Team Project: Octave Band Filtering

ECE 6530: Digital Signal Processing
December 8, 2023

Tyler Bytendorp   Miguel Gomez   Chase Griswold   Benjamin Hayes

University of Utah Electrical and Computer Engineering

Due Date: Dec 8, 2023

4 Bandpass Filter Design

4.1 Simple Bandpass Filter Design

- a) Generate a bandpass filter that will pass a frequency component at $\omega = 0.4\pi$. Make the filter length $L = 40$ equal to 40. Make a plot of the frequency response magnitude and phase.

- b) Use the plot of the frequency response for the length-40 band-pass filter from part (a), and determine the passband width using the 0.5 level to define the pass band.

- c) Make two other plots of BPFs for L D 20 and L D 80 with the same

b) Table of vals for the filters.

length n	starting freq [rad.]	ending freq [rad.]	bandwidth [rad.]
20	1.0713	1.4483	.37699
40	1.1655	1.3509	.18535
80	1.2095	1.3038	.094248

From these figures, it is seen that passband width is inversely proportional to filter length L. For example, if L increases, bandwidth decreases. From the calculated data, you can see further that the proportionality constant is nearly 1. Meaning that if L is doubled, then the bandwidth is nearly halved, and vice versa.

a and c are shown below in the following plots

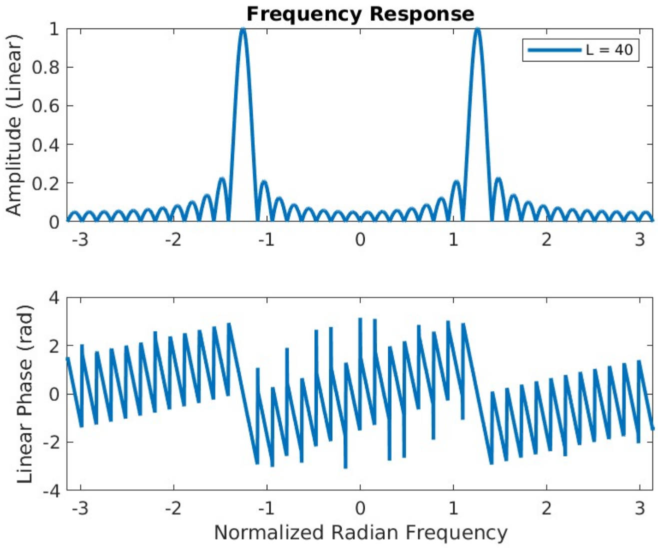

Figure 1: plot for a simple BPF's

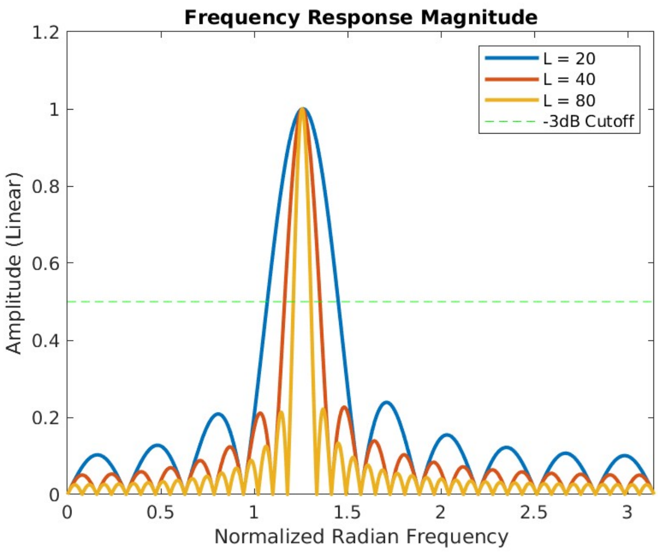

Figure 2: plot for all simple BPF's

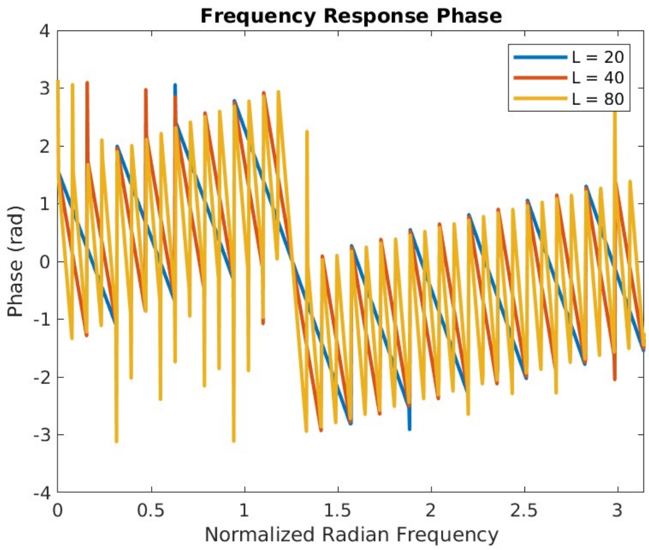

Figure 3: plot for all simple BPF's

4.2 A Better Bandpass Filter Design

a) Generate a Hamming bandpass filter with center frequency $\omega_c = \frac{\pi}{4}$, and filter length $L = 41$. Also determine the Frequency response to a list of specific frequencies. Reference functions for details.
b) Determine the width of the $L = 41$ filter. Then generate 2 more with $L = 21$ and 81. Plot and compare their pass-bands. Again explain the relationship between L and pass-band width.
d)Use the frequency response and passband width to explain how the component at $\omega = \frac{\pi}{4}$ passed and others were rejected.
