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 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. This in turn will suggest a way that we might improve the performance of the vowel classifier we developed in Lab 8 using an automatic modeling method.

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. We will use this design methodology both to design a traditional "goal-oriented" lowpass filter, and to do some filter modeling.

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 causal filter[1] 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 *DSP First*, 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[2] 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 one of the following equivalent definitions:

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

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

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

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

We can derive this result from equation 9.4. If we let $z = e^{j\hat{\omega}}$, then we know that $H(z) = H(e^{j\hat{\omega}}) = \sum_n h[n]e^{-j\hat{\omega}n}$. This is simply the definition of a system's frequency given its impulse response $h[n]$.

[1] Note that in this lab, we will only be concerned with causal filters.

[2] The *complex plane* is simply the set of all complex numbers. The real part of the complex number is indicated by the x-axis, while the imaginary part is indicated by the y-axis.

9.2.2 FIR Filters and the z-transform

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 $\{b_0, \ldots, b_M\}$, 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[3]

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

where K is a real number 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.

The very important point to observe now from equation (9.10) 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[4]. 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 equation (9.10), 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. The specific procedure will be described shortly.

The fact that we may design the frequency response of a causal FIR filter by choosing its zeros[5] stems from the following principle:

> If a filter has a zero r located on the unit circle, i.e. $|r| = 1$, then $\mathcal{H}(\angle r) = 0$, 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 $\mathcal{H}(\angle r) \approx 0$, i.e. the frequency response has a *dip* at frequency $\angle r$. In either case, $\mathcal{H}(\hat{\omega}) \approx 0$, when $\hat{\omega} \approx \angle r$.

[3] The Fundamental Theorem of Algebra guarantees that $H(z)$ factors in this way.

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

[5] The gain does not affect the shape of the frequency response.

The above fact follows from the property that if $\hat{\omega} = \angle r$ and $|r| = 1$, then $e^{j\hat{\omega}} = r$, and so

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

A similar statement shows $\mathcal{H}(\hat{\omega}) \approx 0$ when $|r| \approx 1$ and/or $\hat{\omega} \approx \angle r$.

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[6]. 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 $\hat{\omega}_1, ..., \hat{\omega}_L$ at which the frequency response should contain a null or a dip.
2. Choose zeros $r_i = \rho_i e^{j\hat{\omega}_i}$, $i = 1, \ldots, L$, with $\rho_i = 1$ or $\rho_i \approx 1$, depending upon whether a null or a dip is desired at frequency $\hat{\omega}$. For each $\hat{\omega}_i \neq 0$ choose also a zero r_j that is the complex conjugate of r_i. Let M be the total number of zeros chosen.
3. Form the system function $H(z) = K(1 - r_1 z^{-1}) \times \cdots \times (1 - r_M z^{-1})$, where K is a gain that we also choose.
4. Cross multiply the factors of $H(z)$ found in the previous step so as to express $H(z)$ as a polynomial whose terms are powers of z^{-1}.
5. Identify the FIR filter coefficients $\{b_0, \ldots, b_M\}$, which are simply the coefficients of the polynomial found in the previous step, as shown in equation (9.9).

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*[7]. 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[8] that $y[n] = 0$ for $n < 0$, one can straightforwardly show that the impulse response is

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

[6] Since non-real zeros must occur in conjugate pairs, we must also place a conjugate zero on the unit circle, i.e. a zero whose angle is the negative of the first.

[7] 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.

[8] This assumption is one of the "initial rest conditions" discussed in chapter 8 of *DSP First*.

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.

$$y[n] - a_1 y[n-1] - \cdots - a_N y[n-N] = b_0 x[n] + b_1 x[n-1] + \cdots + b_M x[n-M] \tag{9.15}$$

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

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

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

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*[9].

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

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

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[10]. 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 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 $\hat{\omega}_1, ..., \hat{\omega}_L$ at which the frequency response should contain a null, a dip, or a peak.

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

[10] 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.

2. Choose zeros $r_i = \rho_i e^{j\hat{\omega}_i}$ at those frequencies at which a null or a dip should occur, with $\rho_i = 1$ or $\rho_i \approx 1$, as desired. For each such $\hat{\omega}_i \neq 0$, choose also a zero r_j that is the complex conjugate of r_i. Let M be the total number of zeros chosen.

