files: `pole_zero_place.m` and `pole_zero_place.fig`. To execute *Pole-Zero Place*, execute the command

  ```
  >> pole_zero_place(gains_match,fund_frq,B,A);
  ```

  Here, `gains_match` is a vector of gains at harmonically related frequencies for a desired transfer function and `fund_frq` is the fundamental frequency. The optional third and fourth parameters allow you to specify the feedforward and feedback filter coefficients (`B` and `A`), so that the GUI will start with an initial filter configuration.

  Once the program starts, the GUI window will appear. Hopefully you will find the interface to be intuitive. 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 and a "desired" frequency response. The interface works by letting you add poles and zeros to the z-plane and move them about, all the while showing the effect of the poles and zeros on the frequency response.

  Toggling the *Add Pole* and *Add Zero* buttons allows you to "drop" a conjugate pair of poles or zeros onto the z-plane. You can click-and-drag poles and zeroes to move them around the z-plane, or you can use the arrow keys for fine adjustment of their position. The magnitude and angle of the selected pair of poles or zeros is shown in a pair of edit boxes at the right of the figure window; you can move the currently selected poles or zeros to a desired location by entering the desired magnitude and angle in these edit boxes. The *Delete Poles/Zeros* button at the upper left side of the window allows you to delete the currently selected pair of poles or zeros. Note that the GUI always keeps an equal number of poles and zeros on the z-plane at one time; thus, when you delete a set of poles or zeros, they will be moved to the origin (where they have no effect on the system's frequency response). Note that poles and zeros at the origin are not counted towards the total number of poles and zeros in the system.

  The axis in the lower left shows the magnitude frequency response of the resulting system in blue. This curve will change as you move poles and zeros about the axes in the upper left. The red curve/stem plot in the axes in the lower left of the window shows the desired frequency response that we wish to match. The GUI assumes a sampling frequency $f_s = 8192$ Hz, which is why the x-axis of this plot extends from 0 to 4096 Hz. To the right of these axes are two boxes labeled *Filter Matching Error*. These boxes indicate how closely your filter matches the desired frequency response. The matching error values are computed as the RMS error between the desired frequency response and your filter design in both linear amplitude and in decibels. You can change the overall gain of your filter by dragging the blue frequency response curve or by adjusting the *Filter Gain* edit box in the upper right of the window. Note that there is a set of radio buttons that switch this plot between a linear magnitude and a decibel display.

Once you have a filter design that you are happy with, you can export the filter coefficients to the MATLAB workspace using the *Export Filter Coefs* button. This will store the filter coefficients in the base workspace as the variables `B_pz` and `A_pz`. You can also copy the figure window to the Windows clipboard using the *Copy to Clipboard* button (the preferred method), or print it using the *Print GUI* button. Finally, you can use the *Play Sound* button to "listen" to the filter's response to a glottal-source signal. This is interesting when trying to match the spectrum of a vowel signal.

- ***Pole-Zero Place 3-D*:** It is very useful to be able to visualize the $|H(z)|$ surface given a pole-zero plot. To this end, we provide a modified version of *Pole-Zero Place* called *Pole-Zero Place 3-D*. In this version of the GUI, there is an additional plot containing a mesh plot of the $|H(z)|$ surface. The unit circle is plotted on the surface plot to show where the magnitude frequency response of the system exists on the z-plane. Note that this version of the GUI is considerably slower than the original. Thus, you should use this GUI to get a sense of the effect of poles and zeros on the $|H(z)|$ surface, but use the original for filter design.

- **Calculating the frequency response of IIR Filters:** 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`[7], we compute the frequency response at 256 points using the command:

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

  Again, `H` contains the frequency response and `w` contains the corresponding discrete-time frequencies. Alternately, we can use the command

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

  To return the frequency response for the frequencies $\pi/4$, $\pi/2$, and $3\pi/4$.

- **Converting between filter coefficients and zeros-poles-gain:** 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, while the system zeros are simply the roots of the denominator polynomial. To convert back, use the commands

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

[7] MATLAB's convention for a_k coefficients is that `A(1)`$= 1$ and `A(k+1)` $= -a_k$.