Looking at the figure, we can see the strong pass-band component completely covering the input signal component at $\frac{\pi}{4}$, indicating alot of that power will pass thought while all outside will be rejected. From the calculations earlier, the passband for this filter goes from about 0.2π to $.29\pi$. The only frequency component in this range is 0.25π, which is amplified. All other frequency components are far from this range, and the hamming window stopband is very low, causing frequencies far from the passband range to be heavily attenuated.

See code for worked out bits by hand for part C and the rest of the outputs in html. code can be found in in appendix A.

5 Detecting Octave Bands

5.1 Piano Octaves

This bit was taken care of in Matlab and this is the function to set up the filter bands table. The results are below and a copy of the code is included in appendix A: Since we have a limit on the bands we can recognize that arises from the use of the sampling freq of $8kHz$, we can only obtain unique detection for the set of Octaves whose frequencies are below the Nyquist rate of $\frac{fs}{2}$ or $4kHz$. Or, O_7 in the table above.

val [units]	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8
Lower (Hz)	32.703	65.406	130.81	261.63	523.25	1046.5	2093	4186
Lower (Rad)	0.025685	0.05137	0.10274	0.20548	0.41096	0.82192	1.6438	3.2877
Upper (Hz)	61.735	123.47	246.94	493.88	987.77	1975.5	3951.1	7902.1
Upper (Rad)	0.048487	0.096974	0.19395	0.3879	0.77579	1.5516	3.1032	6.2063
Center (Hz)	47.219	94.439	188.88	377.75	755.51	1511	3022	6044.1
Center (Rad)	0.037086	0.074172	0.14834	0.29669	0.59338	1.1868	2.3735	4.747

Table 1: Frequency Ranges for Octaves 1 to 8 starting with O_1

6 Band-pass Filter Bank Design

Comment on the selectivity of the bandpass filters, i.e., use the frequency response (passbands and stopbands) to explain how the filter passes one octave while rejecting the others. Are the filters passbands narrow enough so that only one octave lies in the passband and the others are in the stop-band? The band pass filter bank was put together in python and has the added benefit of being more portable than the use of Matlab for this application. These are the results from the creation of the x signal needed to pass on to the filter bank. Defining the expression below:

$$x(t) = \begin{cases} \cos(2\pi \cdot 220t) & 0.0 < t \leq 0.25, \\ \cos(2\pi \cdot 880t) & 0.3 < t \leq 0.55, \\ \cos(2\pi \cdot 440t) + \cos(2\pi \cdot 880t) & 0.60 < t \leq 0.85 \\ 0 & \text{otherwise} \end{cases}$$

and now the code to do this along with the plot showing the data is what we expect:

