C.10 Modulation and Demodulation

The purpose of this lab is to learn to use frequency domain concepts in practical applications. The application selected here is **amplitude modulation** (**AM**), a widely used technique in communication systems, including of course AM radio, but also almost all digital communication systems, including digital cellular telephones, voiceband data modems, and wireless networking devices. A secondary purpose of this lab is to gain a working knowledge of the **fast Fourier transform** (**FFT**) algorithm, and an introductory working knowledge of **filter design**. Note that this lab requires the Signal Processing Toolbox of Matlab for filter design.

C.10.1 Background

Amplitude Modulation

The problem addressed by AM modulation is that we wish to convey a signal from one point in physical space to another through some **channel**. The channel has certain constraints, and in particular, can typically only pass frequencies within a certain range. An AM radio station, for example, might be constrained by government regulators to broadcast only radio signals in the range of 720 kHz to 760 kHz, a **bandwidth** of 40 kHz.

The problem, of course, is that the radio station has no interest in broadcasting signals that only contain frequencies in the range 720 kHz to 760 kHz. They are more likely to want to transmit a voice signal, for example, which contains frequencies in the range of about 50 Hz to about 10 kHz. AM modulation deals with this mismatch by **modulating** the voice signal so that it is shifted to the appropriate frequency range. A radio receiver, of course, must **demodulate** the signal, since 720 kHz is well above the hearing range of humans.

In this lab, we present a somewhat artificial scenario in order to maximize the experience. We will keep all frequencies that we work with within the audio range so that you can listen to any signal. Therefore, we will not modulate a signal up to 720 kHz (you would not be able to hear the result). Instead, we present the following scenario:

> Assume we have a signal that contains frequencies in the range of about 100 to 300 Hz, and we have a channel that can pass frequencies from 700 to 1300 Hz.[5] Our task will be to modulate the first signal so that it lies entirely within the channel **passband**, and then to demodulate to recover the original signal.

AM modulation is studied in detail in exercise 5 of chapter 9. In that problem, you showed that if

$$y(t) = x(t)\cos(\omega_c t),$$

then the CTFT is

$$Y(\omega) = X(\omega - \omega_c)/2 + X(\omega + \omega_c)/2.$$

[5] Since Fourier transforms of real signals are symmetric, the signal also contains frequencies in the range -100 to -300 Hz, and the channel passes frequencies in the range -700 to -1300 Hz.

In this lab, we will get a similar result experimentally, but working entirely with discrete-time signals, and staying entirely within the audio range so that we can hear (and not just plot) the results.

The FFT Algorithm

In order to understand AM modulation, we need to be able to calculate and examine Fourier transforms. We will do this numerically in this lab, unlike exercise 5 of chapter 9, where it is done analytically.

In lab C.7, we used a supplied function called `fourierSeries` to calculate the Fourier series coefficients A_k and ϕ_k for signals. In this lab, we will use the built-in function `fft`, which is used, in fact, by the `fourierSeries` function. Learning to use the FFT is extremely valuable; it is widely used all analytical fields that involve time series analysis, including not just all of engineering, but also the natural sciences and social sciences. The FFT is also widely abused by practitioners who do not really understand what it does.

The FFT algorithm operates most efficiently on finite signals whose lengths are a power of 2. Thus, in this lab, we will work with what might seem like a peculiar signal length, 8192. This is 2^{13}. At an 8 kHz sample rate, it corresponds to slightly more than one second of sound.

Recall that a periodic discrete-time signal with period p has a discrete-time Fourier series expansion

$$x(n) = A_0 + \sum_{k=1}^{(p-1)/2} A_k \cos(k\omega_0 n + \phi_k) \tag{C.15}$$

for p odd and

$$x(n) = A_0 + \sum_{k=1}^{p/2} A_k \cos(k\omega_0 n + \phi_k) \tag{C.16}$$

for p even, where $\omega_0 = 2\pi/p$, the fundamental frequency in cycles per sample. Recall further that we can alternatively write the Fourier series expansion in terms of complex exponentials,

$$x(n) = \sum_{k=0}^{p} X_k e^{ik\omega_0 n}. \tag{C.17}$$

Note that this sum covers one cycle of the periodic signal. If what we have is a finite signal instead of a periodic signal, then we can interpret the finite signal as being one cycle of a periodic signal.

