

Signal Analysis Project

ECEN 3300 Linear Systems Spring 2026

Part 1: Instructions

Project: Audio Signal Analysis using Digital Filtering

Objective: In this project, you will apply your knowledge of linear systems and digital filtering to analyze the frequency content of real-world audio signals:

- Noise
- Music
- Female voice
- Male voice
- Digital communication

You will design digital lowpass, bandpass, and highpass filters to isolate specific frequency bands in audio `.wav` files, measure the average signal power in each band, listen to the signal in each band, and combine the bands again to obtain a recombined signal.

Background: The average power P_x of a discrete-time signal $x[n]$ of length N is given by:

$$P_x = \frac{1}{N} \sum_{n=0}^{N-1} |x[n]|^2$$

Tasks:

1. **Load Audio:** Write code to load a `.wav` file. Convert the audio data to floating-point numbers in the range ± 1.0 . Determine the sampling frequency (f_s).
2. **Filter Design and Documentation:** Use `scipy.signal` to design digital Butterworth lowpass, bandpass, and highpass filters for the following frequency bands:
 - **Band 1 (Low):** 0 Hz – 200 Hz (LPF) or 20 Hz – 200 Hz (BPF) (Bass / fundamental frequencies of some voices)
 - **Band 2 (Mid):** 200 Hz – 2,000 Hz (Core vocal range and most instruments)
 - **Band 3 (High):** 2,000 Hz – 20,000 Hz (BPF) or 2,000 Hz – $f_s/2$ Hz (HPF) (Sibilance, cymbals, higher harmonics)

Document your filter selections with frequency response and pole-zero plots.

3. **Filtering:** Apply these filters to the different audio signals.
 4. **Power Calculation and Listening:** Compute the average power of the original signal and the filtered signals in each band. Listen to each of the signals, their decomposition into the different frequency bands, and the recombination from the different bands.
 5. **Visualization:** Plot the time-domain waveform of the original signal, the isolated high, mid, and low-frequency components, and the reconstructed signal (from the three bands) for a short segment (e.g., 0.5 seconds).
 6. **Analysis:** Answer the discussion questions at the end of the notebook.
-

Part 2: Jupyter Notebook Skeleton

Fill in the `# TODO` sections to complete the project.

```
In [209... # Module imports
import numpy as np
import matplotlib.pyplot as plt
import numpy.polynomial.polynomial as npp
from scipy.io import wavfile
from scipy import signal
import IPython.display as ipd

plt.rcParams.update({'figure.dpi': 110, 'axes.grid': True, 'grid.alpha': 0.6
```

Task 1. Load Audio

(10 points)

- Load your audio wav files. Note: The provided wav files are single channel signed 16-bit integer PCM files.
- For processing you have to convert the signals to floating point numbers in the ± 1.0 range.

```
In [210... def load_and_convert(filename):
    """
    Loads a wav file and converts to float in the range [-1.0, 1.0).
    Note: Signed 16-bit integers cover the numeric range [-2^{15}, 2^{15}-1]
    Returns: fs (sampling frequency), data (float signal)
    """
    fs, data = wavfile.read(filename)

    # Handle stereo by taking just the first channel (left)
    if data.ndim > 1:
        data = data[:, 0]
```

```

# TODO: Convert 'data' to float in the range [-1.0, 1.0)
data = data.astype(np.float64) / (2**15)

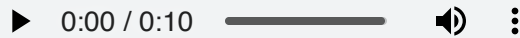
return fs, data

# Test your function here
filename = 'wavFiles/Voice_Female.wav'
fs, data = load_and_convert(filename)

# Play back the loaded file
ipd.Audio(data, rate=fs)

```

Out [210...



Task 2. Digital Filter Design and Documentation

(20 points)

Design and document digital Butterworth lowpass, bandpass, and highpass filters of order N .

You may want to check out the [scipy.signal.butter](#) command for some details.

Digital filters with real-valued impulse responses are generally designed for frequencies in the range $0 < \omega < \pi$ (where π represents the highest frequency). In

`scipy.signal.butter` (without explicitly specifying `fs`), this range is normalized to $[0, 1]$. In this project we are interested in processing audio signals, with frequencies in the range from 0 to 20,000 Hz, that have been sampled with a sampling rate of `fs=44,100` Hz. In this context, the normalized $[0, 1]$ range of

`scipy.signal.butter` corresponds to the frequency range $[0, \text{nyquist}]$, where the Nyquist frequency `nyquist=fs/2` (22,050 Hz).