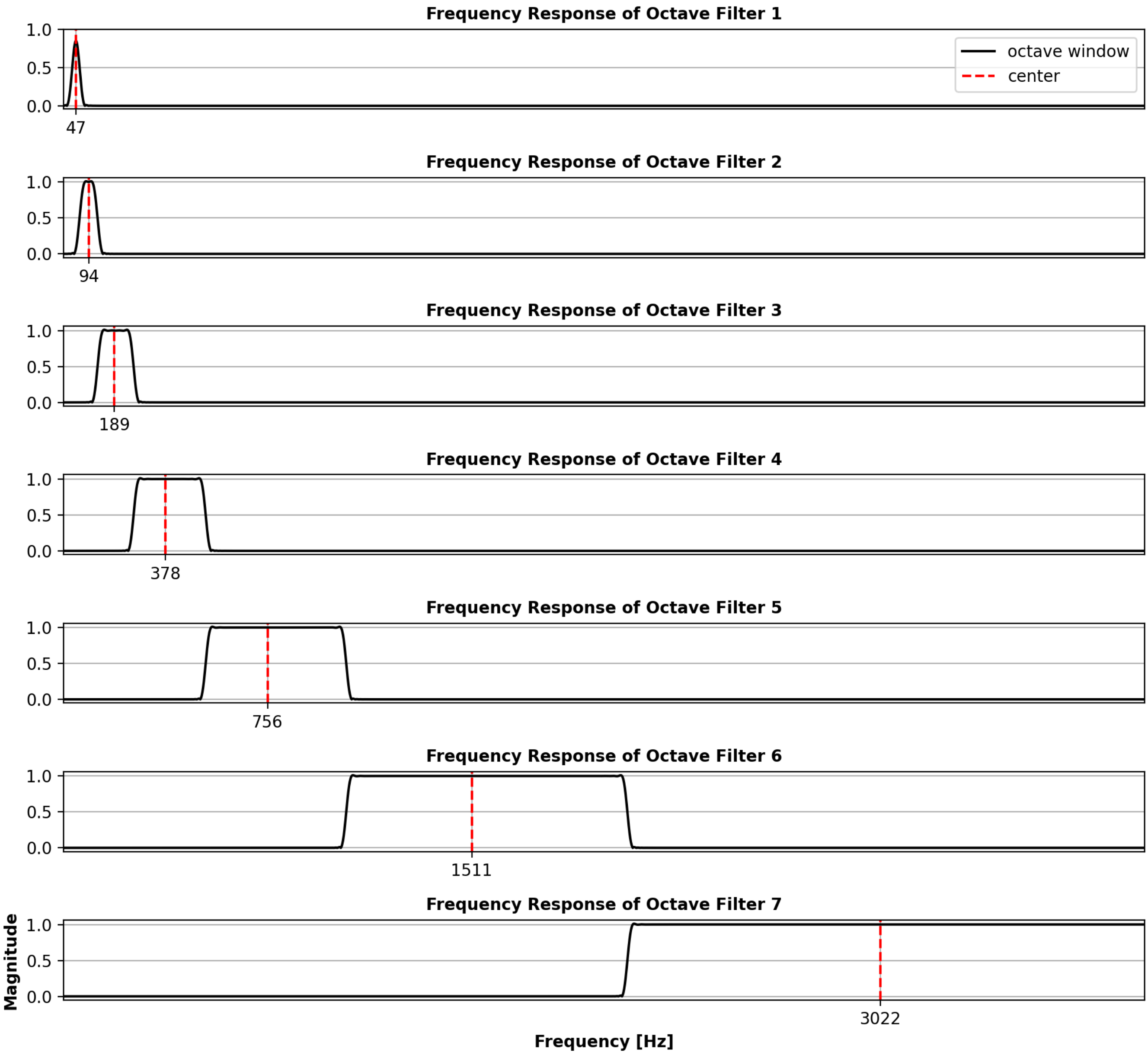

Figure 4: plot for all BPF's

The filters above in fig. 4 begin at a center of $47Hz$ but we only need to worry about the subsequent ones that we were asked for. Aligned at their centers is a vertical line and a tick mark indicating the center frequency. this bank of filters is what we will use to pass the time data created in the equation above. Filters one and two do not pass anything since they are both too low to capture the first freq of $220Hz$, but the third has a center of $189Hz$, meaning it does capture the first frequency. In all cases, we can see that the transition points have the characteristic peaks associated with instant changes in frequency. Any instant change results in

