Lab06: Quantization and Filtering

EECS20: Introduction to Real-Time Digital Systems

By K. H. Chiang, William T. Huang, Brian L. Evans.

URL: `http://www-inst.eecs.berkeley.edu/~ee20`
News: `ucb.class.ee20`

Assigned: 12 Mar 97, Checkoff: 19 Mar 97, Writeup Due: 21 Mar 97

1 Introduction

In this lab, we will study two related concepts in digital systems: quantization and filtering.

Previously, we have seen that we could not represent a continuous-time signal on a digital computer. Instead, we were forced to sample in time to get a discrete-time signal.

Similarly, we cannot exactly represent the amplitude of a given sample on a digital computer, since we only have a limited number of bits for that representation. If we have B bits, we can only have 2^B possible values for the amplitude. But if B is large enough, or if we have a computer that has a floating-point implementation, the approximate value will most likely be good enough for the application at hand.[1]

However, on a smaller, lower cost platform, such as a fixed-point DSP chip, we may be limited to a much smaller set of possible amplitude values. This process of reducing an infinite set of values to a finite set is called *quantization*. Quantization effects are quite pronounced when the number of bits available for representation is very low.

In lecture, we discussed the frequency domain representation of a signal. In the second part of this lab, we will filter speech and music samples with various filters to determine their effect upon the signal. We will also look at the frequency content of various signals and try to reconstruct those signals.

2 Quantization

If the amplitude of a signal takes on values over an interval from a to b, quantization of this signal into N levels can be thought of as breaking the interval into N bins. All values that lie in a given bin are rounded to the reconstruction value associated with that bin; reconstruction values are usually chosen to be the value of the center of each bin. Once the bins and reconstruction values have been determined, each sample of the signal is then quantized by figuring out the bin into which it falls, and then changing the sample value to the associated reconstruction value. This process is illustrated in Figure 1.

[1] Integers on most currently available machines are now 32 bits in length, giving 2^{32} possible values. Single precision floating-point numbers on most currently available computers also use 32 bits, but some of those bits are used to represent the exponent, allowing the representation of numbers from from 10^{-38} to 10^{37}. Double precision floating-point numbers use 64 bits instead, allowing the representation of numbers from 10^{-308} to 10^{307}.

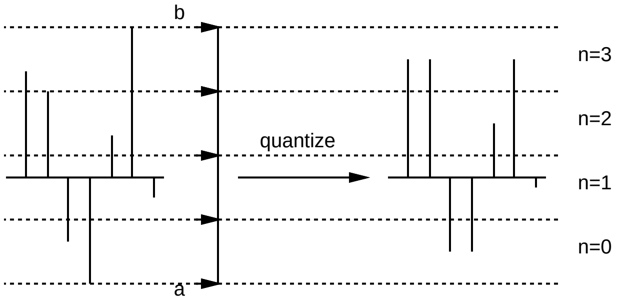

Figure 1: A signal is quantized into four levels. First the range from a to b is divided into four bins, with the four reconstruction values as the centers of each of the bins. Then the signal amplitudes are rounded to the reconstruction level of the bin into which they fall. Dotted lines indicate the bins. Note that the amplitudes that fall on the edge of a bin are arbitrarily declared to lie in the highest bin.

If B bits are used to represent the reconstruction values, then there are $N = 2^B$ possible reconstruction values.

If we do not care about the actual reconstruction values, but only wish to break the signal amplitudes into N possible levels, then one way to quantize is to:

(a) Normalize the signal to fall into the range 0 to 1.

(b) Multiply the normalized signal by N.

(c) Round the sample values down to the nearest integer.

(d) Divide by N.

This will give you N possible values for the signal in a range from 0 to 1.

Quantization can be thought of as taking your original signal and adding noise to it. The noise reflects the error introduced by quantization.

3 Finding the Frequency Content of a Signal

We know from previous lectures that any signal can be decomposed into a sum of scaled and shifted sinusoids of varying frequencies. We can represent this decomposition on two plots:

- a plot of the frequencies of the sinusoids against the magnitude of the scaling of the sinusoids, called the *magnitude response*, and
- a plot of the frequencies of the sinusoids against the phase of the sinusoids, called the *phase response*. This will include the sign of the scalings that was neglected in the previous plot.

Together, both of these plots fully describe the frequency content of the original signal.

We can be a bit more parsimonious by representing both magnitude and phase using complex numbers. This will then give us a single plot, but the magnitude and phase are the properties that we are usually concerned with in practice.

Consider an N point segment of a signal $x[n]$. If $x[n]$ is generated by sampling a continous-time (CT) signal with sampling period T, the corresponding sampling frequency f_s is:

$$f_s = \frac{1}{T}$$

The Nyquist sampling theorem says that the maximum frequency that you can see in a signal is going to be half the sampling frequency:

$$f_{max} = \frac{1}{2} f_s = \frac{1}{2T}$$

If `x` is an N point segment of $x[n]$, one way to determine its frequency content is to take its discrete Fourier transform (DFT) by using the fast Fourier transform (FFT) to compute it:

```
>> X=fft(x);
```

This gives us an N component vector X, which in general will be complex valued. The magnitudes of these complex numbers are a sampled and quantized version of the magnitude response mentioned above, and the phases of these complex numbers are a sampled and quantized version of the phase response.

