Team Project: Octave Band Filtering

ECE 6530: Digital Signal Processing
December 5, 2023

Tyler Bytendorp   Miguel Gomez   Chase Griswold   Benjamin Hayes

University of Utah Electrical and Computer Engineering

Due Date: Dec 5, 2023

5.1 Octave Bands

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

val [units]	O_0	O_1	O_2	O_3	O_4	O_5	O_6
Lower (Hz)	65.406	130.81	261.63	523.25	1046.5	2093	4186
Lower (Rad)	0.05137	0.10274	0.20548	0.41096	0.82192	1.6438	3.2877
Upper (Hz)	123.47	246.94	493.88	987.77	1975.5	3951.1	7902.1
Upper (Rad)	0.096974	0.19395	0.3879	0.77579	1.5516	3.1032	6.2063
Center (Hz)	94.439	188.88	377.75	755.51	1511	3022	6044.1
Center (Rad)	0.074172	0.14834	0.29669	0.59338	1.1868	2.3735	4.747

Table 1: Frequency Ranges for Octaves 0 to 6 starting with C_2

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_5 in the table above.

5.2 Octave Filter Bank

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 filter's passbands narrow enough so that only one octave lies in the passband and the others are in the stop-band?
Comment here and then do the next bit. need a plot from 5.2 that Tyler made. our beta is 1
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:

$$x(t) = \begin{cases} \cos\left(440\pi t\right) & 0.0 < t \leq 0.25, \\ \cos\left(1760\pi t\right) & 0.3 < t \leq 0.55, \\ \cos\left(880\pi t\right) + \cos\left(1760\pi t\right) & 0.60 < t \leq 0.85 \end{cases}$$

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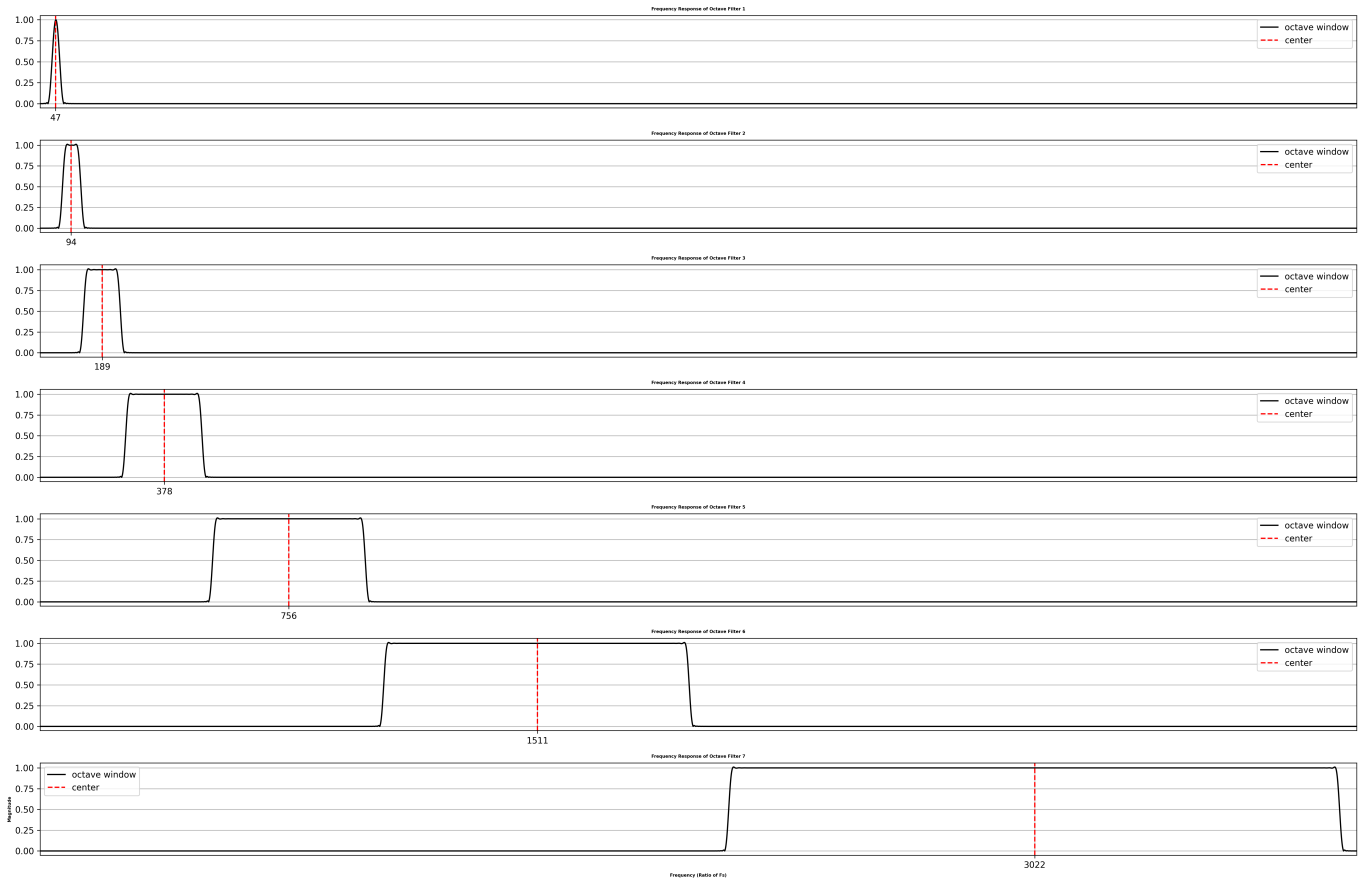