When computing the frequency response of the digital filters, use [scipy.signal.freqz](#) with frequencies specified in an array `w` (omega) in radians ($[0, \pi]$ or $[-\pi, \pi]$). Then display the frequency axes in Hz using $f = f_s * \omega / (2\pi)$.

For the pole-zero plots use `numpy.roots` or the newer [numpy.polynomial.polynomial.polyroots](#) on the `b`, `a` coefficients of the filter designs from `scipy.signal.butter`. Note that the filters can become unstable if the filter order `N` is chosen too high.

In [211...

```

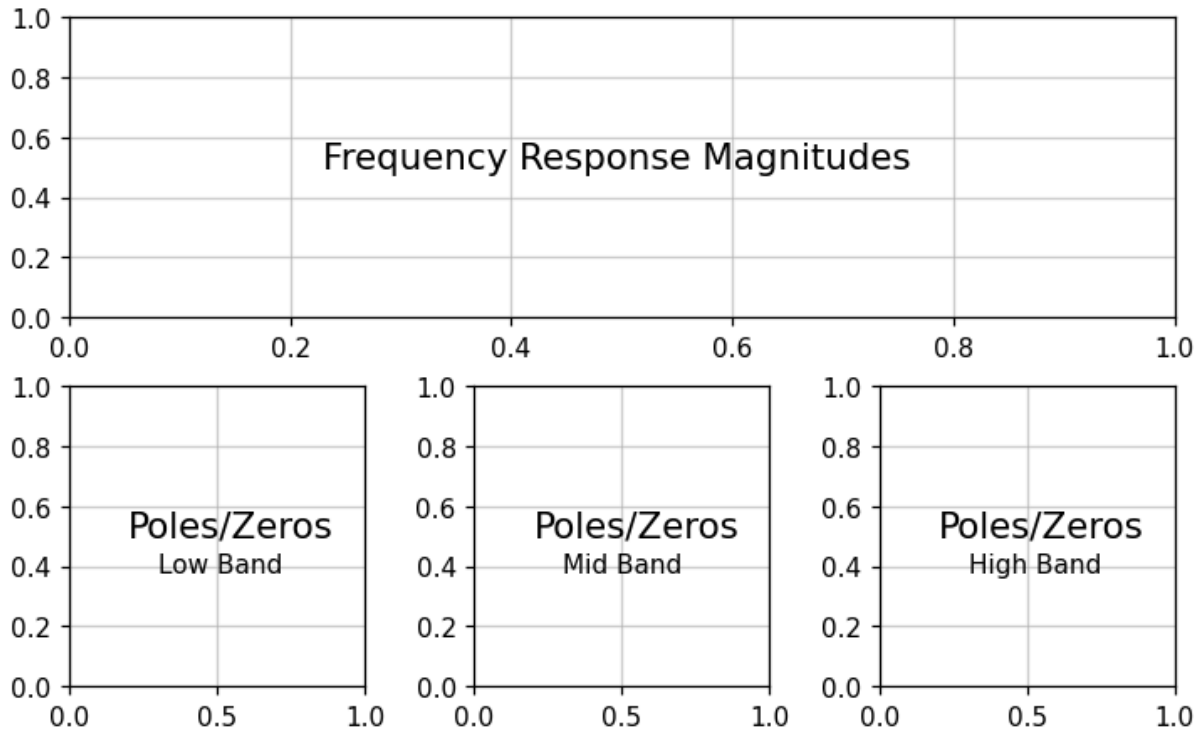
# The goal of this task is to design filters and document the designs in ter
fig = plt.figure(figsize = (6.5, 4), layout = 'constrained')

```

```

spec = fig.add_gridspec(2, 3)
ax0 = fig.add_subplot(spec[0, :])
ax0.annotate('Frequency Response Magnitudes', (0.23,0.5), fontsize=14)
ax10 = fig.add_subplot(spec[1, 0])
ax10.annotate('Poles/Zeros', (0.2,0.5), fontsize=14)
ax10.annotate('Low Band', (0.3,0.38))
ax11 = fig.add_subplot(spec[1, 1])
ax11.annotate('Poles/Zeros', (0.2,0.5), fontsize=14)
ax11.annotate('Mid Band', (0.3,0.38))
ax12 = fig.add_subplot(spec[1, 2])
ax12.annotate('Poles/Zeros', (0.2,0.5), fontsize=14)
ax12.annotate('High Band', (0.3,0.38))
plt.show()

