ENGIN 112: Notes for Module 2 - Signal Processing

1. Signals and Frequency

A **signal** is a function of time (for example, $s(t)$), where the variations in time are used to carry information. In ECE, we usually think of a signal as being a time-varying voltage at some point in a circuit. In typical applications, a **transducer** is used to convert between the physical quantity representing information and an electrical signal that carries that information. For example, a **microphone** converts the pressure wave that represents sound into an electrical signal, and a **speaker** converts an electrical signal back into a sound/pressure wave.

Signal processing systems are operations that store signals, extract information from them, convert them into forms more suitable for use, etc. One familiar example is the MP3 system, which converts electrical signals representing sound into digital data, stores the data in an efficient way, and converts the data back into electrical signals that can be fed into speakers. The key to the operation of MP3 and many other processing systems is the **frequency** content of signals. In particular: **Every signal can be represented as a sum of sinusoids having different frequencies.**

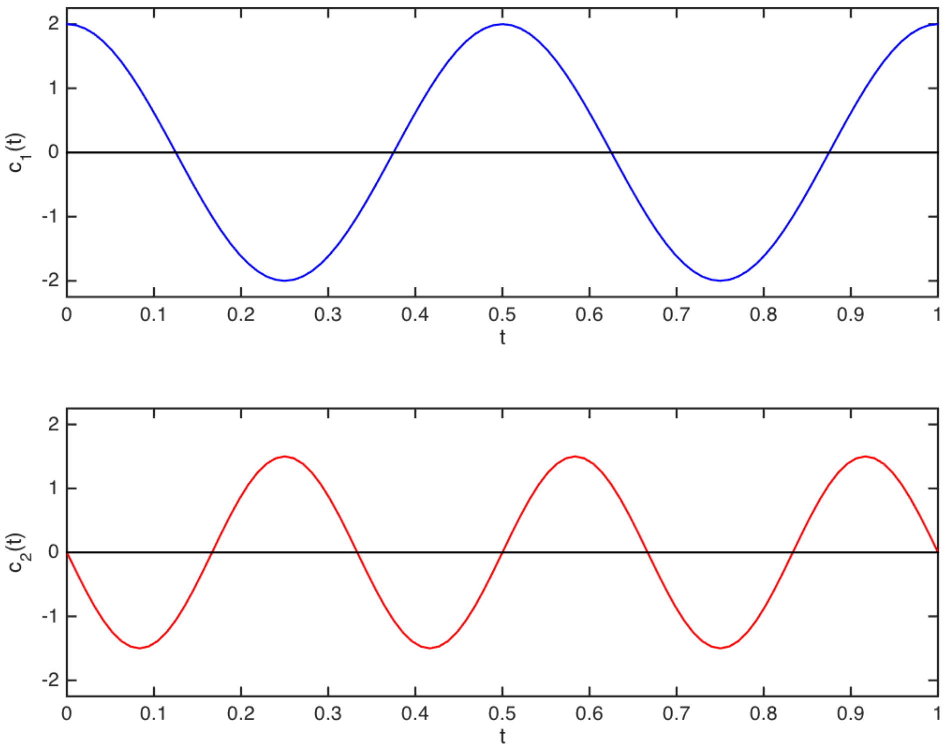

Figure 1: Examples of sinusoids. The top plot shows $c_1(t) = 2\cos(4\pi t)$ $(f = 2, T = 1/2, A = 2, \theta = 0)$, while the bottom plot shows $c_2(t) = 1.75\cos(6\pi t + \pi/2)$ $(f = 3, T = 1/3, A = 1.75, \theta = \pi/2)$.

A sinusoid is a function of the form

$$c(t) = A\cos(2\pi f t + \theta) \tag{1}$$

where A is the sinusoid's *magnitude* (units: volts), f is its *frequency* (units: Hertz (Hz) or cycles/sec), and θ is its *phase* (units: radians). The *period* of the sinusoid is the time it takes to go through one cycle - that is, for the argument of the sinusoid to change by 2π radians. So, the period T (in sec) is equal to $1/f$.