Figure 1: plot for all BPF's

The filters above in fig. 1 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.
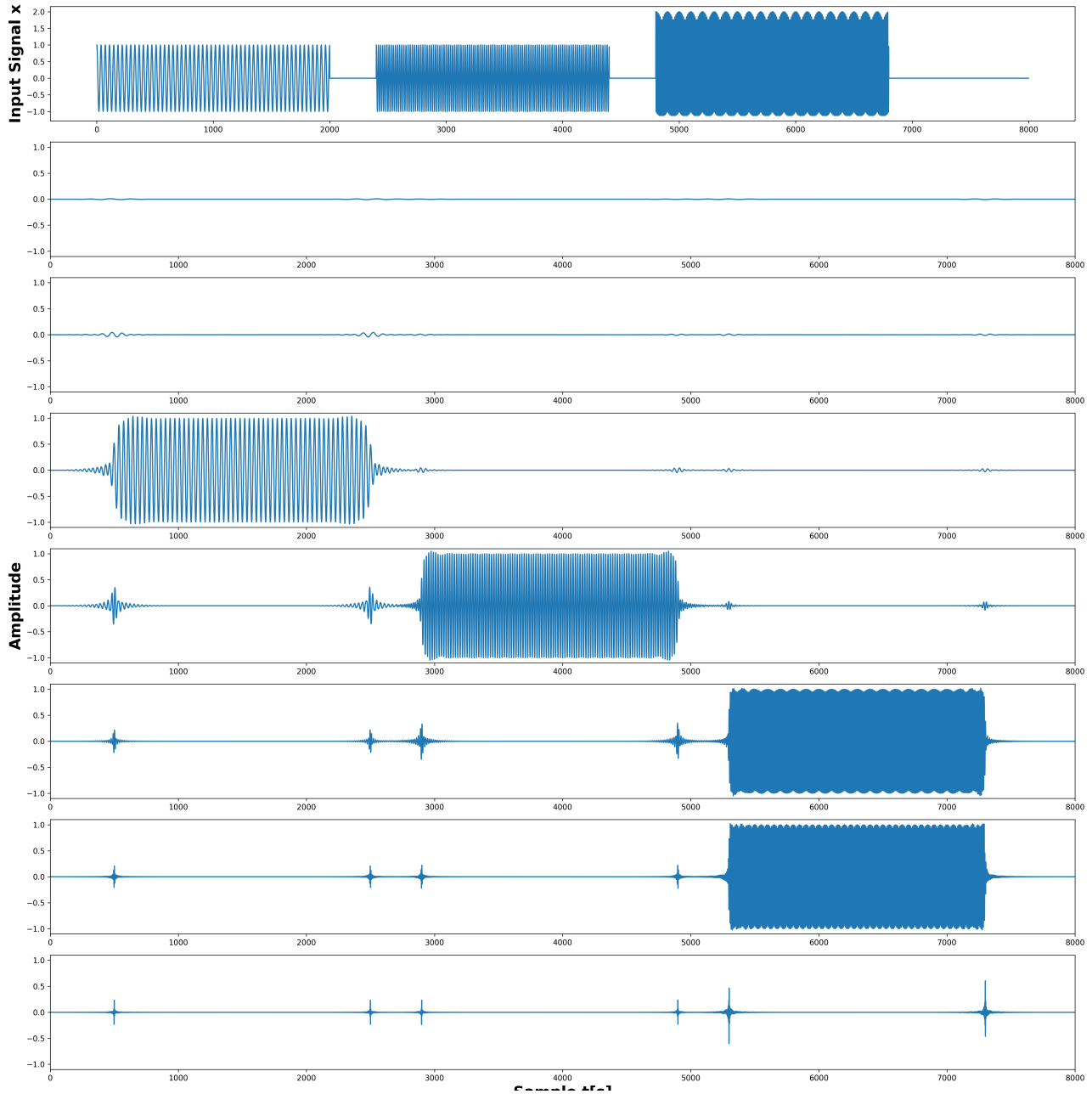
Figure 2: plot for all BPF's processing the time data at the top