In chapter 9, we describe four Fourier transforms. The only one of these that is computable on a computer is the **DFT**, or **discrete Fourier transform**. For a periodic discrete-time signal x with period p, we have the **inverse DFT**, which takes us from the frequency domain to the time domain,

$$\forall\, n \in \mathit{Ints}, \quad x(n) = \frac{1}{p} \sum_{k=0}^{p-1} X'_k e^{ik\omega_0 n}, \tag{C.18}$$

and the **DFT**, which takes us from the time domain to the frequency domain,

$$\forall\, k \in \mathit{Ints}, \quad X'_k = \sum_{m=0}^{p-1} x(m) e^{-imk\omega_0}. \tag{C.19}$$

Comparing (C.18) and (C.17), we see that the DFT yields coefficients that are just scaled versions of the Fourier series coefficients. This scaling is conventional.

In lab C.7 we calculated A_k and ϕ_k. In this lab, we will calculate X'_k. This can be done using (C.19). The FFT algorithm is simply a computationally efficient algorithm for calculating (C.19).

In Matlab, you will collect 8192 samples of a signal into a vector and then invoke the `fft` function. Invoke `help fft` to verify that this function is the right one to use. If your 8192 samples are in a vector `x`, then `fft(x)` will return a vector with 8192 complex number, representing $X_0, ..., X_{8191}$.

From (C.19) it is easy to verify that $X_k = X_{k+p}$ for all integers k (see part 1 of the in-lab section below). Thus, the DFT X is a periodic, discrete function with period p. If you have the vector `fft(x)`, representing $X_0, ..., X_{8191}$, you know all X_k. For example,

$$X_{-1} = X_{-1+8192} = X_{8191}$$

From C.18, you can see that each X_k scales a complex exponential component at frequency $k\omega_0 = k2\pi/p$, which has units of samples per second. In order to interpret the DFT coefficients X_k, you will probably want to convert the frequency units to Hertz. If the sampling frequency is f_s samples per second, then you can do the conversion as follows (see box on page 141):

$$\frac{(k2\pi/p)[\text{radians/sample}]\, f_s[\text{samples/second}]}{2\pi[\text{radians/cycle}]} = kf_s/p[\text{cycles/second}] \tag{C.20}$$

Thus, each X_k gives the DFT value at frequency kf_s/p Hz. For our choices of numbers, $f_s = 8000$ and $p = 8192$, so X_k gives the DFT value at frequency $k \times 0.9766$ Hz.

Filtering in Matlab

The `filter` function can compute the output of an LTI system given by a difference equation of the form

$$a_1 y(n) = b_1 x(n) + b_2 x(n-1) + \cdots + b_N x(n-N+1) - a_2 y(n-1) - ... - a_M y(n-M+1). \tag{C.21}$$

To find the output y, first collect the (finite) signal x into a vector. Then collect the coefficients $a_1, \cdots, a_N$ into a vector `A` of length N, and the coefficients $b_1, \cdots, b_M$ into a vector `B` of length M. Then just do

```
y = filter(B, A, x);
```