Every signal $s(t)$ can expanded in the form:

$$s(t) = A_0\cos(2\pi f_0 t + \theta_0) + A_1\cos(2\pi f_1 t + \theta_1) + A_2\cos(2\pi f_2 t + \theta_2) + \ldots \tag{2}$$

where $(f_k, k = 0, 1, 2, \ldots)$ is a set of frequencies, and (A_k, θ_k) are the magnitude and phase for the sinusoid with frequency f_k. The magnitudes of the sinusoids that make up $s(t)$ are especially important, since they tell how significant each sinusoid is in the signal (large magnitude means an important component; small magnitude means that the component is not very significant). A plot of magnitude vs. frequency for a signal (showing the relative significance of the components at different frequencies that make up the signal) is called the signal's **spectrum**. We can think of a signal's spectrum as a portrayal of the signal in terms of its frequency content.

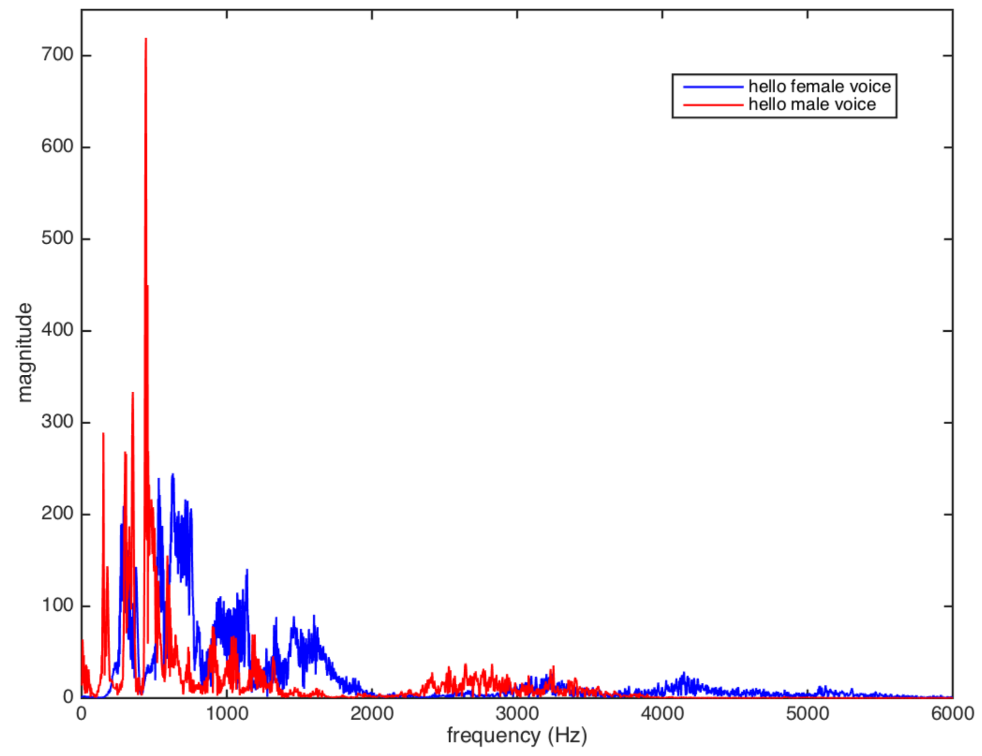

Figure 2. Spectra for the word ”hello” spoken by two different people.

2. Sampling, Interpolation, and Quantization

Most modern signal processing systems (such as MP3 players) employ **digital** processors that require data to be represented by strings of bits. Almost always, information-carrying signals (such as those representing sound) start as *analog* signals - that is, signals that are continuous in time and amplitude. These must be converted to digital representations. The main steps in *analog-to-digital conversion (ADC)* are **sampling** and **quantization**.

(a) **Sampling:** The idea in sampling is to replace a signal that varies continuously in time with a set of samples (points) taken from the signal at regular intervals. That is, say that we have a **sample period** of T_s sec. Then the analog signal $s(t)$ is replaced by the set of samples $(s(nT_s), n = 0, 1, 2, \ldots)$. (*Note:* The quantity $f_s = 1/T_s$ is called the **sampling rate** (in Hz).)

When a sampled signal is to be output to a user (for example, sent by an MP3 player to a speaker), it needs to be converted back into an analog signal by an **interpolator**. The standard for interpolation is that it reconstructs the *lowest frequency* signal that is consistent with the samples. When will the interpolator output be the same as the signal we started with? To answer that, we can go back to the representation of a signal in terms of sinusoidal components (eq. 2). Say we have a sinusoid with frequency f, and we sample that sinusoid at sampling rate $f_s > 2f$. Then if we put the samples through an interpolator, the interpolator output will be the original sinusoid - that is, the lowest-frequency signal that is consistent with the samples is the signal we started with. On the other hand, if we sample at a rate $f_s < 2f$, the interpolator output will be a lower-frequency sinusoid than the original signal - that is, the interpolator will reconstruct the signal at an incorrect (lower) frequency. This problem of a reconstructed signal having the wrong frequency is called **aliasing**. (See Figure 3 for an example.)