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 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.
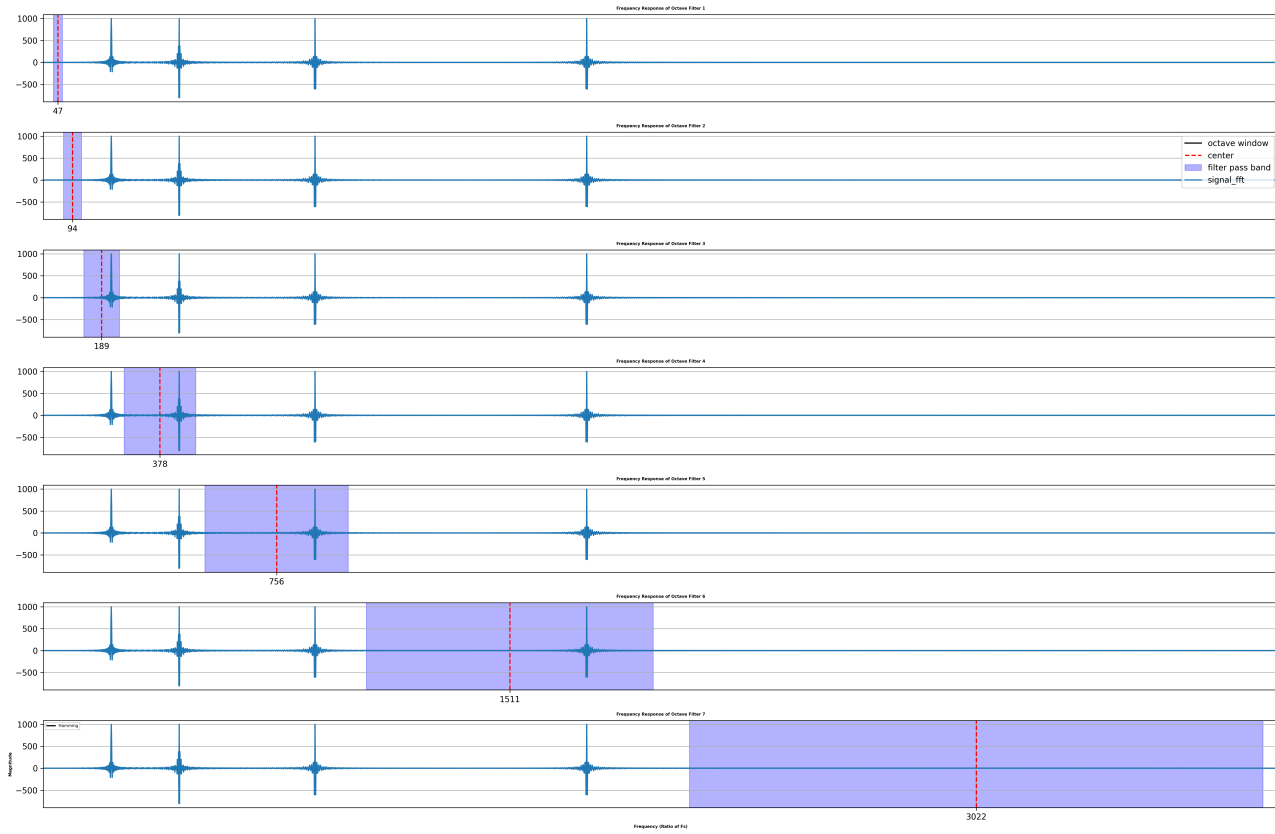
Figure 3: plot for all BPF's bandpass regions