3. Choose poles $p_i = \rho_i e^{j\hat{\omega}_i}$ at those frequencies at which a peak should occur, with $\rho_i = 1$ or $\rho_i \approx 1$ as desired. For each such $\hat{\omega}_i \neq 0$ choose also a pole p_j that is the complex conjugate of p_i. Let N be the total number of poles chosen.

4. Form the system function $H(z) = K\frac{(1-r_1 z^{-1})\times\cdots\times(1-r_M z^{-1})}{(1-p_1 z^{-1})\times\cdots\times(1-p_N z^{-1})}$, where K is a gain that we also choose.

5. Cross multiply the factors of $H(z)$ found in the previous step and express $H(z)$ as the ratio of two polynomials whose terms are powers of z^{-1}.

6. Identify the IIR filter coefficients $\{a_0, \ldots, a_N, b_0, \ldots, b_M\}$, which are simply the coefficients of the polynomials found in the previous step, as shown in equation (9.18).

Poles and zeros at the origin and at infinity

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 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. 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 $\frac{z^M}{z^M}$, 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.

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[11]. 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[12]. In this laboratory, we will primarily be concerned with *nontrivial poles and zeros* (those not at the origin or at infinity).

[11] A zero at infinity means that $|H(z)| \to 0$ as $|z| \to \infty$. It can be shown that a causal filter can never have a pole at infinity.

[12] Trivial poles and zeros *do* affect the phase (and thus the delay or time shift) of a system.

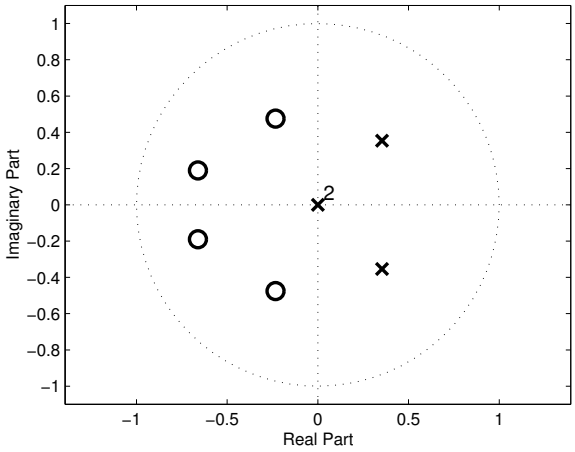

Figure 9.1: A pole-zero plot of an IIR filter.

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 M or 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, i.e. if $r_i = p_j$ for some i and j, then these factors "cancel" each other, i.e. the filter is equivalent to a filter whose system function has neither factor.

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 the complex number 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.

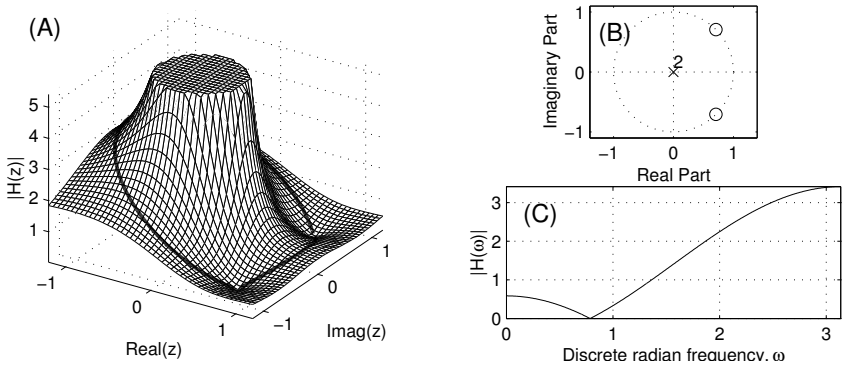

Figure 9.2: (A) The z-plane surface defined by the system function $H(z) = (1 - e^{j\pi/4}z^{-1})(1 - e^{-j\pi/4}z^{-1})$. (B) The corresponding pole-zero plot. (C) The corresponding magnitude frequency response.