```



```

In [212]: def design_butter_filter(f_low, f_high, fs, N):
          """
          Designs a digital Butterworth filter:
          LPF if f_low == 0, order N
          HPF if f_high >= nyquist = fs/2.0, order N
          BPF with 0 < f_low < f_high < nyquist otherwise, effective order 2*N
          Returns: b, a (numerator, denominator filter coefficients)
          """
          nyquist = 0.5 * fs

          # TODO: Calculate normalized frequencies

          # TODO: Use signal.butter to design the filter
          # Hint: Break this up into btype='low', btype='high', btype='band', see

          if f_low == 0:
              # Lowpass filter
              Wn = f_high / nyquist
              b, a = signal.butter(N, Wn, btype='low')

```

```

elif f_high >= nyquist:
    # Highpass filter
    Wn = f_low / nyquist
    b, a = signal.butter(N, Wn, btype='high')
else:
    # Bandpass filter
    Wn = [f_low / nyquist, f_high / nyquist]
    b, a = signal.butter(N, Wn, btype='band')

return b, a

```

In [213...

```

def plot_freqresp_lin(bands, fs, f=None, ax=None, title='Frequency Response,
'''
    Plot magnitude of frequency response in dB for filters specified
    in bands list versus linear frequency axis
    Bands list entries are of the form:
    ('band', f_low, f_high, N, 'color')
    where
        band: 'low', 'mid', or 'high'
        N: filter order
    '''
    nyquist = 0.5*fs

    if ax is None:
        ax = plt.gca()
    if f is None:
        f = np.linspace(0, nyquist, 2000)

    # TODO: Loop through the bands, determine b, a and use freqz to compute
    # Note: Plot versus  $f = fs * w / (2\pi)$ . Use color and label to distinguish the
    for name, low, high, order, color in bands:
        b, a = design_butter_filter(low, high, fs, order)
        w = 2 * np.pi * f / fs
        _, Hejw = signal.freqz(b, a, worN=w)
        H_dB = 20 * np.log10(np.maximum(np.abs(Hejw), 1e-12))
        ax.plot(f, H_dB, color=color, label=name)

    ax.set_xlabel('$f$ [Hz]')
    ax.set_ylabel('$|H(e^{j2\pi f/f_s})|$ [dB]')
    ax.set_title(title)
    ax.legend()
    return ax

```

In [214...

```

def plot_freqresp_log(bands, fs, f=None, ax=None, title='Frequency Response,
'''
    Plot magnitude of frequency response in dB for filters specified
    in bands list versus logarithmic frequency axis
    Bands list entries are of the form:
    ('band', f_low, f_high, N, 'color')
    where
        band: 'low', 'mid', or 'high'
        N: filter order
    '''
    nyquist = 0.5*fs

```

```

if ax is None:
    ax = plt.gca()
if f is None:
    f = np.logspace(np.log10(10), np.log10(nyquist), 2000)

# TODO: Loop through the bands, determine b, a and use freqz to compute
# Note: plot versus  $f = f_s \cdot w / (2\pi)$  using semilogx. Use color and label to
for name, low, high, order, color in bands:
    b, a = design_butter_filter(low, high, fs, order)
    w = 2 * np.pi * f / fs
    _, Hejw = signal.freqz(b, a, worN=w)
    H_dB = 20 * np.log10(np.maximum(np.abs(Hejw), 1e-12))
    ax.semilogx(f, H_dB, color=color, label=name)

ax.set_xlabel('$f$ [Hz]')
ax.set_ylabel('$|H(e^{j2\pi f/f_s})|$ [dB]')
ax.set_title(title)
ax.legend()
return ax

```

```

In [215... def pole_zero_plot(band, fs, ax=None, title='Pole Zero Plot'):
    """
    Make pole-zero plot for filter specified in (single) band
    Format of band is
    ('band', f_low, f_high, N, 'color')
    where
        band: 'low', 'mid', or 'high'
        N: filter order

    """
    if ax is None:
        ax = plt.gca()

    # TODO: Determine b, a coefficients of filter specified in band

    name, low, high, order, color = band
    b, a = design_butter_filter(low, high, fs, order)

    # TODO: Draw a unit circle:  $|z| = 1$ , dashed, gray
    # Hint: parametrize with  $\theta = \text{np.linspace}(0, 2*\text{np.pi}, 360)$ 

    theta = np.linspace(0, 2*np.pi, 360)
    ax.plot(np.cos(theta), np.sin(theta), '--', color='gray', linewidth=0.8)
    # TODO: Draw real and imaginary axes, black

    ax.axhline(0, color='black', linewidth=0.5)
    ax.axvline(0, color='black', linewidth=0.5)
    # TODO: Compute poles and zeros using roots or polyroots

    zeros = np.roots(b)
    poles = np.roots(a)
    # TODO: Plot poles ('x', 'red') and zeros ('o', 'blue'), add labels

    ax.plot(np.real(zeros), np.imag(zeros), 'o', color='blue', markersize=5,

