Reducing Audio Bandwidth with an FFT-based Low-Pass Filter

Trent Geisler
Analytics and Data Science
Kennesaw State University
Kennesaw, Georgia 30144

Andrew Henshaw
Analytics and Data Science
Kennesaw State University
Kennesaw, Georgia 30144

Lauren Staples
Analytics and Data Science
Kennesaw State University
Kennesaw, Georgia 30144

***Abstract*—The abstract goes here.**

I. Introduction

Filtering an audio signal is extremely important for many reasons. When an appropriate filter is designed and used to remove any unwanted frequencies, then the bandwidth and power necessary for audio signal transmission is reduced. Why is this important? The frequency bands used for the transmission of many types of signals are scarce resources. Every transmitter that can interfere with others has to operate in a licensed band and is subject to bandwidth limitations. This is why multiple radio stations can operate in the same geographical area. They are assigned different center frequencies and they have a certain bandwidth they can occupy. If one radio station transmits beyond their assigned bandwidth, they will impact the signal transmission of other local radio stations. One way to ensure that a radio station stays within their assigned bandwidth is through the use of filtering. [2]

Additionally, if you travel to a different city, you will find radio stations that operate at the same frequency as the radio stations located locally here in Atlanta. This is possible because radio transmitters are limited in power. With unlimited power, for example, one could hear FM 106.7 throughout the United States and no other radio station would be able to use the frequency 106.7 MHz. But power is limited, and high-power radio transmissions cost a lot of money [2]. This is why some radio stations like to brag about how powerful their transmissions are. It is also why start-up radio stations have a very limited range. Therefore, if power costs money, one does not want to waste it by boadcasting frequencies that most humans cannot hear.

Both bandwidth and power savings are the motivation behind Fast Fourier Transform (FFT)-based low-pass filters. This paper will demonstrate how a Finite Impulse Response (FIR) low pass filter can remove unecessary frequencies in order to reduce the bandwidth and power required for transmission.

A. Objective

Our objective is to design and demonstrate an FFT-based FIR low-pass filter to reduce bandwidth of a recorded audio signal for the purposes of AM Radio transmission.

II. Methods

We collected data, designed a low pass filter, pushed the signal through the filter, and then evaluated the resulting signal.

A. Data

We collected an audio signal by recording the song ‘Flight of the Bumblebee’ at the full frequency spectrum that a compact disc (CD) is recorded at, which is 44.1 kHz. The human ear does not even hear this full spectrum. This means that there are potential bandwidth ‘savings’ for transmission! This collected audio recording is referred to as the ‘raw signal.’

B. Filter Design

The typical Human can only hear frequencies in the range of 20Hz through 20KHz, and this range only decreases as humans age [3]. Using the website http://onlinetonegenerator.com/hearingtest.html, we will test the hearing range of each member listening to our final presentation. This demonstration will provide a tangible example why appropriate filtering of an audio signal will have little to no impact on the quality of sound that a person hears after the filtering.

When our project group performed the hearing test, the highest audible frequency was 15KHz. Since our project group was not able to hear the frequencies above the 15KHz threshold, any low-pass filter that removed the high frequencies above 15KHz would have no impact on the quality of the audio signal that we would hear. Recall, removing the unecessary frequencies from an audio signal transmission provides a few nice benefits such as the following:

1) It reduces the necessary power for transmission of the audio signal
2) It reduces the necessary bandwidth for transmission

Filtering is the processing of a time-domain signal resulting in some change in that signals original spectral content. The change is usually the reduction or filtering of unwanted input spectral components. [4]. Given the human threshold for hearing, an obvious opportunity is to apply low-pass filtering in order to transmit audio signals that humans can actually hear with smaller bandwidths. The ideal low-pass filter would completely eliminate all frequencies above a certain cutoff point (in our hearing test example, the cutoff would be at

15KHz) while passing all frequencies below the cutoff point [5].