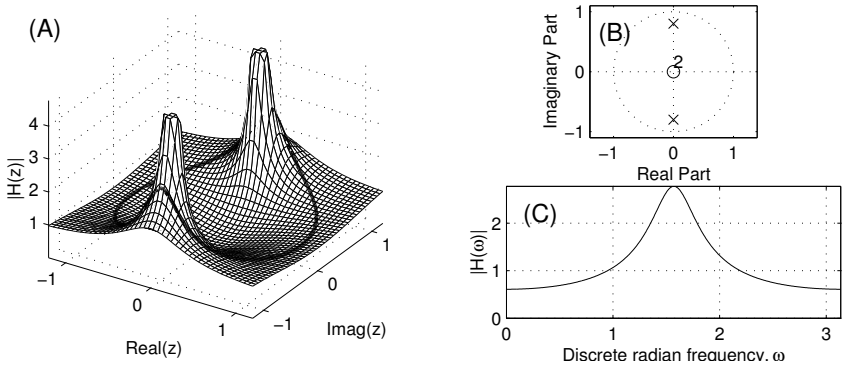

Figure 9.3: (A) The z-plane surface defined by the system function $H(z) = \frac{1}{(1-0.8e^{j\pi/2}z^{-1})(1-0.8e^{-j\pi/2}z^{-1})}$. (B) The corresponding pole-zero plot. (C) The corresponding magnitude frequency response.

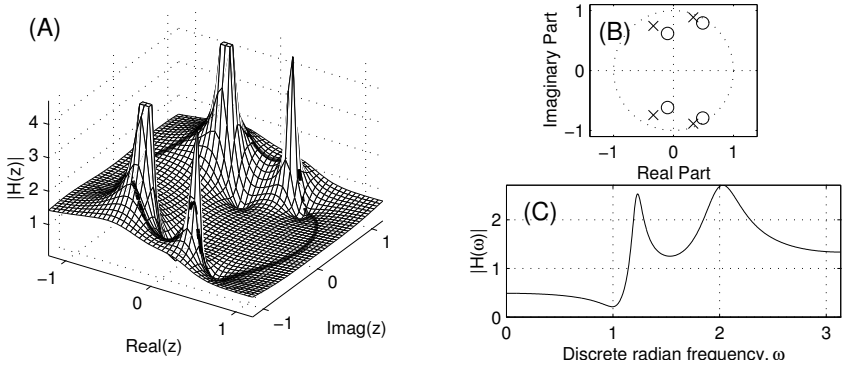

Figure 9.4: (A) The z-plane surface for a complicated system function with four poles and four zeros. (B) The corresponding pole-zero plot. (C) The corresponding magnitude frequency response.

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. 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? 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 *unstable*[13]. 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.

9.2.6 Filter design using manual 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,

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

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 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 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.

Note that there are automatic filter design methods which do not require manual placement of poles and zeros. In Section 9.2.8 we discuss one such method.

9.2.7 Design of Standard Filter Types

Many of the filters that we wish to design and use belong to one of four standard types: lowpass, highpass, bandpass, or bandstop. These filters are characterized by a *passband* (a band of frequencies which are relatively unaltered by the filter) and a *stopband* (a band of frequencies which are significantly *attenuated*, or decreased in amplitude, by the filter). The locations of the passband and stopband are what characterize the different filter types. For instance, a lowpass filter has a passband which contains low frequencies and a stopband which contains high frequencies, while a bandpass filter has a single passband that is surrounded by two stopband regions. Between the passband and the stopband is a *transition*

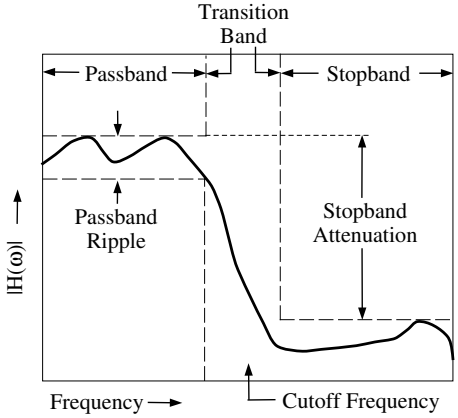

Figure 9.5: An illustration of the various bands of a lowpass filter.

band in which the filter's frequency response changes from high to low. The location of the transition band in frequency determines the filter's *cutoff frequency*. In this lab, we will specify the cutoff frequency by using the two frequencies that bound the transition band.

When designing these types of filter, there are a number of different design goals that we may attempt to achieve. For instance, we may wish to have a very flat frequency response in the *passband* of the filter. Unfortunately, it can be difficult to achieve a flat frequency response over some frequency region. Instead, the frequency response usually varies somewhat over that region; this variation is called *ripple*. Thus, one of design criteria may be to minimize *passband ripple*. In this lab, we define the passband ripple as the (positive) decibel value of the ratio between the maximum and minimum filter gains in the passband.