Example: Consider the difference equation

$$y(n) = x(n) - 0.95y(n-1).$$

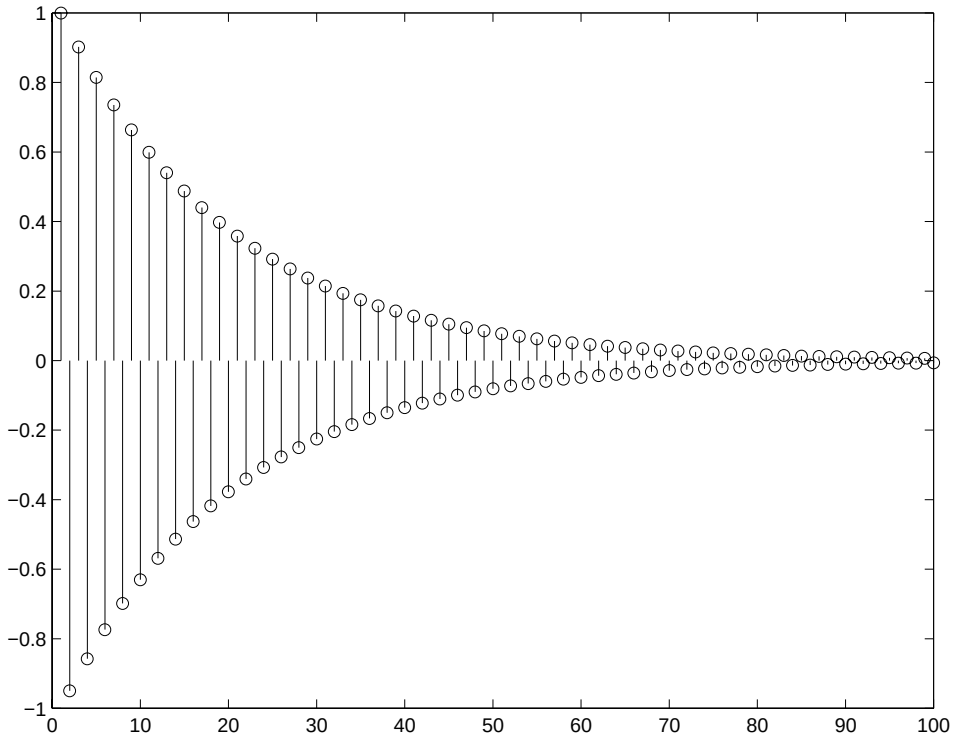

Figure C.12: Impulse response of a simple filter

We can find and plot the first 100 samples of the impulse response by letting the vector x be an impulse and using `filter` to calculate the output:

```
x = [1, zeros(1,99)];
y = filter([1], [1, 0.95], x);
stem(y);
```

which yields the plot shown in C.12. The natural question that arises next is how to decide on the values of `B` and `A`. This is addressed next.

Filter Design in Matlab

The signal processing toolbox of Matlab provides a set of useful functions that return filter coefficients `A` and `B` given a specification of a desired frequency response. For example, suppose we have a signal sampled at 8 kHz and we wish to design a filter that passes all frequency components below 1 kHz and removes all frequency components above that. The following Matlab command designs a filter that approximates this specification:

```
[B, A] = butter(10, 0.25);
```

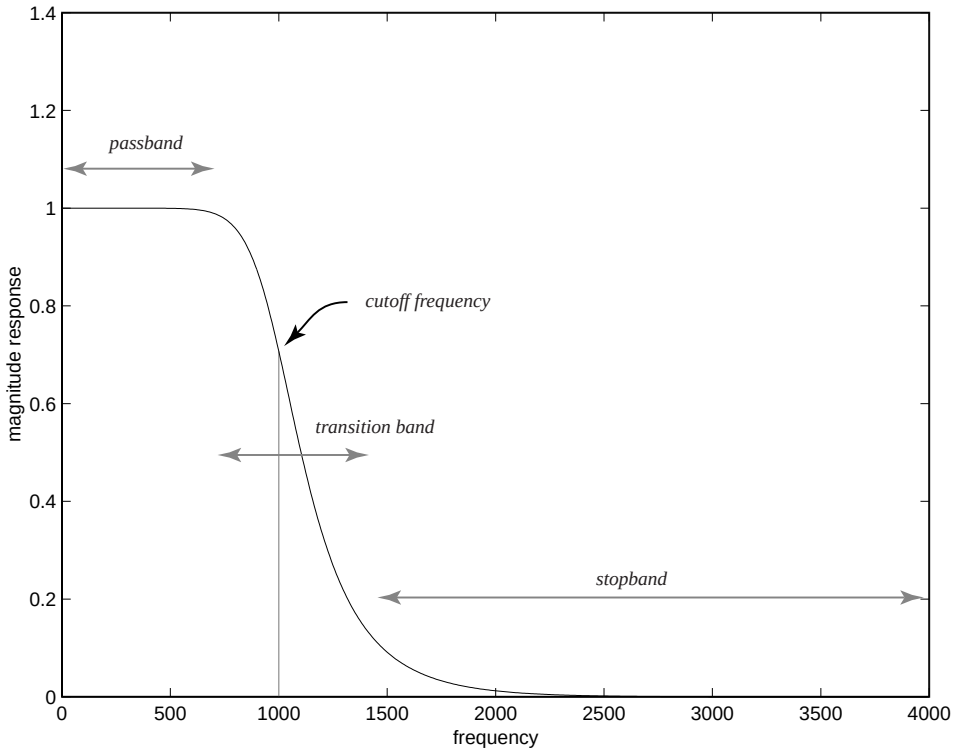

Figure C.13: Frequency response of a 10-th order Butterworth lowpass filter.