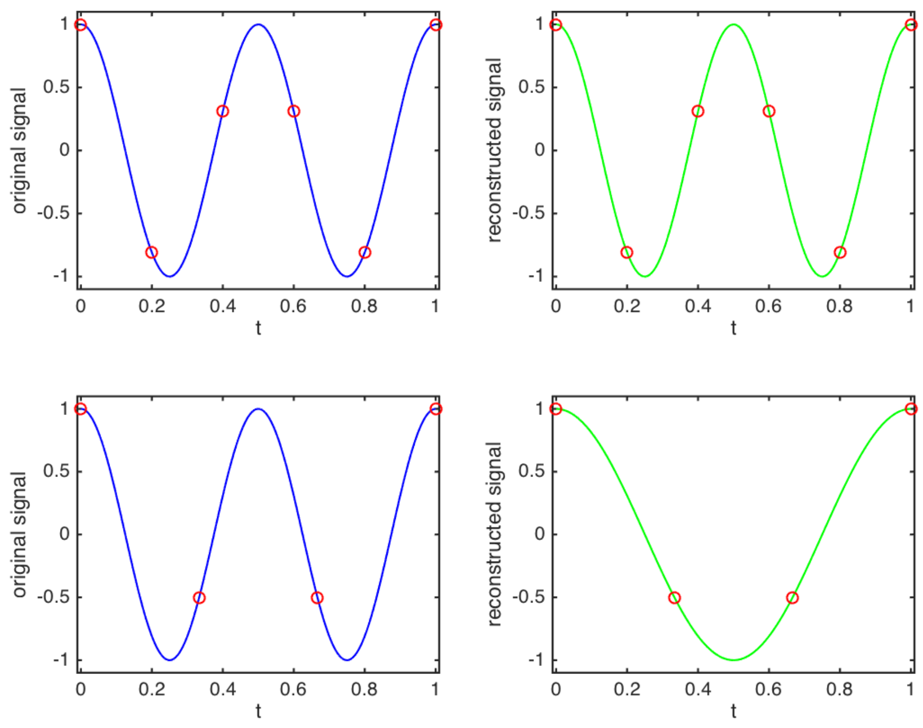

Figure 3. Illustration of sampling and aliasing. The top left plot shows the signal $s(t) = \cos(4\pi t)$ in blue - note that this has a frequency of $f = 2$ Hz. The red circles show samples taken at a sample rate of $f_s = 5$ Hz (so, $f_s > 2f$). The top right plot shows in green the signal that is reconstructed from the samples by an interpolator. Since we sampled the sinusoid at a rate greater than twice its frequency, the reconstructed signal is the same as the original. The bottom left plot shows the same signal $s(t)$, now sampled at a rate $f_s = 3$ Hz (so, $f_s < 2f$). The bottom right plot shows the signal reconstructed from the samples by an interpolator. In this case, because we did not sample at a fast enough rate, the reconstructed signal has the wrong frequency (1 Hz), and we have aliasing.

Now remember from eq. 2 that a signal $s(t)$ is a sum of sinusoids at different frequencies. For $s(t)$ to be reconstructed correctly from its samples by an interpolator (that is, to avoid aliasing), *all* of the signal's sinusoidal components must be reconstructed correctly. That is, we must have a sampling rate that is greater than 2 times the frequency of every component. This leads to the **Nyquist-Shannon Sampling Theorem:**

The sampling rate must be greater than 2 times the highest frequency in the signal.

This theorem is named for Harry Nyquist (1889-1976) and Claude Shannon (1916-2001), two people who developed the principles of sampling, along with many of the other foundations of communication systems and information theory. The frequency $2f_{max}$, where f_{max} is the highest frequency in $s(t)$, is called the *Nyquist Rate* for $s(t)$ - it is the minimum sampling rate needed to avoid aliasing.) For example, the highest sound frequency that is audible to humans is around 20 kHz (kHz = kilohertz = 1000 Hz). Music signals can contain components at frequencies close to that limit. So, if we take $f_{max} = 20$ kHz to be the highest frequency in a music signal, then to avoid aliasing we must sample at a rate $f_s > 2(20) = 40$ kHz. In fact, the sampling rate used for compact disc recordings and MP3 files is 44.1 kHz (that is, 44100 samples/sec).