```

```

ax.plot(np.real(poles), np.imag(poles), 'x', color='red', markersize=7,

ax.set_xlabel("Re")
ax.set_ylabel("Im")
ax.set_title(title)
ax.set_aspect('equal')
ax.legend(fontsize=9)
return ax

```

```

In [216... # Define the frequency bands and filter orders
# Could use LPF, BPF, and HPF
# or all BPFs, 20...200, 200...2000, 2000...20000 Hz
bands = [
    ("Low", 0, 200, 2, 'blue'),
    ("Mid", 200, 2000, 2, 'green'),
    ("High", 2000, 22050, 2, 'crimson')
]

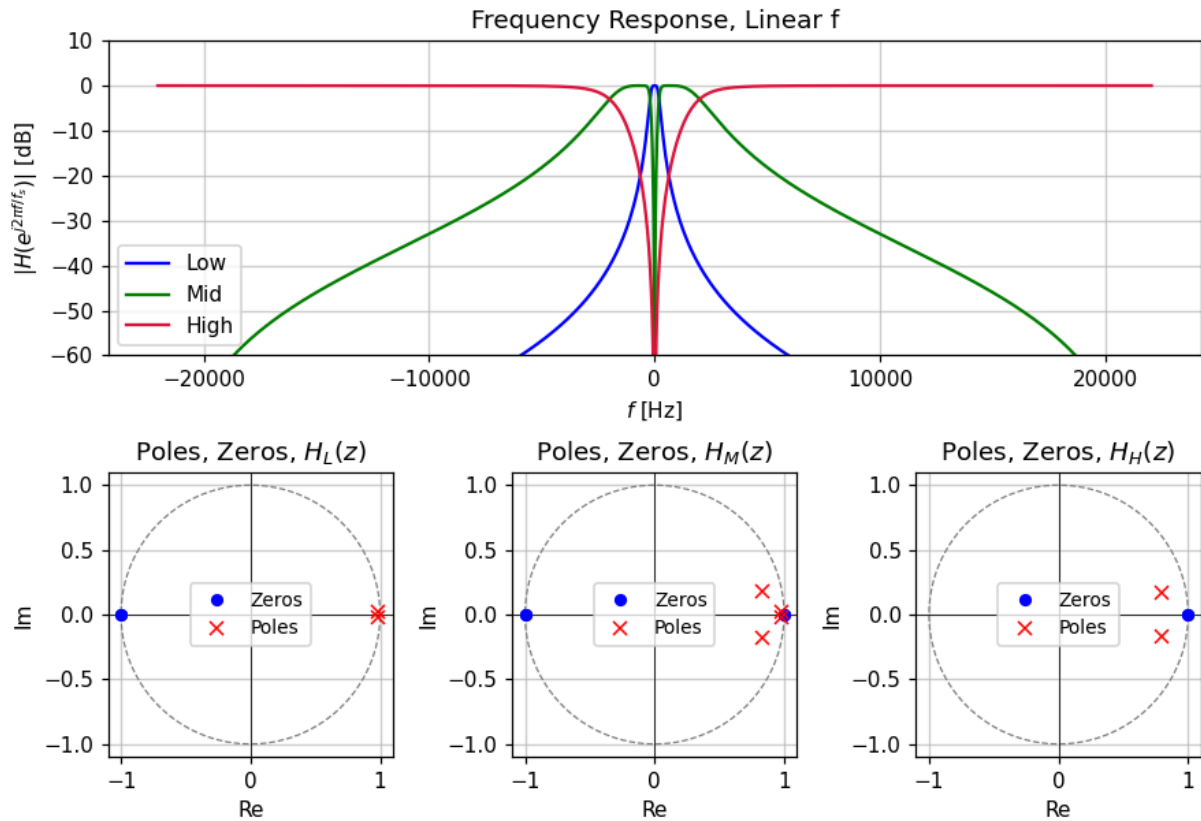
```

```

In [217... # Plot frequency response magnitude and pole locations for all bands
# Linear frequency scale in range [-fs/2, fs/2]

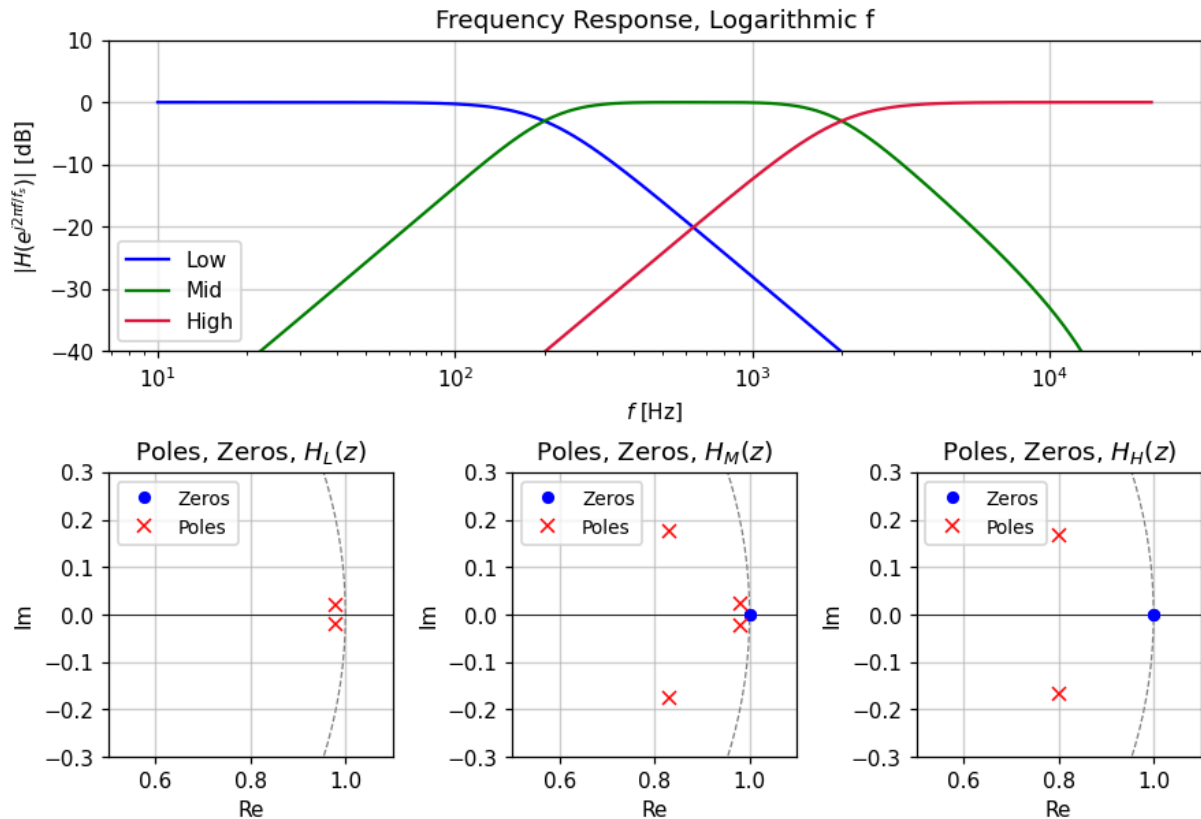
f = np.arange(-fs/2, fs/2)
fig = plt.figure(figsize = (8, 5.5), layout = 'constrained')
spec = fig.add_gridspec(2, 3)
ax0 = fig.add_subplot(spec[0, :])
ax0 = plot_freqresp_lin(bands, fs, f)
ax0.set_ylim([-60, 10])
ax10 = fig.add_subplot(spec[1, 0])
ax10 = pole_zero_plot(bands[0], fs, title='Poles, Zeros,  $H_L(z)$ ')
ax11 = fig.add_subplot(spec[1, 1])
ax11 = pole_zero_plot(bands[1], fs, title='Poles, Zeros,  $H_M(z)$ ')
ax12 = fig.add_subplot(spec[1, 2])
ax12 = pole_zero_plot(bands[2], fs, title='Poles, Zeros,  $H_H(z)$ ')
plt.show()