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. 2. In the figure above fig. 3, 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.

Appendix A

0.1 Matlab Code

```matlab
1  function print_octaves(n,fs)
2  % n should be a range relative to A4. ex: -1:1 gives A3,A4,A5
3  A4 = 440;
4  C4 = A4.*2.^(-9./12);
5  B4 = A4.*2.^(2./12);
6  Octaves = 2.^n;
7  Cs = C4.*Octaves;
8  Bs = B4.*Octaves;
9  n_range = 0:length(n)-1;
10 Centers = (Cs + Bs)./2;
11 cell(size(Centers));
12 octave_array = arrayfun(@(x) sprintf('Octave %d', x), n_range, 'UniformOutput',
       false);
13 w_Cs = Cs.*2.*pi./fs;
14 w_Bs = Bs.*2.*pi./fs;
15 w_Centers = Centers.*2.*pi./fs;
16 rows = {'Lower (Hz)','Lower (Rad)','Upper (Hz)','Upper (Rad)','Center (Hz)','Center
        (Rad)'};
17 % Summarize data in a table
18 T = array2table([Cs; w_Cs; Bs; w_Bs; Centers; w_Centers],'VariableNames',
       octave_array,'RowName',rows);
19 disp(T)
20 disp('Hz are not normalized.')
21 disp('Radians are normalized by sampling frequency.')
22 end
```