If we index `X` by the variable k, the first [$k = 1$] component represents the average (DC) value of the original signal. The second [$k = 2$] component gives the frequency content at $\frac{f_s}{N}$ Hz. The $k = \frac{N}{2}$ component corresponds to f_{max} or $\frac{1}{2} f_s$.

From previous lectures, we know that changing the discrete-time frequency by 1 does nothing to the time domain representation of a signal. That is, $\cos 2\pi f_d n$ and $\cos 2\pi (f_d + 1) n$ are the same discrete-time (DT) signal. So the DFT of a signal should be periodic with period 1; we cannot tell the difference in the time domain between a sinusoid at frequency f_d and a sinusoid at frequency $f_d + 1$, so the frequency content at f_d should be the same as that at $f_d + 1$.

But where is the periodicity? It is reflected in the fact that the DFT is periodic with period N; we have sampled the frequency response at N points.

It turns out that sampling a signal in the time domain forces the frequency domain representation of that signal to be periodic. However, we are also sampling in the frequency domain. This forces the time domain signal to be periodic. This topic will be addressed in more depth when you take EE123.

One final note: if the signal that we DFTed was real valued, the magnitude response would be even about the $k = \frac{N}{2}$th component, and the phase response would be odd.

4 Filtering

Filters are systems designed to modify the frequency content of their input signals. There are five major classes of filters:

- lowpass: removes the high frequency portions of the input signal, with high frequency portions defined as those frequencies around f_{max}.
- highpass: removes the low frequency portions of the input signal, with low frequency portions defined as those frequencies around 0.

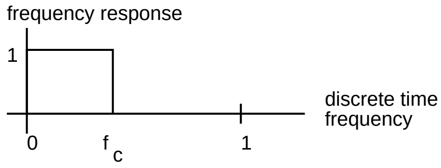

Figure 2: Ideal lowpass filter frequency response with cutoff frequency f_c.

- bandpass: passes only a range of frequencies.
- bandstop: removes a range of frequencies, but lets all other frequencies though.
- allpass: passes all frequencies. This looks useless, but the phase shifting of certain frequencies is what we are after here.

In filter design programs, the maximum frequency f_{max} is usually taken to be 1. So the allowable range of "normalized frequencies" is from 0 to 1.

The `fir1` command will design lowpass filters, given the cutoff frequency and the filter order n. The cutoff frequency for a lowpass filter is defined as the frequency above which all other frequencies will be removed. As mentioned above, its allowable range of values is between 0 and 1, corresponding to 0 and f_{max}.

The filter order determines the "goodness" of the filter. Ideally, a filter would pass all frequencies below the cutoff frequency, and reject all frequencies above the cutoff, as in Figure 2. Since these filters are nonideal ones, the transition band, or the range of frequencies over which the filter stops passing frequencies and begins rejecting them instead, will not be as steep. A higher order filter will have a smaller transition band, but at the expense of higher computational cost. There is no such thing as a free lunch.

The `fir1` command returns two vectors labeled b and a:

```
>> [b,a] = fir1(order, fc)
```

To view the results of `fir1`, use the `freqz` command:

```
>> freqz(b,a)
```

This will plot the filter frequency response in magnitude and in phase against the "normalized frequencies" from 0 to 1, which can be appropriately translated into actual frequencies.

To actually use the filter that you have designed, try the Matlab command:

```
y=filter(b,a,x)
```

where x is the input sequence and y is the output sequence. If you compare the DFT of the input sequence $X[k]$ to the DFT of the output sequence $Y[k]$, you will find that $Y[k]$ is the product of $X[k]$ and the frequency response of the filter.

5 Questions

1. Record speech sampled at $f_s = 11$ kHz. Generate 8 bit, 4 bit, 2 bit, and 1 bit quantized speech. Compare the original and quantized speech both graphically and audibly.

2. Record a speech sample with $f_s = 44$ kHz. Design filters using `fir1` that cut off at 15 kHz, 5 kHz, and 4 kHz to simulate the signals transmitted by broadcast FM radio, broadcast AM radio, and the telephone network. Play back the filtered signals. Repeat the above for a music signal. Use your judgment in deciding what the filter orders should be. Remember that the higher the order, the more time you have to spend waiting for the computer to determine the filter response; however, the higher the order, the closer the filter frequency response approaches that of the ideal filter.

3. Record speech sampled at $f_s = 44$ kHz. Keep every fourth sample, using Matlab constructs similar to the ones you used in `half` from lab01. [This process is called downsampling; downsampling followed by filtering is called decimation.] Play back the result. Then design a lowpass filter with a cut off frequency at 5500Hz, and use it to filter the downsampled signal. Play back the result.

4. You are given two signals $x[n]$ and $y[n]$. You are also told that, with $x[n]$ as input, $y[n]$ is the output of a lowpass filter with cutoff frequency f_c. How would you obtain $z[n]$, the output of a high pass filter with cutoff frequency f_c and input $x[n]$?

5. How would you create a bandpass filter from lowpass and highpass filters? Draw a block diagram to illustrate your idea.

6. Record a speech sample with $f_s = 22$ kHz. Using filters, divide the speech sample into a low frequency band from 0Hz to 1kHz, a mid frequency band from 1kHz to 4kHz, and a high frequency band from 4kHz to 11kHz. Listen to the filter outputs. Can you identify the original speech?

7. Generate a musical note with the electronic keyboard, and sample it at $f_s = 11$ kHz. Cut off the beginning and ending transients. Then analyze its frequency content by using `fft`. Duplicate that note using the sum of appropriately scaled and shifted sinusoids.