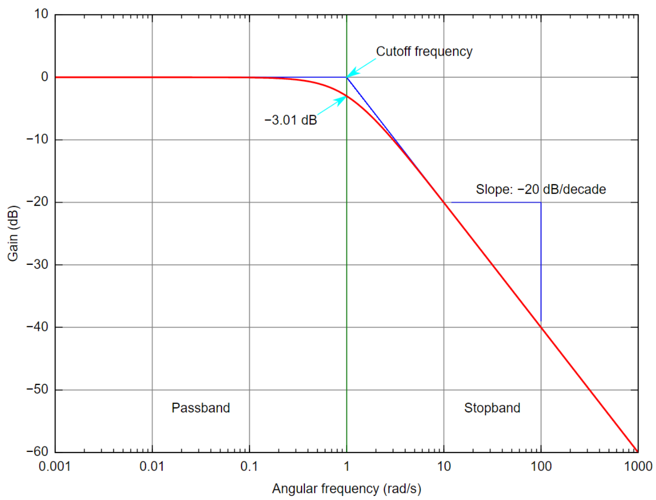

Fig. 1. Example of Low-Pass Filter: https://upload.wikimedia.org/wikipedia/commons/6/60/Butterworth_response.svg

In order to filter out higher frequencies in an audio signal transmission, we will build a low-pass filter. Specifically, we will build a low-pass filter using a finite number of non-zero filter coefficients which is called a *Finite Impulse Response* filter or FIR. Given an impulse response, we can find the coefficients of the filter, and vice versa [2]. For example, Figure 1 shows a graphical depiction of a low-pass filter that begins to cut out the frequencies above 1 rad/s. Frequencies below the cutoff frequency are considered to be in the "passband," or allowable frequency band. Frequencies above the cutoff frequency filtered out and are considered in the "stopband." Ideally, the slope between the passband and the stopband would be as steep as possible to limit the passing of unwanted high audio frequencies. The question still remains, how do you build a filter that eliminates these frequencies? What is an impulse response? How do we translate an impulse response to a frequency response and vice versa. Let's address those questions next.

1) Impulse Input, Impulse Response, and FFT: In our filtering example, we will be dealing with a linear time-invariant (LTI) System. A linear system is a class of systems where the system's outputs are the sum of the outputs of it's parts. Additionally, time-invariant refers to a system where a time delay in the input sequence causes an equivalent delay in the output sequence. Now, if we are given a LTI system, we can calculate everything about the system if we know the *unit impulse response*. The *unit impulse response* refers to the system's time-domain output sequence when the input is a single unity-valued sample (unit impulse) surounded by zero-valued samples. Furthermore, knowing the impulse response of an LTI system, the *output sequence* is calculated by taking the *convolution* of the input sequence and the system's impulse response [4]. We will talk more about convolutions later, but we typically do not perform convolutions in the time domain since they are computationally expensive. Instead, multiplication is performed in the frequency domain. In order to transform the impuse response into a *frequency response*, we take the Fast Fourier Transform (FFT) of the impulse response [4].

Let's explain with a simple example. If we let $x(n)$ be a discrete-time sequence of individual signal amplitudes, then we can define a simple LTI *averager* system that takes the average of the last four inputs as follows:

$$y(n) = \frac{1}{4}\left[x(n) + x(n-1) + x(n-2) + x(n-3)\right] = \frac{1}{4}\sum_{k=n-3}^{n} x(k)$$

Given this simple averager, we can show the block diagram in Figure 3 (a), impulse input and impulse response in Figure 3 (b), and the frequency magnitude response created from the FFT of the impulse response in Figure 3 (c). The block diagram in Figure 3 (a) - also referred to as the *filter structure*, simply shows how an impulse input is transformed into a impulse response. The impulse response, $y(n)$, is created by passing the impulse input, $x(n)$, into the system and storing the four most recent values. Once there are four values, they are added together and multiplied by $\frac{1}{4}$ to calculate the average. The sequence proceeds one step, and then performs another average of the four most recent $x(n)$ values. The averaging continues as long as impulse input enters the filter. Since there are four separate input sample values to calculate an output value, the structure of this filter can be referred to as a *4-tap tapped-delay line FIR filter* using digital filter vernacular. The coefficients of this filter are all $\frac{1}{4}$ and we can use an FFTon the filter to provide all the frequency domain information.