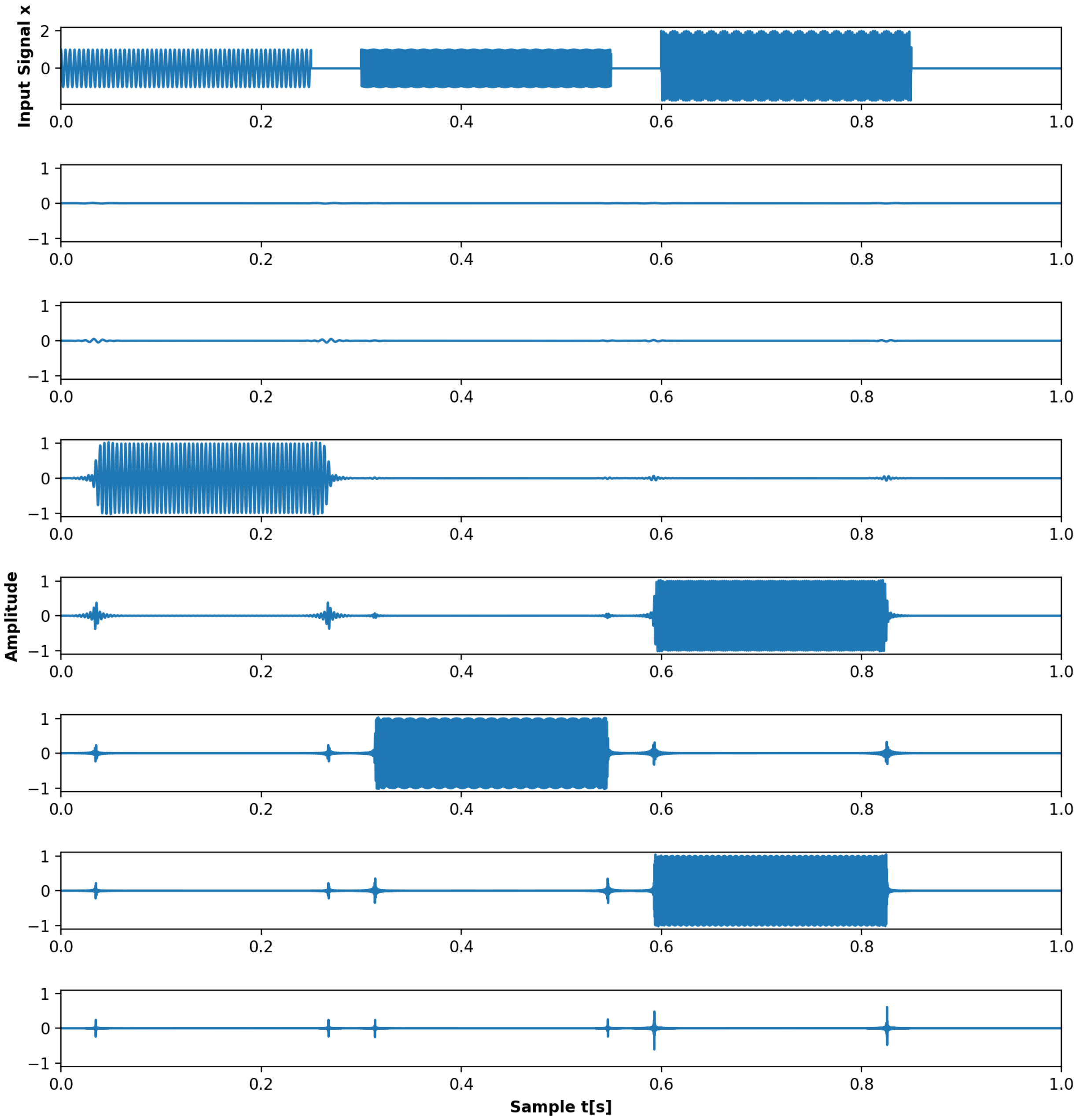

Figure 5: plot for all BPF's processing the time data at the top

a sharp change which corresponds to an infinite number of frequencies. the third filter is capable of capturing the $880Hz$, the fourth captures the $1760Hz$ in superposition with the $880Hz$, and no signal is passed through the last except the transients that appear in all filters.