(b) **Quantization:** After sampling, the signal is represented by a set of voltage samples. The task of a quantizer is to convert each voltage sample into a string of bits. Say that we want to use B bits/sample. Then we can represent a total of $L = 2^B$ distinct voltage levels (or steps). Suppose that the maximum (positive and negative) input voltage that we want to represent is V_{max}. In a **uniform quantizer**, we set the step size to be $\Delta = 2V_{max}/L$, and set the L quantizer output voltage levels to be $-(L-1)\Delta/2, -(L-3)\Delta/2, \ldots, -\Delta/2, \Delta/2, \ldots, (L-3)\Delta/2, (L-1)\Delta/2$. Each output level is then coded by a string of B bits. (See Figure 4 for an example.)

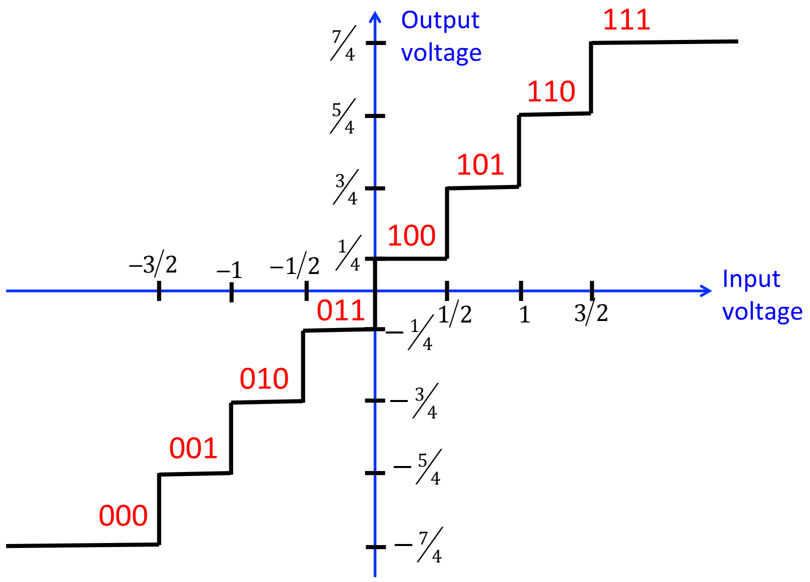

Figure 4. Example of a quantizer input-output characteristic. In this example we use $B = 3$ bits/sample, so there are $L = 2^3 = 8$ output voltage levels. The maximum input voltage is $V_{max} = 2$ volts, the step size is $\Delta = 2(2)/8 = 1/2$ volt, and the output levels are at $-(8-1)(1/2)/2 = -7/4, -(8-3)(1/2)/2 = -5/4, \ldots, (7-1)(1/2)/2 = 7/4$ volts. Each output voltage level is then represented by a string of 3 bits.

Note that quantization necessarily leads to error in the representation of the samples, because an entire range of input voltages gets mapped into one output voltage level and corresponding bit string. For example, in the quantizer of Fig. 4, every input voltage in the range $1/2 \leq V_{in} < 1$ gets mapped to the single output voltage $V_{out} = 3/4$ volt, and is represented by the bit string 101. Since the digital processor only has the bit string, the exact value of the input voltage sample is lost. The error caused by quantization is called **quantization noise**. (In ECE, unwanted random signals are frequently called *noise*.) We characterize the size of the noise by the **quantization noise power**, which is defined to be the average value of the square of the quantization error. We can usually assume that the quantization error for any given sample is equally likely to take any value between $-\Delta/2$ and $\Delta/2$. This leads to

$$\textit{Quantization noise power} = \Delta^2/12 \quad \textit{Watts.} \tag{3}$$

Note that if we increase the number of bits/sample from B to $B+1$, then we double the number of quantizer output levels (from 2^B to $2^{B+1} = 2(2^B)$). But that means we cut the step size Δ in half, and reduce Δ^2 by a factor of 4. So, **every increase of 1 bit in the length of the digital representation of a sample leads to a reduction in quantization noise power by a factor of 4**. The quality of a signal after quantization is measured by the **signal-to-quantization noise ratio (SQNR)**, which is defined to be

$$\textit{SQNR} = \frac{\textit{average input signal power}}{\textit{quantization noise power}} \tag{4}$$