For example, we can create a simple input impulse in Python by generating random values between -0.5 and 0.5 and then using the numpy package in Python to apply the FFT in order to get the frequency response, labeled x_spectrum in Listing 2, of the impulse input. We can then take get the frequency response of the filter coefficients (labeled h_n in Listing 2) by performing another FFT. The frequency response of the filter coefficients is labled as freq_response in Listing 2. Finally, we can generate the new output spectrum by multiplying the x_spectrum and the freq_response since they are both in the frequency domain. See the entire code snipit below in Listing 2. All of the code for this example to include the code for the plots is listed in the Appendix.

Listing 1. Simple Averager Filter

```
#generate impulse input
num_samples = 100
x_n = np.random.rand(num_samples)

#generate the frequency response for impulse input
X_m = np.fft.fft(x_n,N_fft)

#generate filter coefficients for simple averager
h_n = np.ones(4)/4

#generate the frequency response of the filter
# the second argument of fft is the number of fft bins
```

```
N_fft = 1024 #use a power of 2
H_m = np.fft.fft(h_n,N_fft)

#generate the output after filter
Y_m = H_m * abs(X_m)
```

Figure 2 shows the impact of the simple averager filter on the original X_m (input impulse after transformation into the frequency domain). The original X_m is ploted in blue in the bottom graph in Figure **??** and has a lot of frequency information across the entire domain. Once the simple average filter is applied, the output spectrum, Y_m in red has greatly reduced frequency content as the frequency gets further away from 0. Additionally, you should notice that the output spectrum (red line) is created by simply multiplying the frequency response of the filter coefficients (top blue graph) by the input spectrum (bottom blue graph).

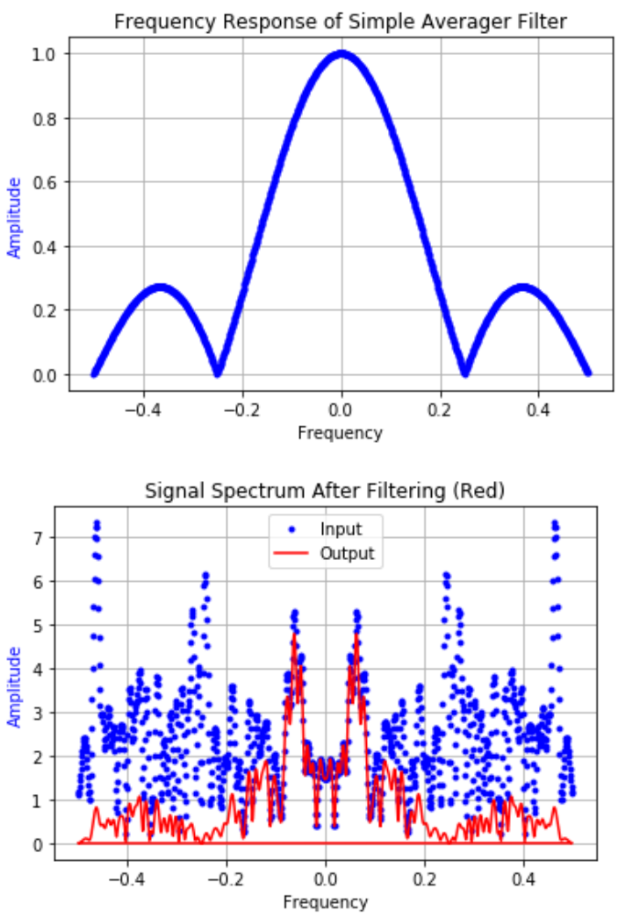

Fig. 2. Frequency Response from the Filter Coefficients (TOP) and the Spectrum for Original Spectrum (blue) and Output Spectrum (red) after filter is applied (BOTTOM)

The overview of this entire process is shown again in Figure 3. Looking at Figure 3 (c), the FFT transforms the impulse response $y(n)$ into the frequency information of $Y(m)$. $Y(m)$ provides the frequency magnitude response after the filter has been applied to the impulse input (also shown as the red line in the bottom graph of Figure 2). This is actually an example of a low-pass filter where the *averager* reduces the amplitude (attenuates) of the high-frequency signal. In our construction of a filter, we will have to use a different impulse response since we want to remove, not attenuate, the high frequency information content. The following section will explain how we design an FIR filter using the FFT and inverse FFT.