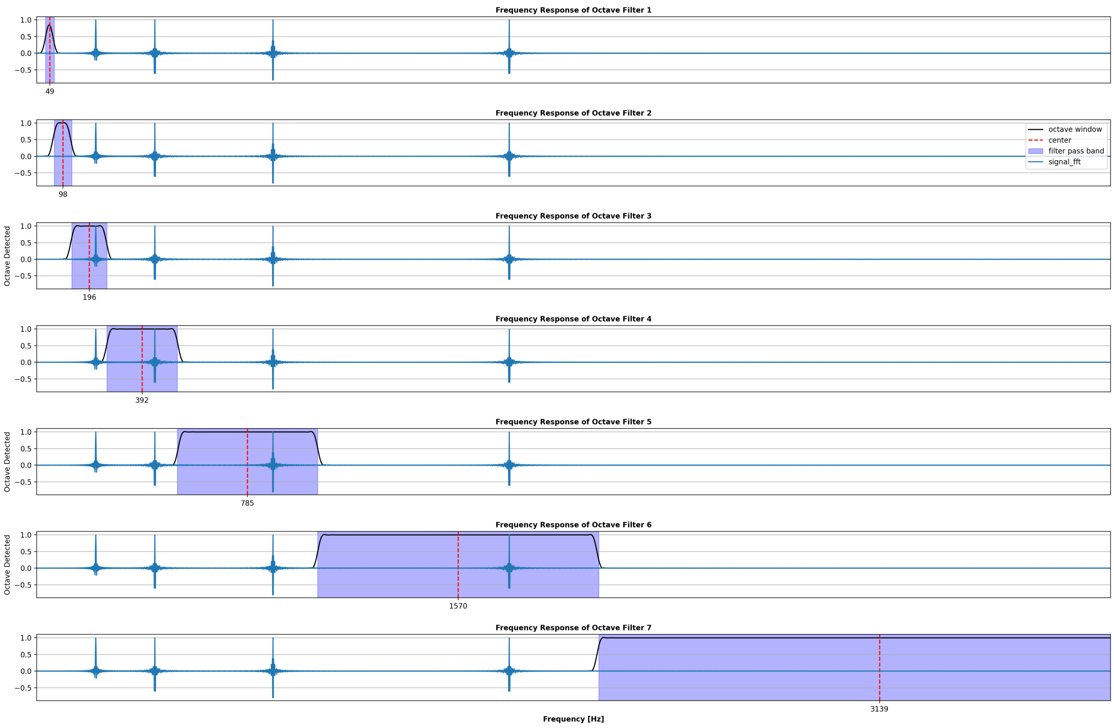

Figure 6: plot for all BPF's bandpass regions

In this figure, it is easier to see that the band-pass filters correctly capture the frequencies that we see in the time plot of fig. 5. In the figure above fig. 6, we can see that the filters that do not capture any signal do not have an overlapping note peak in the shaded region. However, the ones that we see with a signal passed through them do have a corresponding note in the shaded band-pass region.

6.1 Creating the FIR Filters

To create the FIR filters, we used a python class shown in subsection 7.2. The class works by first defining N, the number of points to be used in the FIR filter. We then declare the pass band region, the frequencies we wish for the FIR filter to pass through while the others are rejected. This is shown in fig. 7. Effectively, we are specifying the desired frequency response of the FIR filter but only using N number of points to define the filter in the frequency domain.