```



In [218... `# Plot frequency response magnitude and pole locations for all bands`
`# Logarithmic frequency scale in range [log10(10), log10(fs/2)]`
`# Zoom in on pole locations near z=1`

```
fig = plt.figure(figsize = (8, 5.5), layout = 'constrained')
spec = fig.add_gridspec(2, 3)
ax0 = fig.add_subplot(spec[0, :])
ax0 = plot_freqresp_log(bands, fs)
ax0.set_ylim([-40, 10])
ax10 = fig.add_subplot(spec[1, 0])
ax10 = pole_zero_plot(bands[0], fs, title='Poles, Zeros, $H_L(z)$')
ax10.set_xlim([0.5, 1.1]); ax10.set_ylim([-0.3, 0.3])
ax11 = fig.add_subplot(spec[1, 1])
ax11 = pole_zero_plot(bands[1], fs, title='Poles, Zeros, $H_M(z)$')
ax11.set_xlim([0.5, 1.1]); ax11.set_ylim([-0.3, 0.3])
ax12 = fig.add_subplot(spec[1, 2])
ax12 = pole_zero_plot(bands[2], fs, title='Poles, Zeros, $H_H(z)$')
ax12.set_xlim([0.5, 1.1]); ax12.set_ylim([-0.3, 0.3])
plt.show()
```