The first argument, called the **filter order**, gives M and N in (C.21) (a constraint of the `butter` function is that $M = N$). The second argument gives the **cutoff frequency** of the filter as a fraction of half the sampling frequency. Thus, in the above, the cutoff frequency is $0.25 * (8000/2) = 1000$ Hertz. The cutoff frequency is by definition the point at which the magnitude response is $1/\sqrt{2}$. The returned arrays `B` and `A` are the arguments to supply to `filter` to calculate the filter output.

The frequency response of this filter can be plotted using the `freqz` function as follows:

```
[H,W] = freqz(B,A,512);
plot(W*(4000/pi), abs(H));
xlabel('frequency');
ylabel('magnitude response');
```

which yields the plot shown in figure C.13. (The argument 512 specifies how many samples of the continuous frequency response we wish to calculate.)

This frequency response bears further study. Notice that the response transitions gradually from the passband to the stopband. An abrupt transition is not implementable. The width of the transition band can be reduced by using an order higher than 10 in the `butter` function, or by designing more sophisticated filters using the `cheby1`, `cheby2`, or `ellip` functions in the signal processing toolbox. The Butterworth filter returned by `butter`, however, will be adequate for our purposes in this lab.

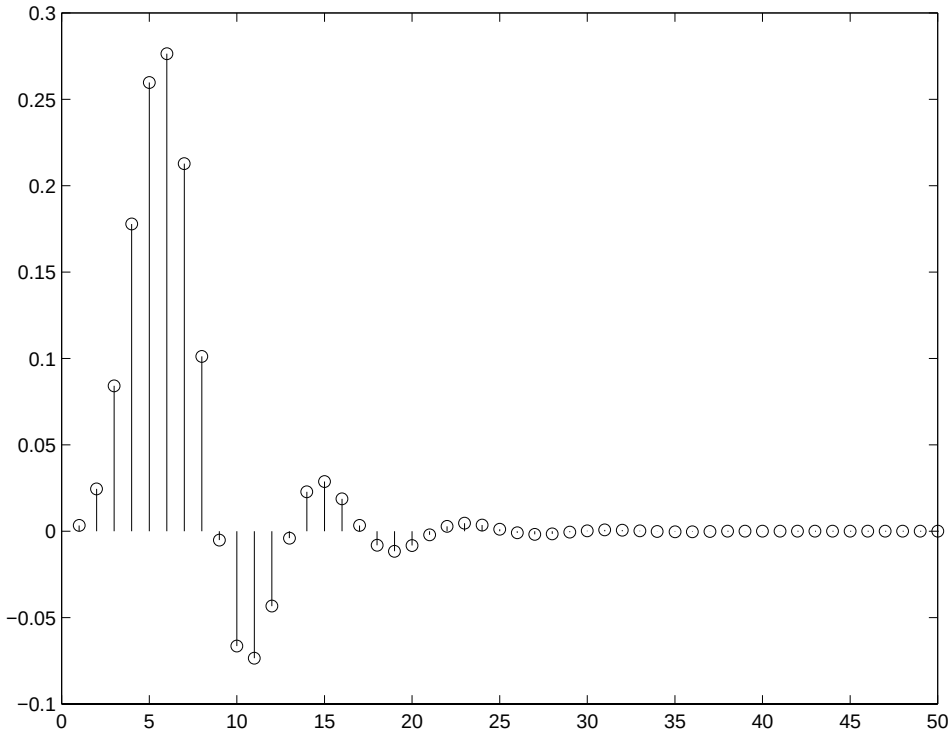

Figure C.14: Impulse response of a 10-th order Butterworth lowpass filter.

Using a higher order to get a narrower transition band can be an expensive proposition. The function `filter` works by implementing the difference equation (C.21). As M and N get larger, each output sample $y(n)$ requires more computation.

The first 50 samples of the impulse response of the filter can be plotted using the following Matlab code:

```
x = [1, zeros(1,49)];
y = filter(B, A, x);
stem(y);
```

This yields the plot shown in figure C.14.

C.10.2 In-lab section

1. Use (C.19) to show that $X'_k = X'_{k+p}$ for all integers k. Also, show that the DFT is conjugate symmetric, i.e. $X'_k = (X'_{-k})^*$ for all integers k, assuming $x(n)$ is real for all integers n.