Another common goal is to try to minimize the gain in the stopband of the filter relative to the gain in the passband. That is, we wish to maximize the *stopband attenuation*. In this lab, we define the stopband attenuation as the decibel ratio between the maximum filter gain in the passband and the maximum filter gain in the stopband.

Figure 9.5 shows the passband, transition band, and stopband for a lowpass filter. The figure also illustrates the the passband ripple and stopband attenuation. In Problem 2 of this assignment, you will design a lowpass filter to maximize stopband attenuation.

9.2.8 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 9.6 shows a diagram of the larynx and vocal tract. When we speak a vowel, the lungs push a stream of air through the larynx and the *vocal folds* (commonly referred to as the *vocal chords*). Given the appropriate muscular tension, this stream of air causes the vocal

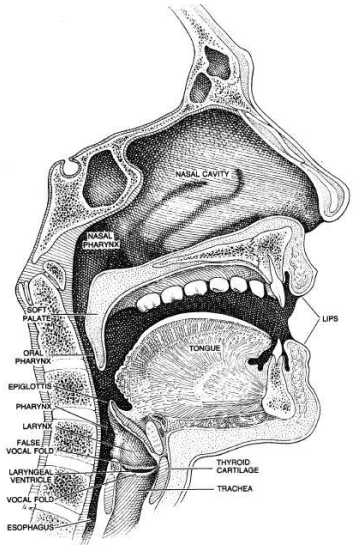

Figure 9.6: A diagram showing the larynx and vocal tract.

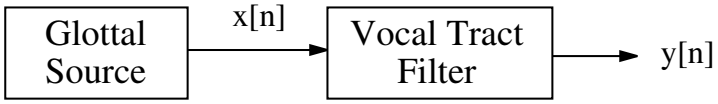

Figure 9.7: A block diagram of the source-filter model of speech production.

folds to vibrate[14]. This in turn creates 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 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 folds 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 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 9.7 shows a block diagram of the source-filter model. The first block is the *glottal source*, which

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

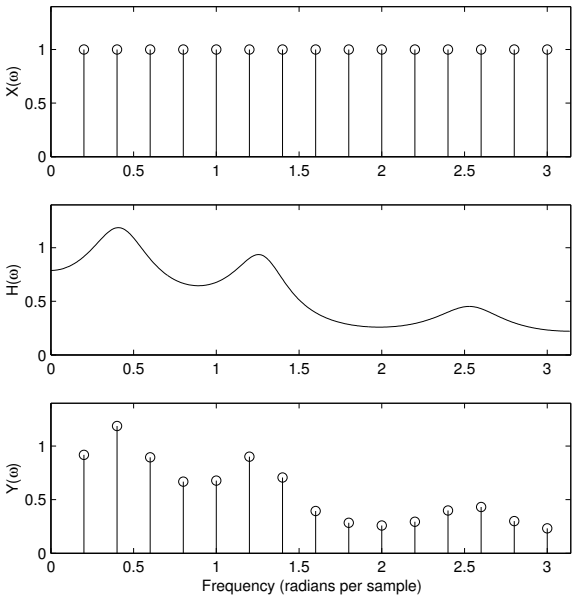

Figure 9.8: A plot of the magnitude spectrum of a glottal source signal, the frequency response of a vocal tract filter, and the magnitude spectrum of the output signal.

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. 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 $\hat{\omega}_0$ 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 $k\hat{\omega}_0$. This is illustrated in Figure 9.8 which shows a particular example. 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.

In Problem 3 of this assignment, 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.

Typically, our vocal tract filter can have relatively few filter coefficients (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 non-

trivial 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

- **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`, we compute the frequency response at 256 points using the command:

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

  As with FIR filters, MATLAB's convention for the b_k coefficients is `B(1)`$= b_0$, `B(2)`$= b_1$, ..., `B(M+1)`$= b_M$. MATLAB's convention for the a_k coefficients is `A(1)`$= 1$, `A(2)`$= -a_1$, ..., `A(N+1)`$= -a_N$. Both b_k and a_k are given as defined in equation 9.18. `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 $\pi/4$, $\pi/2$, and $3\pi/4$.