- **Generating pole-zero plots:** Occasionally, we'd like to use MATLAB to make a pole-zero plot from a filter. If our filter is defined by feedforward coefficients `B` and feedback coefficients `A` (both row vectors), we can generate a pole-zero plot using the command:

  ```
  >> zplane(B,A);
  ```

  Alternately, if we have a list of poles, `p`, and a list of zeros, `z`, (both column vectors) we can use the following command:

  ```
  >> zplane(z,p);
  ```

- **Automatic all-pole modeling:** Using the MATLAB command `aryule`, we can compute an all-pole filter model for a time-domain waveform. If our time-domain waveform is given by `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 to 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);
  ```

9.4 Demonstrations in the Lab Section

- The z-transform, system functions, and IIR filters
- The transfer function "surface," $|H(z)|$
- Using *Pole-Zero Place* for filter design
- Vocal tract modeling

9.5 Laboratory Assignment

1. (Fit an FIR filter frequency response.) Download the files `pole_zero_place.m`, `pole_zero_place.fig`, and `lab9_data.mat`. In this problem, you will get familiar with the *Pole-Zero Place* program for filter design using pole-zero placement.

   In `lab9_data`, the variable `FIR_fr` contains samples of the frequency response for a simple FIR filter with order six. (That is, the filter six zeros.) Execute *Pole-Zero Place* using the command

   ```
   >> pole_zero_place(FIR_fr,1);
   ```

   Use the GUI to find an FIR filter with six zeros[8] that matches the frequency response of the original filter. You should be able to get the linear matching error to be less than 0.1. (Hint: The original filter had all six of its zeros inside the unit circle, so yours should as well.)

[8] The GUI will automatically place poles at the origin, but you should not count these towards your total number of poles and zeros.

- [6] Include the GUI window with your matching filter in your report. (Note: The easiest way to do this is to use the *Copy to Clipboard* button on a Windows machine. After hitting the button, wait for the GUI to flash white and then paste the result into your report.)
- [2] What are the filter coefficients b_k and a_k for your filter?
- [2] Where are the zeros on the z-plane? Give your answers in rectangular form.

2. (Design a lowpass filter.) In the previous problem, you could have theoretically achieved zero error if you put in the "true" values for the filter's zeros. However, this is generally not possible when designing filters in the real-world. In this problem, we'll use pole-zero placement to design a lowpass filter. The frequency response that you'll be matching has a transition which is sharper than we can achieve with a low order filter model. However, we will accept some error in the filter design.

   For the parts of this problem, use the command

   ```
   >> pole_zero_place(lpf,100);
   ```

   to start *Pole Zero Place* with target frequency response `lpf`, which contains the samples of the frequency response for an ideal lowpass filter. Note that the variable `lpf` is found in `lab9_data.mat`.

   (a) (Minimize linear error using 10 zeros.) Once you have started the GUI, make sure that the frequency response is in "linear plot" mode (rather than "decibel plot" mode). Find a filter with 10 zeros and no poles (except those automatically placed at the origin) that produces a linear matching error which is as small as possible. You should be able to achieve a linear error less than 0.15.

   - [8] Include the GUI window with your matching filter in your report.
   - [2] What are the filter coefficients b_k and a_k for your filter?
   - [2] Where are the zeros on the z-plane? Give your answers in rectangular form.

   (b) (Minimize decibel error using 10 zeros.) Restart the GUI, and put the program into "decibel plot" mode. Find another filter with 10 zeros and no poles (except those automatically placed at the origin) that makes the decibel matching error which is as small as possible. You should be able to achieve a decibel error less than 7.

   - [8] Include the GUI window with your matching filter in your report.
   - [2] What are the filter coefficients b_k and a_k for your filter?
   - [2] Where are the zeros on the z-plane? Give your answers in rectangular form.

   (c) (Minimize linear error for a general IIR filter.) Restart the GUI again. Once again, put the program into "linear plot" mode. Now, find a general IIR filter with a total of ten poles *or* zeros that makes the linear matching error as small as possible. (That is, the number of poles plus the number of zeros should add up to ten. Again, don't count poles or zeros at the origin towards your total.) You should be able to achieve a linear matching error of less than 0.1. (If you can get a linear matching error less than 0.05, you will get +2 bonus points. It is possible to get an error less than 0.03.)