A typical value for the average input signal power is ${V_{max}}^2/3$ Watts, which gives

$$\textit{SQNR} = \frac{{V_{max}}^2/3}{\Delta^2/12} = L^2 = 2^{2B} = 4^B. \tag{5}$$

We usually write SQNR in terms of **decibels (dB)**, defined as

$$\textit{(SQNR)}_{dB} = 10\log_{10}(\textit{SQNR})$$

For example, cd recordings use $B = 16$ bits/sample, so SQNR= $4^{16} = 4.3 \times 10^9$; in decibels, this is $10\log_{10}(4.3 \times 10^9) = 96.3$ dB. Also, as we've seen, increasing the length of the digital representation for a sample by 1 bit reduces the quantization noise power by a factor of 4, which *increases* the SQNR by a factor of 4. Note that for any positive number x, $10\log_{10}(4x) = 10\log_{10}(x) + 10\log 10(4) \approx 10\log_{10}(x) + 6$. This leads to the **6 dB rule for quantizer design:**

Every increase of 1 bit in the length of the digital representation of a sample leads to an increase of 6 dB in the SQNR.

3. Memory Requirements and Compression Coding

We can now calculate how much memory is required to store a typical cd recording. An average song lasts about 227 seconds. The signal is sampled at a rate of 44100 samples/sec, for a total of $227 \times 44100 \approx 1 \times 10^7$ samples. For stereo recordings, there are two signals/song (the left and right channel signals), so we have about 2×10^7 samples/song. Each sample is represented by 16 bits = 2 bytes. So, storing one song in cd format requires about (2×10^7 samples/song) $\times$ 2 bytes/sample $= 4 \times 10^7$ bytes/song. That is, we need about 40 MB (MB = megabytes = 1 million bytes) of memory to store one song. A collection of 1000 songs would require 40 GB of memory (GB = gigabytes = 1 billion bytes).

Large memory requirements are one price that we pay for having such high quality in cd recordings. The goal of MP3 developers was to implement efficient **compression coding** - that is, to reduce memory requirements while not causing too serious a deterioration in the quality of the recorded song. To do this, they again turned to the frequency content of the signals. In particular: for cd recordings we directly store samples from the time signal. In MP3, the digitized signal samples are split into groups of 576 points (corresponding to time intervals of 576 samples $\times$ 44100 samples/sec = 0.013 seconds). Each group of samples is then run through a **Fast Fourier Transform** - this is an algorithm (program) that computes the decomposition of the signal in the time interval into sinusoids of different frequencies (like eq. 2). The frequencies of the sinusoids are known in advance, so the frequencies themselves don't have to be stored; to represent the signal we just need to keep the magnitude and phase for each frequency. MP3 then takes advantage of the fact that most frequency components are insignificant. In particular, MP3 uses **perceptual coding** - that is the principle that large (loud) frequency components mask any smaller (quieter) components that occur at around the same time. Our ears cannot perceive the smaller components, so there's no point in saving them - MP3 just sets the magnitudes of all of those smaller components to zero. The small set of significant frequency component magnitudes and phases are stored. To reconstruct the signal, MP3 uses an **inverse FFT** that essentially forms the sum in eq. 2 using the stored magnitudes and phases (where most of the magnitudes have been set to 0).

The use of perceptual coding allows MP3 to be very efficient in terms of memory requirements. Recall that for a cd recording of a song, the number of bits/sec that must be stored (called the **bit rate**) is 2 signals $\times$ 44100 samples/sec for each signal $\times$ 16 bits/sample = **1.411 Mbits/sec**. The most common version of MP3 has a bit rate of **128 kbits/sec** - about 1/11 $\times$ the cd bit rate. So, while storing 1000 songs in cd format requires 40 GB of memory, in MP3 format we only need 40/11 = 3.6 GB. We do pay a slight price in quality for this storage efficiency. While SQNR for a cd recording is about 96 dB, the overall signal-to-noise ratio for a typical MP3 recording (which includes the extra error introduced by discarding small frequency components) is about 80 dB. However, that still gives a signal power that is $10^8\times$ the noise (error) power.