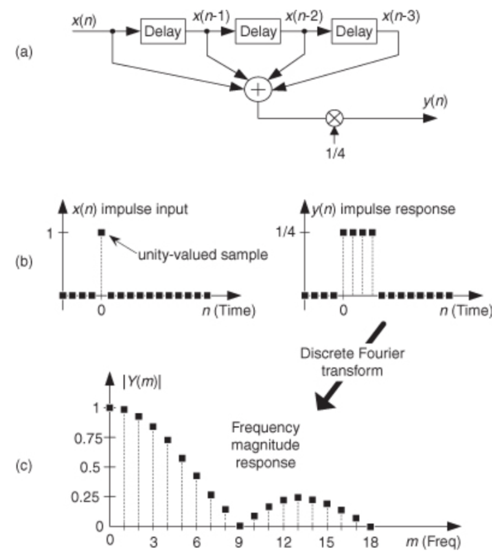

Fig. 3. Block Diagram (a), Impulse Input and Impulse Response (b), and Frequency Magnitude Response after FFT of Impulse Response (c). Taken from Reference [4] below and is Figure 1-12 from Chapter 1

2) Finite Impulse Response (FIR): Now that we have laid down the groundwork with the general overview and a simple example, we will discuss how to create an FIR filter. An FIR filter has a finite duation of nonzero output values given a finite duration of input values (this is how they were named!). Recall, in our example above, we used four "taps" that were all equal to $\frac{1}{4}$. The two factors that affect an FIR filter's frequency response are the number of taps and the coefficients [4]. If we were to calculate the impulse response, $y(n)$, from the impulse input, $x(n)$, in the time-domain, then we would have to perform the mathematical operation of a convolution. More specifically for an M-tap filter, we would calculate $y(n)$ as follows:

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k)$$

Where $h(k)$ are the filter coefficients for each of the taps. **The terms FIR filter coefficients and impulse response mean the same thing** [4]. We can re-write the above equation using *convolution* notation as follows:

$$y(n) = h(k) \star x(n)$$

The major concept is that convolution in the time domain is equal to multiplication in the frequency. The process in which we can move from one domain by using an FFT. The

FFT of $y(n)$ is equal to $Y(m) = H(m)X(m)$, which is the spectrum of the filter output. In our simple averager example above, we used multiplication to create the blue line, $Y(m)$, in the bottom graph of Figure 2. In a similar way, we can determine $y(n) = h(k) \star x(n)$ by taking the inverse FFT (denoted as I-FFT) of $Y(m)$ [4] The FFT and the inverse FFT allow us to move from the time domain to the frequency domain, and from the frequency domain back to the the time domain. To keep track of all the transformations, we listed the important relationships are as follows:

1) $x(n) \xrightarrow{FFT} X(m)$
2) $X(m) \xrightarrow{I-FFT} x(n)$
3) $h(k) \xrightarrow{FFT} H(m)$
4) $H(m) \xrightarrow{I-FFT} h(k)$
5) $y(n) \xrightarrow{FFT} Y(m) \Rightarrow h(k) \star x(n) \xrightarrow{FFT} H(m)X(m)$
6) $Y(m) \xrightarrow{I-FFT} y(n) \Rightarrow H(m)X(m) \xrightarrow{I-FFT} h(k) \star x(n)$

The following section will explain how we design an FIR filter using the GNU Radio Filter Design tool with the window method in order to create a low-pass filter that removes high frequency information content. The process "behind the scenes" is similar to the example with the simple averager.