0.2 Python Code

```python
1  from util.OctaveBandFilt import OctaveBandFilter, ourOctaveBandFilter
2  from util.FIR_filter import FIRFilter
3  import numpy as np
4  import matplotlib.pyplot as plt
5  from rich import print
6  import math
7
8  fontsize = 20
9
10 def calculate_grid_dimensions(n):
11     columns = round(math.sqrt(n))
12     rows = math.ceil(n / columns)
13     return rows, columns
14
15
16 def calculate_octave_ranges(base_freq, num_octaves, sample_rate):
17     octave_ranges = []
18     for i in range(num_octaves+1):
19         fmin = base_freq * (2 ** i)  # Minimum frequency for the octave
20         fmax = base_freq * (2 ** ((i*12+11)/12))  # Maximum frequency for the
       octave
21
22         # Normalize the frequencies
23         fmin_normalized = fmin
24         fmax_normalized = fmax
25
26         octave_ranges.append((fmin_normalized, fmax_normalized))
27     return octave_ranges
```

```python
def windowed_sinc(filter_length, low_freq, high_freq):
    n = np.arange(filter_length)
    mid = (filter_length - 1) / 2
    window = np.hamming(filter_length)

    # Debugging: Print frequency values
    print("low_freq:", low_freq)
    print("high_freq:", high_freq)

    sinc_low = np.sinc(2 * low_freq * (n - mid))
    sinc_high = np.sinc(2 * high_freq * (n - mid))
    bp_filter = sinc_high - sinc_low

    # Debugging: Print sinc function outputs
    print("sinc_low sample values:", sinc_low[:5])
    print("sinc_high sample values:", sinc_high[:5])

    bp_filter *= window
    bp_filter_sum = np.sum(bp_filter)

    # Debugging: Check for zero sum
    print("bp_filter sum after window:", bp_filter_sum)
    if bp_filter_sum == 0:
        bp_filter_sum = 1  # To avoid division by zero

    bp_filter /= bp_filter_sum

    # Debugging: Final check
    print("Final bp_filter sample values:", bp_filter[:5])
    return bp_filter


# Example usage
filter_length = 150  # A typical length for FIR filters
center_frequency = 220  # Center frequency in Hz
sampling_frequency = 12000  # Sampling frequency in Hz

# Recreating the OctaveBandFilter instance with the updated class definition
octave_filter = OctaveBandFilter(filter_length, center_frequency,
    sampling_frequency)
octave_filter.calculate_coefficients()


# Recreating the OctaveBandFilter instance with the updated class definition
ourOctave_filter = ourOctaveBandFilter(filter_length, center_frequency,
    sampling_frequency, windowed_sinc)
ourOctave_filter.calculate_coefficients()
print("Coefficients:", ourOctave_filter.coefficients)
# Generating a test signal - a simple sine wave at the center frequency
test_signal = np.sin(2 * np.pi * center_frequency / sampling_frequency * np.arange
    (0, sampling_frequency))

# Applying the filter to the test signal
filtered_signal = octave_filter.apply_filter(test_signal)
# Applying the filter to the test signal
ourFiltered_signal = ourOctave_filter.apply_filter(test_signal)


def generate_fm_signal(sampling_frequency, duration, start_freq, end_freq):
    """
```

```python
87          Generate a frequency modulated (FM) signal.
88          """
89          t = np.arange(0, duration, 1/sampling_frequency)  # Time vector
90          instantaneous_frequency = np.linspace(start_freq, end_freq, len(t))
91          phase = 2 * np.pi * np.cumsum(instantaneous_frequency) / sampling_frequency
92          signal = np.sin(phase)
93          return signal
94
95
96     # Parameters for the OctaveBandFilter
97     filter_length = 150
98     center_frequency = 440  # Center frequency in Hz
99     sampling_frequency = 8000  # Sampling frequency in Hz
100    # Lambda function for windowed sinc
101
102
103
104    # Creating the filter instance and calculating coefficients
105    Octave_filter = OctaveBandFilter(filter_length, center_frequency,
           sampling_frequency)
106    Octave_filter.calculate_coefficients()
107    # Creating the filter instance and calculating coefficients
108    ourOctave_filter = ourOctaveBandFilter(filter_length, center_frequency,
           sampling_frequency, windowed_sinc)
109    ourOctave_filter.calculate_coefficients()
110
111    # Generating an FM signal
112    duration = 5  # 1 second duration
113    start_freq = 1  # Starting frequency of 0 Hz
114    end_freq = 4 * ourOctave_filter.center_frequency  # Ending at twice the center
           frequency of the filter
115    fm_signal = generate_fm_signal(sampling_frequency, duration, start_freq, end_freq)
116
117    # Filtering the FM signal
118    filtered_fm_signal = Octave_filter.apply_filter(fm_signal)