- [8+2] Include the GUI window with your matching filter in your report.
- [2] What are the filter coefficients b_k and a_k for your filter?
- [2] Where are the poles and zeros on the z-plane? Give your answers in rectangular form.

3. (Matching a vowel signal's spectrum using pole-zero placement.) `lab9_data.mat` contains the variable `vowel2`, which is a short vowel signal sampled at 8192 Hz. In this problem we will find a filter model of the vocal tract used to produce this vowel.

   (a) First, let's examine the signal itself.

   - [2] Use `soundsc` to listen to `vowel2`. What vowel does this signal represent?
   - [4] Plot the magnitude DFT coefficients for this signal in decibels. Include only the first half of the DFT coefficients on your plot, and make sure that the x-axis is displayed in Hz rather than the DFT coefficient number. (Hint: The maximum value on your x-axis should be 4096 Hz.)
   - [2] From this plot, estimate the fundamental frequency of the vowel signal.

   (b) `lab9_data.mat` contains the variables `vowel2_amps`, which contains the amplitudes of the harmonics that make up `vowel2`. We'll use the amplitudes to find a vocal tract filter for this vowel. Start *Pole-Zero Place* using the command

   ```
   >> pole_zero_place(vowel2_amps,frq);
   ```

   where `frq` is the fundamental frequency that you estimated. (Hint: The red stems should go all the way across the frequency response plot. Also, there should be one stem for each element of `vowel2_amps`. If not, you have probably estimated your fundamental frequency incorrectly.)

   Set the GUI to "decibel plot" and find a filter with six poles and six zeros that makes the decibel matching error as small as possible. You should be able to get the decibel error below 4.2. (It is possible to achieve a decibel error of 3.65.)

   - [8] Include the GUI window with your matching filter in your report.
   - [2] What are the filter coefficients b_k and a_k for your filter?
   - [2] Where are the poles and zeros on the z-plane? Give your answers in rectangular form.

   (c) Now, repeat the above for a filter with 10 poles and no zeros (except for those at the origin). You should be able to achieve a decibel matching error below 2.6. (It is possible to achieve a decibel error of 2.1.)

   - [8] Include the GUI window with your matching filter in your report.
   - [2] What are the filter coefficients b_k and a_k for your filter?
   - [2] Where are the poles on the z-plane? Give your answers in rectangular form.

   If you are working on a computer with audio capability, you should use the *Play Sound* button to listen to a synthesis of the vowel signal. Note that the all-pole model produces less error with fewer total coefficients. This suggests that all-pole filters are more appropriate for vocal tract modeling.

4. (All-pole Modeling and Classification) In this problem, we'll look at an automatically generated all-pole model of `vowel2`, and we'll see how we can use this model to help us improve our vowel classifier performance from Lab #8.

   (a) Use `aryule` to compute an all-pole model for `vowel2` with 10 poles.

   - [2] What are the resulting feedback coefficients?
   - [2] Make a pole-zero plot of the resulting filter.
   - [4] Use `freqz` to plot the frequency response of this filter and the frequency response of the filter you found in Problem 3c in two subplots of the same figure. Display the two frequency responses in decibels. (Note: The two frequency responses in decibels may be offset by some constant, which corresponds to a scaling of the original spectrum.)

   (b) Here, we'll consider 16 samples of the frequency response of an all-pole filter to be a potential feature vector for vowel classification. `lab9_data.mat` contains five matrices which contain all-pole feature vectors for the same vowel instances we examined in Lab #8. They are `oo_ap`, `oh_ap`, `ah_ap`, `ae_ap`, and `ee_ap`.

   - [5] As you did in Lab #8, compute the mean all-pole feature vectors for each of the five vowel classes. Make sure you label which mean vector belongs with which class.
   - [5] Combine the above vectors into a matrix of mean feature vectors. Then, use `confusion_matrix.m` (from Lab #8) to calculate the confusion matrix for the all-pole features. As you did in Lab #8, use the vowel order "oo," "oh," "ah," "ae," and "ee". (Note: if you were unsuccessful at completing `confusion_matrix` in Lab #8, you can use the compiled function `confusion_matrix_demo.dll` for this problem.)
   - [4] Compare this confusion matrix with the two confusion matrices that you computed in Lab #8. Is the performance of the classifier better with this feature class? Note any similarities between the three confusion matrices.

5. On the front page of your report, please provide an estimate of the average amount of time spent outside of lab by each member of the group.