3) GNU Radio: Filter Design Tool: Now that we understand how to move between the frequency and time-domains using the FFT or inverse FFT, let's design our own low-pass filter by determining the desired frequency response and moving back to the time domain through an inverse FFT to calculate the filter coefficients that will provide the desired frequency response. Recall, in our hearing test example, our project group was unable to hear any frequency information content above 15KHz. As such, we will attempt to design a low-pass filter that removes all of the frequency content above this threshold. Therefore, our *cutoff* frequency is 15KHz (see Figure 1 for a depiction of the cutoff frequency). To design the filter, we define the **frequency cut-off values** (end of passband and beginning of stopband), the **sample rate**, and the **stopband attenuation** (in decibels), and then use a Filter Design Tool within the GNU Radio application (which is written in Python) to apply the inverse FFT function to get the FIR filter coefficients.

We set the sample rate $= 44,100$, the end of the passband at 15 KHz, the beginning of the stopband at 16KHz, and the stopband attenuation at 60 dB. After hitting the "Design" button, the tool creates the frequency response for the desired filter which is displayed in Figure 4. As expected, the frequency response shows a drop-off at 15KHz where the passband ends and the frequencies are cut. The tool will take the inverse FFT of the frequency response in order to generate the filter coefficients, or taps. Figure 5 shows the filter coefficients that provide the frequency response that removes frequency above 15KHz. The filter coefficients are saved and then passed into the GNU Radio Tool that implements the filter created.

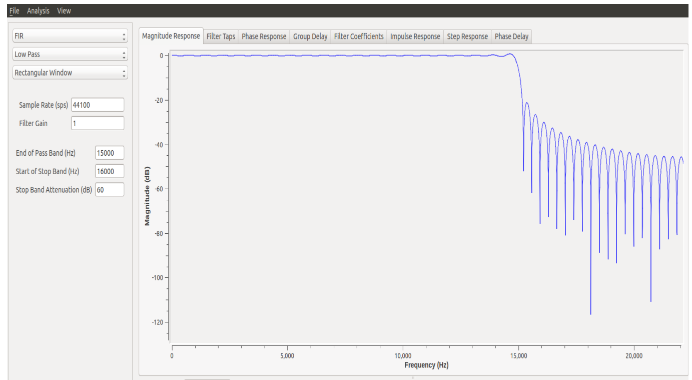

Fig. 4. Designing the 15KHz Filter using the Filter Design Tool within GNU Radio

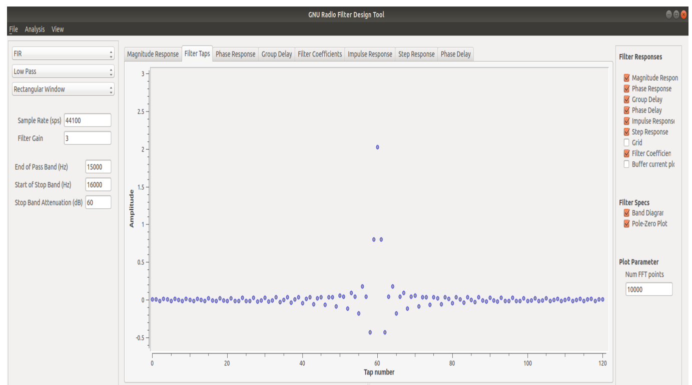

Fig. 5. Filter Coefficients for the 15KHz Filter using the Filter Design Tool within GNU Radio

C. Nyquist-Shannon Sampling Theorem

When recording audio signals, one has to set a sampling rate (rate for analog signal, frequency for discrete signal). There are several reasons why we set a sampling rate:

1) Data storage conservation,
2) Bandwith conservation,
3) Power conservation.

The formula for sampling is:

$$x[n] = x(t)|_{t=nT_S} = x(nT_S) \quad (1)$$

Where T_S is the sampling period and $f_S = \frac{1}{T_S}$ is the sampling frequency [2].