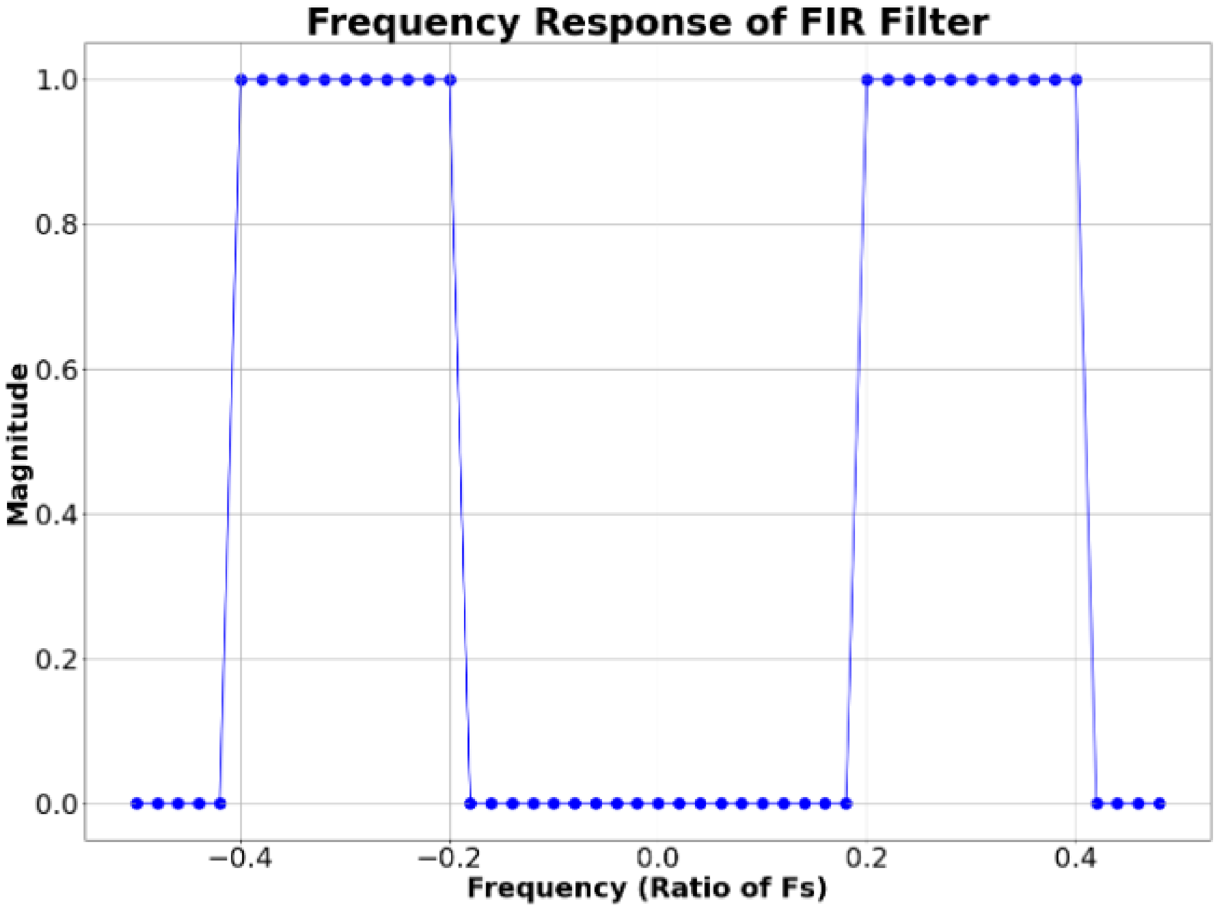

Figure 7: Band-pass region of the FIR filter being created. The filter response is specified by N number of points where N is the length of the FIR filter being created.

After defining the frequency response with N number of points, we need to also ensure that the frequency response is symmetric about the y-axis. This ensures that our FIR filter will be purely real. To obtain the time domain FIR filter, we then take the inverse fourier transform of the N number of points specifying the frequency response. The FIR filter corresponding to the frequency response shown in 7 above.

7 Appendix A

7.1 Matlab Code

```
function print_octaves(n,fs)
% n should be a range relative to A4. ex: -1:1 gives A3,A4,A5
A4 = 440;
C4 = A4.*2.^(-9./12);
B4 = A4.*2.^(2./12);
Octaves = 2.^n;
Cs = C4.*Octaves;
Bs = B4.*Octaves;
n_range = 0:length(n)-1;
Centers = (Cs + Bs)./2;
cell(size(Centers));
octave_array = arrayfun(@(x) sprintf('Octave %d', x), n_range, 'UniformOutput',
    false);
w_Cs = Cs.*2.*pi./fs;
w_Bs = Bs.*2.*pi./fs;
w_Centers = Centers.*2.*pi./fs;
rows = {'Lower (Hz)','Lower (Rad)','Upper (Hz)','Upper (Rad)','Center (Hz)','Center
     (Rad)'};
% Summarize data in a table
T = array2table([Cs; w_Cs; Bs; w_Bs; Centers; w_Centers],'VariableNames',
    octave_array,'RowName',rows);
disp(T)
disp('Hz are not normalized.')
disp('Radians are normalized by sampling frequency.')
end
```