The choice of **filter order** is a compromise between stability of the filter and keeping the percentage of out-of-band power low. The in-band and out-of-band powers P_{in} and P_{out} can be estimated from $|H(e^{j\omega})|^2$ for the values of $\omega = 2\pi f/f_s$ that fall inside or outside the passband of the filter, respectively. Complete the `outofband_power` function below to determine $100 \cdot P_{out}/P_{in}$ in percent.

```
In [219... # Estimate percentage of out-of-band power
def outofband_power(band, fs):
    """
    Determine percentage of out-of-band power from  $|H(e^{j\omega})|^2$ 
    Returns:  $100 \cdot P_{out\_band}/P_{in\_band}$ 
    """
    nyquist = fs/2 # Nyquist frequency

    # TODO: try  $f = np.linspace()$  and  $f = np.logspace()$ . Which is better? What
    f = np.linspace(0, nyquist, 5000)
    # f = np.logspace(np.log10(20), np.log10(nyquist), 5000)

    # TODO: Compute frequency response  $H_{ejw}$ 

    name, low, high, order, color = band
    b, a = design_butter_filter(low, high, fs, order)

    w = 2 * np.pi * f / fs
    _, Hejw = signal.freqz(b, a, worN=w)
    H2 = np.abs(Hejw)**2

    # TODO: Determine in-band and out-of-band indexes
```

```

in_band = (f >= low) & (f <= high)
out_band = ~in_band

P_in = np.sum(H2[in_band])
P_out = np.sum(H2[out_band])

return 100*P_out/P_in

```

In []:

```

In [220...] # List percentage of out-of-band powers
for i in range(len(bands)):
    pP_out = outofband_power(bands[i], fs) # percentage of out-of-band pow
    print(f'{bands[i][0]}: {pP_out:1.3f}%, ({bands[i][1]}, {bands[i][2]}), N=

Low: 27.489%, (0,200), N=2
Mid: 26.941%, (200,2000), N=2
High: 1.350%, (2000,22050), N=2

```

Task 3. Apply Filters and Calculate Power

(10 points)

Apply the `bands` filters to the audio signals and listen to the outputs and the recombined signals. Calculate the total average power and the percentages of how the total power splits up into the different bands.

You will look at two implementations of filtering in `scipy` with given `b`, `a` coefficients:

- `scipy.signal.lfilter` which essentially performs the DT convolution $y[n] = x[n] * h[n]$, where the `b`, `a` coefficients define $h[n]$, and
- `scipy.signal.filtfilt` which uses `lfilter` twice (once forward, once backward) to eliminate any phase distortion or time shift that would normally be introduced by a causal single-pass through `lfilter`.

```

In [221...] def apply_lfilter(data, b, a):
    """
    Applies scipy.lfilter to the data.
    """
    # TODO: Apply the filter: y = signal.lfilter()
    y = signal.lfilter(b, a, data)
    return y

```

```

In [222...] def apply_filtfilt(data, b, a):
    """
    Applies scipy.filtfilt to the data for zero-phase filtering.
    """
    # TODO: Apply the filter: y = signal.filtfilt()

```

```
y = signal.filtfilt(b, a, data)
return y
```

```
In [223... def calculate_power(sig):
            """
            Calculates the average signal power.
            """
            # TODO: Implement the power formula
            #  $P = 1/N * \sum(|x[n]|^2)$ 
            P = np.mean(np.abs(sig)**2)
            return P
```

Task 4. Main Analysis Loop

(10 points)

(Re)Define your bands, loop through them, filter the signals, and print the power.

The signals to analyze are:

- `Noise.wav` : A white noise signal with constant intensity at all frequencies below `fs / 2`.
- `Music.wav` :
- `Voice_Female.wav` :
- `Voice_Male.wav` :
- `CommSig.wav` :

```
In [224... # Print the current definition of the `bands`
for i in range(len(bands)):
    print(bands[i])
```

```
('Low', 0, 200, 2, 'blue')
('Mid', 200, 2000, 2, 'green')
('High', 2000, 22050, 2, 'crimson')
```

```
In [225... # TODO: For all your audio signals, loop through the bands.
# 1. (Re)Design the filters if necessary
# 2. Apply the filters
# 3. Listen to original signals, the decomposed signals, and the recombined
#    and describe the effect of the decomposition into the different bands.
#    check how close recombining the decomposed signals comes to the original
# 4. Calculate and print the power in that band
# 5. Store the filtered signals for plotting

#filename, filtype = 'wavFiles/Noise.wav', 'lfilter'
#filename, filtype = 'wavFiles/Music.wav', 'lfilter'
#filename, filtype = 'wavFiles/Voice_Female.wav', 'lfilter'
#filename, filtype = 'wavFiles/Voice_Male.wav', 'lfilter'
#filename, filtype = 'wavFiles/Voice_Male.wav', 'filtfilt'
#filename, filtype = 'wavFiles/Voice_Male.wav', 'filtfilt'
#filename, filtype = 'wavFiles/CommSig.wav', 'lfilter'
```

```

filename, filtype = 'wavFiles/CommSig.wav', 'filtfilt'

fs, data = load_and_convert(filename)

print(f'{filename}, {filtype}')
total_power = calculate_power(data)
print(f"Total Power: {total_power:.5f}")

filtered_signals = {}
recombined_signal = np.zeros(data.size)
for name, low, high, order, color in bands:
    b, a = design_butter_filter(low, high, fs, order)
    if filtype == 'filtfilt':
        filtered = apply_filtfilt(data, b, a)
    else:
        filtered = apply_lfilter(data, b, a)
    filtered_signals[name] = filtered
    recombined_signal += filtered
    power = calculate_power(filtered)
    print(f"Band {name} ({low}-{high} Hz) Power: {power:.3f} ({(power/total_power):.1%}")

recombined_power = calculate_power(recombined_signal)
print(f'Recombined Power: {recombined_power:.3f} ({recombined_power/total_power:.1%}')

filtered_signals['Orig'] = data
filtered_signals['Recomb'] = recombined_signal

```

```

wavFiles/CommSig.wav, filtfilt
Total Power: 0.80450
Band Low (0-200 Hz) Power: 0.227 (28.2%)
Band Mid (200-2000 Hz) Power: 0.420 (52.2%)
Band High (2000-22050 Hz) Power: 0.031 (3.8%)
Recombined Power: 0.843 (104.8%)

```

```

In [226... # Listen to the different parts of the decomposition
sel = ['Orig', 'Low', 'Mid', 'High', 'Recomb']
Selection = 4 # choose 0...4
ipd.Audio(filtered_signals[sel[Selection]], rate=fs)

```

Out [226...]

▶ 0:00 / 0:10 ———— 🔊 ⋮

Task 5. Visualization

(10 points)

Plot the first 0.5 (or 0.1 for CommSig.wav) seconds of the original signal, its three filtered components, and its recombination.

```

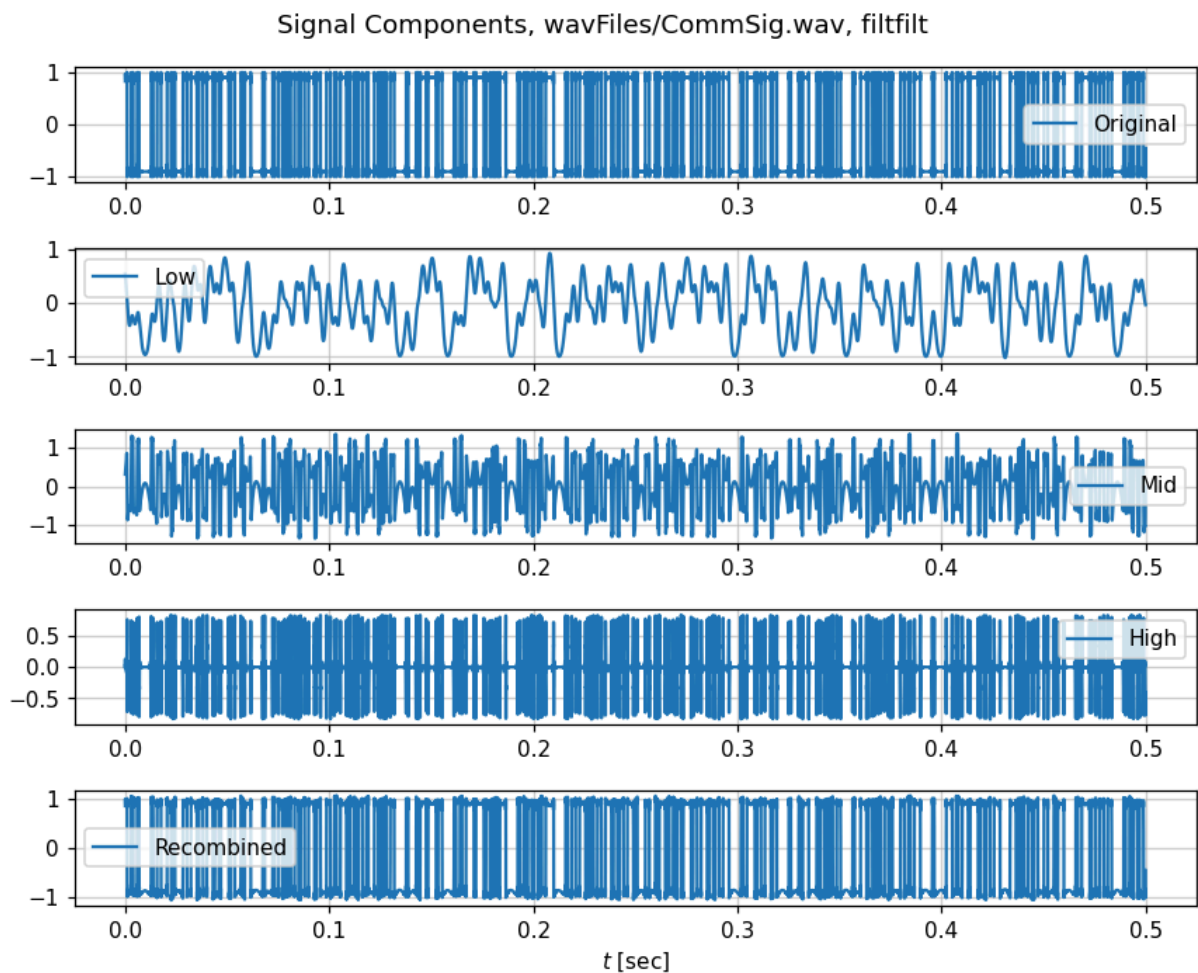
In [227... # TODO: Plot the original and filtered signals using matplotlib.pyplot
# Make sure to label your axes (Time in seconds, Amplitude).
t = np.arange(int(0.5 * fs)) / fs