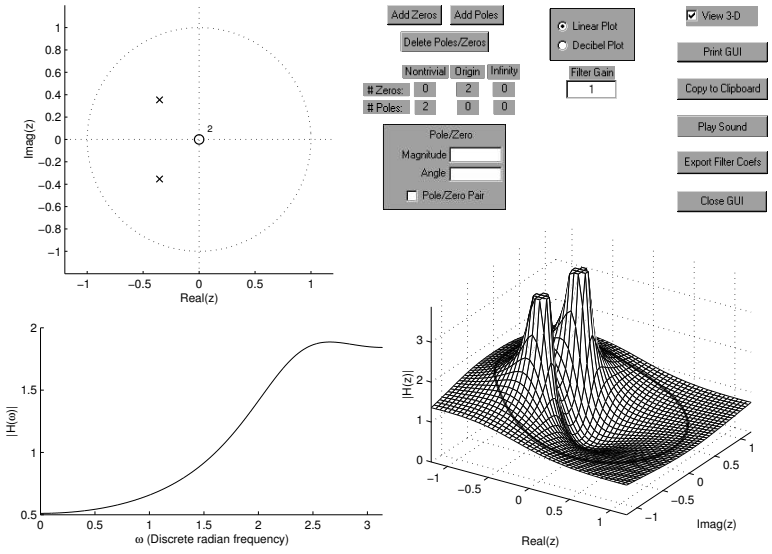

Figure 9.9: The GUI window for *Pole-Zero Place 3-D*.

- ***Pole-Zero Place 3-D*:** 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: `pole_zero_place3d.m` and `pole_zero_place3d.fig`. To begin *Pole-Zero Place 3-D*, simply execute[15] the command

  ```
  >> pole_zero_place3d;
  ```

  Once the program starts, the GUI window shown in Figure 9.9 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 3-D axis which displays a 3-D graph of the $|H(z)|$ surface[16].

  The interface allows you to do a wide variety of things.

  1. To add a 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. The state of the *Place pair* checkbox determines whether a single (real) pole or zero is added, or whether a conjugate pair is added.

[15] This program was designed to run using Windows systems running MATLAB 6 or higher; it will not work with previous versions of MATLAB. It should work with Unix operating systems running MATLAB 6, but this has not been tested.

[16] This surface plot requires significant computation, and thus it can be toggled on and off using the *View 3-D* checkbox in the upper right.

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* edit boxes.
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.
6. To rotate the 3-D $|H(z)|$ plot, simply click-and-drag the axes in the lower right of the GUI. To enable or disable the 3-D plot, toggle the *View 3-D* checkbox in the upper right of the GUI
7. 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(B,A);
   ```

   This is useful if you wish to start continue working on a design that you had previously saved. You may set either of these parameters to empty (`[]`) if you do not wish to specify the filter coefficients.
8. To print the GUI window, you can either use the *Copy to Clipboard* button to copy an image of the figure into the clipboard[17], or you can print the figure using the *Print GUI* button.
9. 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.
10. 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.

- ***Pole-Zero Place 3-D* – Filter Matching Mode:** In "filter matching mode," you specify samples of a desired transfer function at harmonically related frequencies and

[17] Windows operating systems only

try to match that transfer function. The GUI plots a red curve or stem plot along with the frequency response function; this is the response we wish to match. Two edit boxes labeled *Linear Matching Error* and *Decibel Matching Error* 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.

To start the GUI in this mode, use the following command:

```
>> pole_zero_place3d(B,A,filter_gains,fund_frq);
```

`filter_gains` are the values of the desired filter frequency response at harmonically related frequencies, and `fund_frq` is the fundamental frequency (in radians per sample) of the harmonic series at which `filter_gains` are defined.

- ***Pole-Zero Place 3-D* – Lowpass Design Mode:** In "lowpass design mode," you specify the maximum frequency of the passband and the minimum frequency of the stopband (both in radians per sample). To start the GUI in this mode, use the following command:

  ```
  >> pole_zero_place3d(B,A,[pass_max,stop_min]);
  ```

  In this mode, the GUI computes the passband ripple and the stopband attenuation of your lowpass filter design. You can use these measures to evaluate your filter design. The figure in the lower right also displays the passband and stopband of the filter, with appropriate minima and maxima.

- **Converting between filter coefficients and zeros-poles:** 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`. We can see this in equation 9.18. To convert back, use the commands

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

  Note that we loose the filter's gain coefficient, K, with both of these conversions.

- **Generating pole-zero plots:** Frequently, we'd like to use MATLAB to make a pole-zero plot for 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);
```

An example of a pole-zero plot resulting from this command is shown in Figure 9.1.