7.2 Python FIR Filter Class

```python
import numpy as np
from numpy import zeros, append
from numpy.fft import fftshift, fft
import matplotlib.pyplot as plt


class FIRFilter:
    def __init__(self, N=10000, fmin=3, fmax=7, padding_factor=9, fs=8000):
        self.N = N
        self.padding_factor = padding_factor
        self.fs = fs  # Sampling rate
        self.H = zeros(N)
        self.w = zeros(N)
        self.pos = np.arange(N)
        self.fmin = fmin*self.N/self.fs
        self.fmax = fmax*self.N/self.fs

        self.h = None
        self.h_pad = None
        self.H_pad = None
        self.w_pad = None
        self.h_ham = None
        self.H_ham_pad = None

        self.create_filter()
        self.apply_padding()
        self.apply_hamming_window()


    def create_filter(self):
        k = np.arange(-int(self.N/2), int(self.N/2))
        self.w = k * self.fs / self.N  # Adjusted to use actual frequency values
        self.H = np.where((np.abs(k) >= self.fmin) & (np.abs(k) <= self.fmax), 1, 0)
        self.h = fftshift(fft(fftshift(self.H)))

    def apply_padding(self):
        NP = self.N + self.padding_factor * self.N
        self.h_pad = append(self.h, zeros(self.padding_factor * self.N))
        self.H_pad = fftshift(fft(self.h_pad)) / self.N
        k = np.arange(-NP/2, NP/2)
        self.w_pad = k * self.fs / NP

    def apply_hamming_window(self):
        self.h_ham = self.h * 0.5 * (1 + np.cos(2 * np.pi * (self.pos - self.N / 2) / self.N))
        self.h_ham_pad = append(self.h_ham, zeros(self.padding_factor * self.N))
        self.H_ham_pad = fftshift(fft(self.h_ham_pad)) / self.N

    def process(self, input_data):
        return np.convolve(input_data, self.h_ham)/self.N

    def plot_filter(self, fig, row, col, pos):
        # Frequency Response Plot
        ax1 = fig.add_subplot(row, col, pos)
        # ax1.scatter(self.w, self.H.real, c='b', s=150)
        # ax1.plot(self.w_pad, abs(self.H_pad), 'r')
        ax1.plot(self.w_pad, abs(self.H_ham_pad), 'black')
        #ax1.set_xlim(0, .5)
        ax1.set_title(f'Frequency Response of Octave Filter {pos}', fontsize=5,
```

```
fontweight='bold')
        if pos == row-1:
            ax1.legend(['Hamming'], prop={'size': 5})
            ax1.set_xlabel('Frequency (Ratio of Fs)', fontsize=5, fontweight='bold'
)
            ax1.set_ylabel('Magnitude', fontsize=5, fontweight='bold')
        ax1.grid(True)

    def plot_filter1(self):
        # MatPlotLib plotting
        fig = plt.figure(figsize=(22, 16))

        # Frequency Response Plot
        ax1 = fig.add_subplot(211)
        ax1.scatter(self.w, self.H.real, c='b', s=150)
        ax1.plot(self.w_pad, abs(self.H_pad), 'r')
        ax1.plot(self.w_pad, abs(self.H_ham_pad), 'black')
        ax1.set_xlabel('Frequency (Hz)', fontsize=15, fontweight='bold')
        ax1.set_ylabel('Magnitude', fontsize=15, fontweight='bold')
        ax1.set_title('Frequency Response of FIR Filter', fontsize=15, fontweight='
bold')
        ax1.legend(['Ideal', 'Actual', 'Hamming'], prop={'size': 15})
        ax1.tick_params(axis='both', labelsize=15)
        ax1.grid(True)

        # Time Domain Plot
        ax2 = fig.add_subplot(212)
        ax2.vlines(self.pos, 0, self.h.real, 'b')
        # ax2.vlines(self.pos, 0, self.h.imag, 'r')
        ax2.scatter(self.pos, self.h.real, c='b', s=150)
        # ax2.scatter(self.pos, self.h.imag, c='r', s=150)
        ax2.set_xlabel('Position', fontsize=15, fontweight='bold')
        ax2.set_ylabel('Value (Unscaled)', fontsize=15, fontweight='bold')
        ax2.set_title('Time Domain FIR Filter', fontsize=15, fontweight='bold')
        # ax2.legend(['Real', 'Imag'], prop={'size': 15})
        ax2.tick_params(axis='both', labelsize=15)
        ax2.grid(True)

        plt.show(block=False)
```

7.3 Python Code - Generate Plots

```python
from util.OctaveBandFilt import OctaveBandFilter, ourOctaveBandFilter
from util.FIR_filter import FIRFilter
import numpy as np
import matplotlib.pyplot as plt
from rich import print
import math

fontsize = 10

def calculate_grid_dimensions(n):
    columns = round(math.sqrt(n))
    rows = math.ceil(n / columns)
    return rows, columns

def calculate_octave_ranges(base_freq, num_octaves, sample_rate, N):
    octave_ranges = []
    for i in range(num_octaves+1):
        fmin = base_freq * (2 ** i)  # Minimum frequency for the octave
        if N < 500:
            fmax = base_freq * (2 ** (i+(11/12)))
        else:
            fmax = base_freq * (2 ** (i+1)) # Maximum frequency for the octave

        # Normalize the frequencies
        fmin_normalized = fmin
        fmax_normalized = fmax

        octave_ranges.append((fmin_normalized, fmax_normalized))
    return octave_ranges


# Create a set of filters
N = 600
sampling_frequency = 8000  # Sampling frequency in Hz
# Sample rate and base frequency
base_freq = 32.703  # Starting frequency of the first octave
# Calculate octave ranges
num_octaves = 6
# for the filtering
octave_ranges = calculate_octave_ranges(base_freq, num_octaves, sampling_frequency,
     2*N)
# for the midpoints
octave_ranges1 = calculate_octave_ranges(base_freq, num_octaves, sampling_frequency
    , N/2)


filters = [FIRFilter(N, fmin=fmin, fmax=fmax, padding_factor=9, fs=
    sampling_frequency) for fmin, fmax in octave_ranges]
num_filters = len(filters)
#filters = [FIRFilter(N, fmin=fmin*N, fmax=fmax*N, padding_factor=9) for fmin, fmax
     in octave_ranges]
print(f"octave ranges: {octave_ranges}")
# Calculate grid size
total_subplots = num_filters
rows, cols = calculate_grid_dimensions(total_subplots)
print(f"rows: {rows}")
print(f"cols: {cols}")
# Create a figure for plotting
fig = plt.figure(figsize=(10, 10))