119    # Filtering the FM signal
120    ourFiltered_fm_signal = ourOctave_filter.apply_filter(fm_signal)
121
122    # Sample rate and base frequency
123    base_freq = 32.703*1  # Starting frequency of the first octave
124    # Calculate octave ranges
125    num_octaves = 6
126    octave_ranges = calculate_octave_ranges(base_freq, num_octaves, 8000)
127
128
129    # Create a set of filters
130    N = 1000
131    filters = [FIRFilter(N, fmin=fmin, fmax=fmax, padding_factor=9, fs=8000) for fmin,
           fmax in octave_ranges]
132    num_filters = len(filters)
133    #filters = [FIRFilter(N, fmin=fmin*N, fmax=fmax*N, padding_factor=9) for fmin, fmax
             in octave_ranges]
134    print(f"octave ranges: {octave_ranges}")
135    # Calculate grid size
136    total_subplots = num_filters
137    rows, cols = calculate_grid_dimensions(total_subplots)
138    print(f"rows: {rows}")
139    print(f"cols: {cols}")
140    # Create a figure for plotting
141    fig = plt.figure(figsize=(22, 16))
142
143    mids = [(b+a)/2 for a, b in octave_ranges]
```

```python
144 print(mids)
145 # Plot each filter's response
146 for i, filter in enumerate(filters):
147     # Calculate position
148     position = i * 2 + 1  # Position for frequency response plot
149     x = mids[i]
150     filter.plot_filter(fig, num_filters+1, 1, i+1)
151     plt.axvline(x=x, ymin=0, ymax=1, color='r', linestyle='--')
152     plt.legend(['octave window', 'center'])
153     plt.xticks([x])
154     ax = plt.gca()
155     ax.set_xlim([0, filter.fs/2])
156
157 plt.tight_layout()
158 plt.show(block=False)
159 plt.savefig("freq_data_2.png", dpi=300, transparent=False)
160 plt.close()
161 num_octaves = 6
162 octave_ranges = calculate_octave_ranges(base_freq, num_octaves, 1000)
163 filters = [FIRFilter(N, fmin=fmin, fmax=fmax, padding_factor=9, fs=8000) for fmin,
        fmax in octave_ranges]
164 num_filters = len(filters)
165 # Generate the signal 5.2
166 fs = 8000
167 t1 = np.linspace(0, 0.25, int(fs*0.25), endpoint=False)
168 t2 = np.linspace(0.3, 0.55, int(fs*0.25), endpoint=False)
169 t3 = np.linspace(0.6, 0.85, int(fs*0.25), endpoint=False)
170 t_end = np.linspace(0.85, 1, int(fs*0.15), endpoint=False)
171
172 x1 = np.cos(2*np.pi*220*t1)
173 x2 = np.cos(2*np.pi*440*t2)
174 x3 = np.cos(2*np.pi*880*t3) + np.cos(2*np.pi*1760*t3)
175
176 zero_padding = np.zeros(int(fs*0.05))
177 zeros_end = np.zeros(int(fs*0.15))
178 x = np.concatenate((x1, zero_padding, x2, zero_padding, x3, zeros_end))
179 # Filter and plot the signal
180 fig = plt.figure(figsize=(20, 20))
181 plt.title('Filtered Signal with Filters', fontsize=fontsize+10, fontweight='bold')
182 plt.subplot(len(filters)+1, 1, 1)
183 plt.plot(x)
184 plt.ylabel('Input Signal x', fontsize=fontsize, fontweight='bold')
185
186 ax.set_xlim([0, fs])
187 for i, filter in enumerate(filters):
188     filtered_x = filter.process(x)
189     filtered_sig = (filtered_x)
190     plt.subplot(len(filters)+1, 1, i+2)
191     plt.plot(np.real(filtered_sig))
192     ax = plt.gca()
193     ax.set_ylim([-1.100, 1.100])
194     ax.set_xlim([0, fs])
195     plt.tight_layout()
196     plt.ylabel('Amplitude', fontsize=fontsize, fontweight='bold') if i == (len(
        filters)//2) else None
197
198
199 plt.xlabel('Sample t[s]', fontsize=fontsize, fontweight='bold')
200 plt.show(block=False)
201 plt.savefig("time_data.png", dpi=300, transparent=False)
202 plt.close()
203
```

```
204
205 fig = plt.figure(figsize=(22, 16))
206
207 mids = [(b+a)/2 for a, b in octave_ranges]
208 x_fftd = np.fft.fft(x)
209
210 # Plot each filter's response
211 for i, filter in enumerate(filters):
212     # Calculate position
213     position = i * 2 + 1  # Position for frequency response plot
214     x = mids[i]
215
216     # Determine the width of the rectangle from the octave range
217     fmin, fmax = octave_ranges[i]
218     rect_width = fmax - fmin
219
220     filter.plot_filter(fig, num_filters+1, 1, i+1)
221     plt.axvline(x=x, ymin=0, ymax=1, color='r', linestyle='--')
222
223     # Adding the shaded rectangle
224     plt.axvspan(x - rect_width/2, x + rect_width/2, ymin=0, ymax=1, alpha=0.3,
     color='blue')
225
226     plt.plot(x_fftd)
227     plt.legend(['octave window', 'center', 'filter pass band', 'signal_fft'], loc='
     upper right', bbox_to_anchor=(1, 1)) if i == 1 else None
228     plt.xticks([x])
229     ax = plt.gca()
230     ax.set_xlim([0, filter.fs/2])
231     plt.tight_layout()
232
233
234 plt.show(block=False)
235 plt.savefig("fft_freqdata.png", dpi=300, transparent=False)
236 input()
```