- **Automatic all-pole modeling:** 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 that in some sense matches the magnitude frequency response of the signal. If signal 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 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 system function "surface," $|H(z)|$
- Using *Pole-Zero Place 3-D* for filter design
- Vocal tract modeling

9.5 Laboratory Assignment

1. (Fit an FIR filter's frequency response.) Download the files `pole_zero_place3d.m`, `pole_zero_place3d.fig`, and `lab9_data.mat`. In this problem, you will get familiar with the *Pole-Zero Place 3-D* 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 six zeros. Execute *Pole-Zero Place 3-D* using the command

   ```
   >> pole_zero_place3d([],[],FIR_fr,2*pi/8192);
   ```

   Use the GUI to find an FIR filter with six nontrivial zeros 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.)

- Include the GUI window with your matching filter in your report. In this and the following problems, make sure that it is possible to read the filter evaluation scores on your printout. (Note: The easiest way to include the GUI window 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.)
- What are the filter coefficients b_k and a_k for your filter?
- Where are the zeros on the z-plane? Give your answers in rectangular form.

2. (Design a lowpass filter.) In this problem, we will use the "lowpass design mode" of *Pole-Zero Place 3-D* to design some lowpass filters, as described in Section 9.2.7. For the various parts of this problem, use the command

```
>> pole_zero_place3d([],[],2*pi*[1500 2000]/8192);
```

This sets the filter transition band to 1500 Hz to 2000 Hz if we assume a sampling rate of 8192 samples per second.

(a) (Design an FIR lowpass filter to maximize stopband attenuation.) First, let's see what we can do with just zeros (that is, with FIR filters). Using only six nontrivial zeros (i.e., three zero pairs), design a lowpass filter with a stopband attenuation of at least 30 dB. (Remember, we want our stopband attenuation to be as large as possible). For now, take note of your filter's passband ripple, but don't worry about minimizing it.

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

Food for thought: Using just zeros, try to find a way to minimize the passband ripple. What does this do to your stopband attenuation? Try this with more zeros, but don't use any poles.

(b) (Design an IIR lowpass filter to maximize stopband attenuation.) There are two primary benefits to the use of IIR filters. First, it is very easy to get very high gain at certain frequencies. This lets us design a lowpass filter with very high stopband attenuation. Using a single pair of nontrivial poles, design a lowpass filter that has a stopband attenuation greater than 60 dB. Use the same transition band as in the previous problem. Again, you should take note of the passband ripple, but don't worry about minimizing it.

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

(c) (Design an IIR lowpass filter for both high stobpand attenuation and low ripple.) The second benefit of IIR filters is the ability to achieve fast transitions between high gain and low gain. Among other things, this allows us to transition between the passband and stopband more quickly, which in turn allows us to achieve relatively high stopband attenuation with low passband ripple.

Once again using the same transition band, design a lowpass filter with a passband ripple of less than 2 dB and a stopband attenuation of at least 20dB. You may use as many poles and zeros as you wish, but it is possible to meet these criteria with only two poles and four zeros. (Hint: use decibel mode to help you increase the stopband attenuation, and linear mode to help you decrease the passband ripple.)

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

Food for thought: For a more interesting challenge, design a lowpass filter with a passband ripple of less than 1 dB and a stopband attenuation of 60 dB. This can be done with six poles and six zeros, but you might want to use more than this.

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.

   - Use `soundsc` to listen to `vowel2`. What vowel does this signal represent?
   - As a measure of the spectrum, plot the magnitude DFT coefficients for this signal in decibels versus frequency in Hertz. Use only the first half of the DFT coefficients, where the maximum value on the x-axis is one half of the sampling rate.
   - From this plot, estimate the fundamental frequency of the vowel signal in Hertz.

   (b) (Design a vocal tract filter with both poles and zeros.) `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 3-D* using the command

   ```
   >> pole_zero_place3d([],[],vowel2_amps,2*pi*frq/8192);
   ```

   where `frq` is the fundamental frequency (in Hz) 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 nontrivial poles and six nontrivial 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.)

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

(c) (Design a vocal tract filter with only poles and zeros.) Now, repeat the above for a filter with 10 poles nontrivial and no nontrivial 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.)

- Include the GUI window with your matching filter in your report.
- What are the filter coefficients b_k and a_k for your filter?
- 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) (Automatically generate an all-pole filter model.) Use `aryule` to compute an all-pole model for `vowel2` with 10 poles.

- What are the resulting feedback coefficients?
- Make a pole-zero plot of the resulting filter.
- 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) (Construct and evaluate the vowel classifier.) 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`.

- 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.
- 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.)
- 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.