However, we have to be careful not to set this sampling rate too low, or else we run into a problem called aliasing. Aliasing is often called undersampling, and occurs when a different time function with a lower frequency produces the same set of samples [7]. Say we have a signal with the following function (example provided by Baxley et al [8]):

$$x[t] = cos(2\pi 10t) \quad (2)$$

This signal is plotted below:

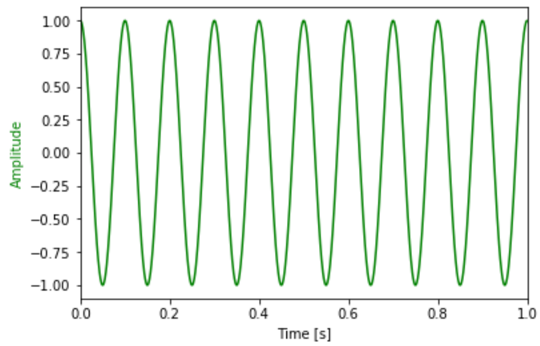

Fig. 6. Original signal, from equation 1 [8].

This is the true function of the signal, but it may be unknown to those wishing to record it. Say we sample this signal at 18 Hz. We get the following alias:

$$x[n] = cos(2\pi\frac{10n}{18}) \tag{3}$$

$$x[n] = cos(10\pi\frac{n}{9}) \tag{4}$$

$$x[n] = cos(10\pi\frac{n}{9} - 2\pi n) \tag{5}$$

$$x[n] = cos(10\pi\frac{n}{9} - \frac{18\pi n}{9}) \tag{6}$$

$$x[n] = cos(-8\pi\frac{n}{9}) \tag{7}$$

$$x[n] = cos(8\pi\frac{n}{9}) \tag{8}$$

Since a full period is 2π, we have aliasing (in our example, we have an aliase of $8\pi\frac{n}{9}$. Our sample rate of 18Hz is too low. Below is our sampling overlaid on the original curve.

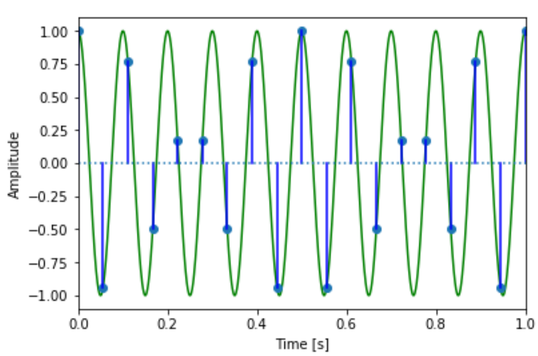

Fig. 7. Overlaid sampling (in blue) on the original curve (in green) 1 [8].

As aforementioned, aliasing means that there are additional functions that can cross through the sampling points of our original signal. In the graph below, notice that a new red curve appears, and that it interesects the blue sampling points like our original green signal.

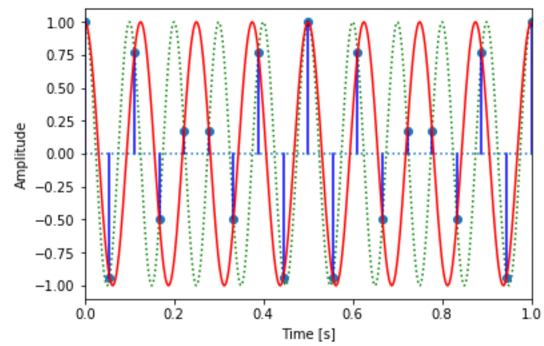

Fig. 8. The aliased signal (in red) intersects the original signal (in green) at the sampling points (in blue) 1 [8].

So, if our objective is to sample as little as possible, how do we determine the minimum rate necessary? For perfect reconstruction (of the original signal from the sample), we need:

$$2f_{max} \leq f_s. \tag{9}$$

This equation says that the frequency rate should be greater than or equal to twice the maximum frequency in the true signal. But, how do we know the maximum frequency of an unknown signal? We use a function called the sinc function, and take a rectangular pulse of the signal. The local maxima and minima of the unnormalized sinc function correspond to its intersections with the cosine function [9]. This means, that we can back-calculate the maximum frequency of an unknown signal through the use of the FFT. Then, we can calculate the sampling rate of this using formula 9.