```

```python
mids = [(b+a)/2 for a, b in octave_ranges]
mids1 = [(b+a)/2 for a, b in octave_ranges1]
print(f"midpoints_overlap: {mids}")
print(f"midpoints_no_overlap: {mids1}")
# Plot each filter's response
for i, filter in enumerate(filters):
    # Calculate position
    position = i * 2 + 1  # Position for frequency response plot
    x = mids1[i]
    filter.plot_filter(fig, num_filters+1, 1, i+1)
    plt.axvline(x=x, ymin=0, ymax=1, color='r', linestyle='--')
    if i == 0:
        plt.legend(['octave window', 'center'])
    plt.xticks([x])
    plt.yticks([0, .5, 1])
    ax = plt.gca()
    ax.set_xlim([0, filter.fs/2])

plt.tight_layout()
plt.show(block=False)
plt.savefig("images/post_lock_freq_data.png", dpi=200, transparent=False)
plt.close()

num_octaves = 6
octave_ranges = calculate_octave_ranges(base_freq, num_octaves, sampling_frequency,
    N)
filters = [FIRFilter(N, fmin=fmin, fmax=fmax, padding_factor=9, fs=
    sampling_frequency) for fmin, fmax in octave_ranges]
num_filters = len(filters)
# Generate the signal 5.2
fs = sampling_frequency
t1 = np.linspace(0, 0.25, int(fs*0.25), endpoint=False)
t2 = np.linspace(0.3, 0.55, int(fs*0.25), endpoint=False)
t3 = np.linspace(0.6, 0.85, int(fs*0.25), endpoint=False)
t_end = np.linspace(0.85, 1, int(fs*0.15), endpoint=False)

x1 = np.cos(2*np.pi*220*t1)
x2 = np.cos(2*np.pi*880*t2)
x3 = np.cos(2*np.pi*440*t3) + np.cos(2*np.pi*1760*t3)

zero_padding = np.zeros(int(fs*0.05))
zeros_end = np.zeros(int(fs*0.15))
x = np.concatenate((x1, zero_padding, x2, zero_padding, x3, zeros_end))
t = np.linspace(0, 1, len(x), endpoint=False)
# Filter and plot the signal
fig = plt.figure(figsize=(10, 10))
plt.title('Filtered Signal with Filters', fontsize=fontsize+10, fontweight='bold')
plt.subplot(len(filters)+1, 1, 1)
plt.plot(t, x)
plt.ylabel('Input Signal x', fontsize=fontsize, fontweight='bold')
ax = plt.gca()
ax.set_xlim([0, 1])

for i, filter in enumerate(filters):
    filtered_x = filter.process(x)
    t = np.linspace(0, 1, len(filtered_x), endpoint=False)
    filtered_sig = filtered_x
    plt.subplot(len(filters)+1, 1, i+2)
    plt.plot(t, np.real(filtered_sig))
    ax = plt.gca()
    ax.set_ylim([-1.100, 1.100])
    ax.set_xlim([0, 1])
```

```
    plt.tight_layout(pad=2.0)
    plt.ylabel('Amplitude', fontsize=fontsize, fontweight='bold') if i == (len(
    filters)//2) else None


plt.xlabel('Sample t[s]', fontsize=fontsize, fontweight='bold')
plt.show(block=False)
plt.savefig("images/post_lock_time_data.png", dpi=200, transparent=False)
plt.close()


fig = plt.figure(figsize=(22, 16))
x_fftd = np.fft.fft(x)

# Plot each filter's response
for i, filter in enumerate(filters):
    # Calculate position
    position = i * 2 + 1  # Position for frequency response plot
    x = mids[i]
    # Determine the width of the rectangle from the octave range
    fmin, fmax = octave_ranges[i]
    rect_width = fmax - fmin
    filter.plot_filter(fig, num_filters+1, 1, i+1)
    plt.axvline(x=x, ymin=0, ymax=1, color='r', linestyle='--')

    # Adding the shaded rectangle
    plt.axvspan(x - rect_width/2, x + rect_width/2, ymin=0, ymax=1, alpha=0.3,
    color='blue')

    plt.plot(x_fftd/max(x_fftd))
    plt.legend(['octave window', 'center', 'filter pass band', 'signal_fft'],
    fontsize=10, loc='upper right', bbox_to_anchor=(1, 1)) if i == 1 else None
    plt.xticks([x])
    ax = plt.gca()
    ax.set_xlim([0, filter.fs/2])
    ax.set_ylabel("Octave Detected", fontsize=10) if i >= 2 and i < 6 and i != 3
    else ax.set_ylabel(" ", fontsize=10)
    plt.tight_layout(pad=2.0)


plt.show(block=False)
plt.savefig("images/post_lock_fft_freqdata.png", dpi=200, transparent=False)
plt.close()
```