0.3 Python FIR Filter Class

```
1 import numpy as np
2 from numpy import zeros, append
3 from numpy.fft import fftshift, fft
4 import matplotlib.pyplot as plt
5
6
7 class FIRFilter:
8     def __init__(self, N=10000, fmin=3, fmax=7, padding_factor=9, fs=8000):
9         self.N = N
10        self.padding_factor = padding_factor
11        self.fs = fs  # Sampling rate
12        self.H = zeros(N)
13        self.w = zeros(N)
14        self.pos = np.arange(N)
15        self.fmin = fmin*self.N/self.fs
16        self.fmax = fmax*self.N/self.fs
17
18        self.h = None
19        self.h_pad = None
20        self.H_pad = None
21        self.w_pad = None
22        self.h_ham = None
23        self.H_ham_pad = None
```

```python
24
25          self.create_filter()
26          self.apply_padding()
27          self.apply_hamming_window()
28
29
30      def create_filter(self):
31          k = np.arange(-int(self.N/2), int(self.N/2))
32          self.w = k * self.fs / self.N  # Adjusted to use actual frequency values
33          self.H = np.where((np.abs(k) >= self.fmin) & (np.abs(k) <= self.fmax), 1,
        0)
34          self.h = fftshift(fft(fftshift(self.H)))
35
36      def apply_padding(self):
37          NP = self.N + self.padding_factor * self.N
38          self.h_pad = append(self.h, zeros(self.padding_factor * self.N))
39          self.H_pad = fftshift(fft(self.h_pad)) / self.N
40          k = np.arange(-NP/2, NP/2)
41          self.w_pad = k * self.fs / NP
42
43      def apply_hamming_window(self):
44          self.h_ham = self.h * 0.5 * (1 + np.cos(2 * np.pi * (self.pos - self.N / 2)
         / self.N))
45          self.h_ham_pad = append(self.h_ham, zeros(self.padding_factor * self.N))
46          self.H_ham_pad = fftshift(fft(self.h_ham_pad)) / self.N
47
48      def process(self, input_data):
49          return np.convolve(input_data, self.h_ham)/self.N
50
51      def plot_filter(self, fig, row, col, pos):
52          # Frequency Response Plot
53          ax1 = fig.add_subplot(row, col, pos)
54          # ax1.scatter(self.w, self.H.real, c='b', s=150)
55          # ax1.plot(self.w_pad, abs(self.H_pad), 'r')
56          ax1.plot(self.w_pad, abs(self.H_ham_pad), 'black')
57          #ax1.set_xlim(0, .5)
58          ax1.set_title(f'Frequency Response of Octave Filter {pos}', fontsize=5,
        fontweight='bold')
59          if pos == row-1:
60              ax1.legend(['Hamming'], prop={'size': 5})
61              ax1.set_xlabel('Frequency (Ratio of Fs)', fontsize=5, fontweight='bold'
        )
62              ax1.set_ylabel('Magnitude', fontsize=5, fontweight='bold')
63          ax1.grid(True)
64
65      def plot_filter1(self):
66          # MatPlotLib plotting
67          fig = plt.figure(figsize=(22, 16))
68
69          # Frequency Response Plot
70          ax1 = fig.add_subplot(211)
71          ax1.scatter(self.w, self.H.real, c='b', s=150)
72          ax1.plot(self.w_pad, abs(self.H_pad), 'r')
73          ax1.plot(self.w_pad, abs(self.H_ham_pad), 'black')
74          ax1.set_xlabel('Frequency (Hz)', fontsize=15, fontweight='bold')
75          ax1.set_ylabel('Magnitude', fontsize=15, fontweight='bold')
76          ax1.set_title('Frequency Response of FIR Filter', fontsize=15, fontweight='
        bold')
77          ax1.legend(['Ideal', 'Actual', 'Hamming'], prop={'size': 15})
78          ax1.tick_params(axis='both', labelsize=15)
79          ax1.grid(True)
80
```

```python
        # 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)
```