In our case, before processing with a filter, our raw signal had a sampling rate of 44.1 kHz. This is the standard for CD-quality audio. We know from Equation 9 that the maximum frequency that can be represented at any given sampling rate is half the sampling rate; thus a 44.1 kHz CD can capture tones up to 22.05 kHz. This is often over-kill, since as previously mentioned, humans often cannot hear beyond 15 kHz. We have space to trim that down, but we must make sure to use filters to prevent aliasing.

III. Evaluation of Filtered Signal

We reduced the raw signal from 44.1 kHz to () kHz, with no perceived quality degradation to the three judges.

IV. Conclusion

The conclusion goes here.

References

[1] *GNU Radio*, https://www.gnuradio.org/.

[2] Baxley, R., Henshaw, A., Nowlan, S., & Trewhitt, E. *Software-Defined Radio with GNU Radio: Theory and Application, Georgia Tech Professional Education course notes.* (2017)

[3] Rossing, Thomas (2007). Springer Handbook of Acoustics. Springer. pp. 747, 748. ISBN 978-0387304465.

[4] Lyons, Richard G. Author. "Understanding Digital Signal Processing." Ch. 1, 3, and 5. Web.

[5] https://en.wikipedia.org/wiki/Low-pass_filter. Accessed Nov. 7, 2019.

[6] Harris, Fred J. ”Multirate Signal Processing for Communication Systems.” Page 216, Equation 8.16.

[7] https://en.wikipedia.org/wiki/Nyquist_frequency. Accessed Nov. 15, 2019.

[8] Baxley, R., Henshaw, A., Nowlan, S., & Trewhitt, E. *Juyter Notebook for Sampling: Sampling.ipynb.* (2017)

[9] https://en.wikipedia.org/wiki/Nyquist_frequency. Accessed Nov. 16, 2019.

APPENDIX

A. Python Code for Filter Design [2]

Listing 2. Simple Averager Filter

```
num_samples = 100

# set seed value (optional)
np.random.seed(10)

# generate numpy array of random numbers
#this are the x(n) values
x = np.random.rand(num_samples)-0.5

# filter order
L = 4
h = np.ones(L)/L

plt.figure(11)
plt.stem(h)
plt.xlim([-1,4])
plt.title('Impulse Response - Coefficients of Simple Averager Filter')
N_fft = 1024 #use a power of two

# frequency response
# the second argument of fft is the number of fft bins to use.
freq_response = np.fft.fft(h,N_fft)

# Transform using DFT to get the frequency response
freq = np.fft.fftfreq(freq_response.size)


# plot filter frequency response
plt.figure(1)
plt.plot(freq, abs(freq_response), 'b.');
plt.ylabel('Amplitude', color='b');
plt.xlabel('Frequency');
plt.grid(True);
plt.title('Frequency Response of Simple Averager Filter');


# plot input frequency spectrum by
# using the FFT to convert time to frequency
x_spectrum = np.fft.fft(x,N_fft)

plt.figure(2)
plt.plot(freq, abs(x_spectrum), 'b.', label='Input');
plt.ylabel('Amplitude', color='b');
plt.xlabel('Frequency');
plt.grid(True);
plt.title('Input Signal Frequency Spectrum');

# Plot output spectrum
y = np.convolve(h,x)

#the following two lines provide the same thing
#1) output spectrum by convolution in the time domain and
#using FFT to convert to frequency domain

y_spectrum_convolve = np.fft.fft(y,N_fft)

#2) output spectrum by multiplying in the frequency domain

y_spectrum_multiply = abs(x_spectrum)*freq_response

#plt.plot(freq, abs(y_spectrum), 'r.', label='Output');
plt.plot(freq, abs(y_spectrum_convolve), 'r', label = 'Output')
plt.ylabel('Amplitude');
plt.xlabel('Frequency');
plt.grid(True);
plt.title('Signal Spectrum After Filtering (Red)');
plt.legend();
```