4. Time-Varying Frequency Content and the Spectrogram

The reason that MP3 splits a signal into small time segments before converting into frequency components is that signals like music have frequency content that can vary significantly over time - some parts of a song may have lots of low-frequency sounds (bass) while other parts of the same song may have a lot of high-frequency content (treble). To be effective at identifying the frequency components that are insignificant and can be discarded in perceptual coding, we need to use time windows that are short enough that the frequency content doesn't change.

Because the frequency content of signals like music changes over time, a single spectrum does not give a very complete picture of that content. From a single spectrum, we can identify which frequency components are important on average over time, but we can't tell how the frequency changes with time. To show how the frequency content of a signal varies with time we use a **spectrogram**. For a spectrogram, the (sampled and digitized) signal is split into short time segments, and the frequency content for each segment is computed with an FFT (like in MP3). A spectrogram is a plot that has frequency f on one axis (usually the horizontal) and time t on the other axis (usually the vertical). At each point (f, t), a color is displayed that shows the magnitude of the sinusoidal component having frequency f in the time interval containing t. (So, each row in a spectrogram is like a spectrum for a small segment of the signal, where spectral magnitude is shown by color). For example, Figure 5 shows the spectrogram for a 6.8 second segment of a piano concerto.

We can think of a spectrogram as a picture of how the frequency content of an audio signal changes with time. Spectrograms are very useful for distinguishing different sound signals, and are used in applications like echocardiograms for identifying heart rhythms. Another example is the the well-known app Shazam, which identifies songs from short (10 second) clips. Shazam does this by matching the pattern of spectrogram peaks (the frequency with maximum magnitude in each short time segment) for the clip to spectrogram peak patterns stored for a large number of songs.

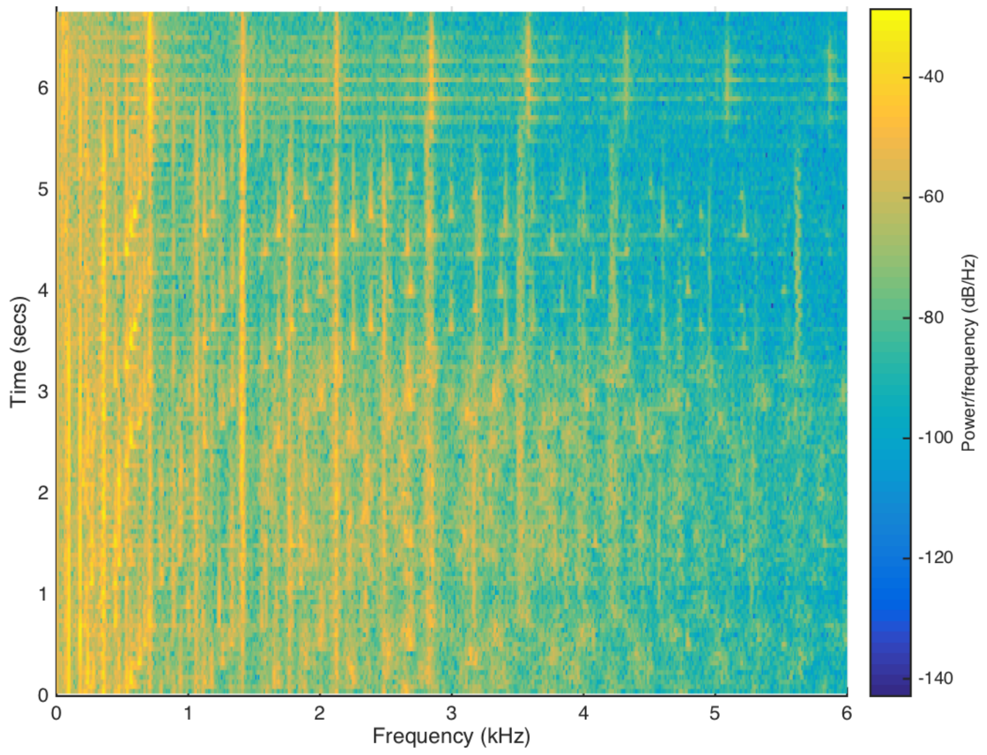

Figure 5. Spectrogram of a segment of a piano concerto. Brighter colors indicate higher magnitudes for frequency components. Multiple instruments are playing early in the segment, and significant components are spread out across frequencies. At the end of the segment only a piano is playing, and there are just a few very distinct strong components.