2. In part 2 of the in-lab portion of lab C.7, we studied a **chirp** signal. We will use a similar signal here, although it will vary over a narrower range of frequencies. Construct the signal `x` given by:

```
t = [0:1/8000:8191/8000];
x = sin(2*pi*100*t + 2*pi*100*(t.*t));
```

This is a chirp that varies from about 100 Hz to about 300 Hz. Listen to it. Calculate its DFT using the `fft` function, and plot the magnitude of the DFT. Construct your plot in such a way that your horizontal axis is labeled in Hertz, not in the index k of X_k. The horizontal axis should vary in frequency from 0 to 8000 Hz.

3. Your plot from part 2 should show frequency components in the range 100 Hz to 300 Hz, but in addition, it shows frequency components in the range 7700 to 7900. These extra components are the potentially the most confusing aspect of the DFT, but in fact, they are completely predictable from the mathematical properties of the DFT.

   Recall that the DFT of a real signal is conjugate symmetric. Thus,

   $$|X'_k| = |X'_{-k}|.$$

   Thus, if there are frequency components in the range 100 to 300 Hz, then there should also be frequency components with the same magnitude in the range -100 to -300 Hz. These do not show up on your plot simply because you have not plotted the frequency components at negative frequencies.

   Recall that the DFT is periodic with period p. That is, $X_k = X_{k+p}$ for all integers k. Recall from (C.20) that the $k-th$ DFT coefficient represents a frequency component at kf_s/p Hertz, where f_s is the sampling frequency, 8000 Hertz. Thus, a frequency component at some frequency f must be identical to a frequency component at $f + f_s$. Therefore, the components in the range -100 to -300 Hertz must be identical to the components in the range 7700 to 7900 Hertz! The image we are seeing in that latter range is a consequence of the conjugate symmetry and periodicity of the DFT!

   Since the DFT is periodic with period 8000 Hertz (when using units of Hertz), it possibly makes more sense to plot its values in the range -4000 to 4000 Hertz, rather than 0 to 8000 Hertz. This way, we can see the symmetry. Since the DFT gives the weights of complex exponential components, the symmetry is intuitive, because it takes two complex exponentials with frequencies that are negatives of one another to yield a real-valued sinusoid.

   Manipulate the result of the `fft` function to yield a plot of the DFT of the chirp where the horizontal axis is -4000 to 4000 Hertz. It is not essential to include both endpoints, at -4000 and at 4000 Hertz, since they are constrained to be identical anyway by periodicity.

C.10.3 Independent section

1. For the chirp signal as above, multiply it by a sine wave with frequency 1 kHz, and plot the magnitude of the DFT of the result over the frequency range -4000 to 4000 Hz. Verify that the resulting signal will get through the channel described in the scenario on page 338. Listen to the modulated chirp. Does what you hear correspond with what you see in the DFT plot?

2. The modulated chirp signal constructed in the previous part can be demodulated by multiplying it again by a sine wave with frequency 1 kHz. Form that product, and plot the magnitude

of the DFT of the result over the frequency range -4000 to 4000 Hz. How does the result differ from the original chip? Listen to the resulting signal. Would this be an acceptable demodulation by itself?

3. Use the `butter` function to design a filter that will process the result of the previous part so that it more closely resembles the original signal. You should be able to get very close with a modest filter order (say, 5). Filter the result of the previous part, listen to the result, and plot the magnitude of its DFT in the frequency range -4000 to 4000 Hz.

   The modulation and demodulation method you have just implemented is similar to what is used many communication systems. A number of practical problems have to be overcome in practice, however. For example, the receiver usually does not know the exact frequency and phase of the carrier signal, and hence it has to somehow infer this frequency and phase from the signal itself. One technique is to simply include the carrier in the modulated signal by adding it in. Instead of transmitting

   $$y(t) = x(t)\cos(\omega_c t),$$

   we can transmit

   $$y(t) = (1 + x(t))\cos(\omega_c t),$$

   in which case, the transmitted signal will contain the carrier itself. This can be used for demodulation. Another technique is to construct a **phase locked loop**, a clever device that extracts the carrier from the transmitted signal without it being explicitly present. This method is used in digital transmission schemes. The details, however, must be left to a more advanced text.

   In the scheme we have just implemented, the amplitude of a carrier wave is modulated to carry a signal. It turns out that we can also vary the phase of the carrier to carry additional information. Such **AM-PM** methods are used extensively in digital transmission. These methods make more efficient use of precious radio bandwidth than AM alone.