```

```

fig, ax = plt.subplots(5, 1, figsize=(8, 6.5))
fig.suptitle(f'Signal Components, {filename}, {filttype}')
ax[0].plot(t, filtered_signals["Orig"][:len(t)], label='Original')
ax[0].legend()
ax[1].plot(t, filtered_signals["Low"][:len(t)], label='Low')
ax[1].legend()
ax[2].plot(t, filtered_signals["Mid"][:len(t)], label='Mid')
ax[2].legend()
ax[3].plot(t, filtered_signals["High"][:len(t)], label='High')
ax[3].legend()
ax[4].plot(t, filtered_signals["Recomb"][:len(t)], label='Recombined')
ax[4].set_xlabel('$t$ [sec]')
ax[4].legend()
plt.legend()
plt.tight_layout()
plt.show()

```



Task 6. Discussion Questions

(40 points)

1. How did the power distribution across the three bands differ for the different signals? Show the results in the form of a table and state the parameters of the

filters you used as well as the filter operation (`lfilter` or `filtfilt`).

The following table shows the power distribution across the three bands for each signal. All filters are Butterworth, order N=2. Low band: LPF 0-200 Hz. Mid band: BPF 200-2000 Hz. High band: HPF 2000-22050 Hz.

Signal	Filter	Low (0-200 Hz)	Mid (200-2000 Hz)	High (2000-22050 Hz)	Recombined
Noise	<code>lfilter</code> , N=2	1.0%	9.0%	90.0%	83.1%
Music	<code>lfilter</code> , N=2	31.8%	61.7%	8.4%	46.5%
Voice_Female	<code>lfilter</code> , N=2	6.1%	80.2%	16.6%	47.0%
Voice_Male	<code>lfilter</code> , N=2	23.4%	78.4%	2.0%	30.7%
CommSig	<code>lfilter</code> , N=2	37.0%	60.0%	5.3%	45.1%
Voice_Male	<code>filtfilt</code> , N=2	10.2%	67.7%	1.3%	107.8%
CommSig	<code>filtfilt</code> , N=2	28.2%	52.2%	3.8%	104.8%

The noise signal has 90% of its power in the high band, consistent with white noise having equal power per Hz — since the high band spans the most Hz (2000-22050), it contains the most power. The voice signals concentrate power in the mid band (78-80%), which makes sense since the core vocal range falls in 200-2000 Hz. The male voice has more low-band power (23.4%) than the female voice (6.1%), reflecting the lower fundamental frequency of male speech. Music has a broad distribution with most power in the mid band (61.7%) and significant low-band content (31.8%) from bass and drums. The communication signal has power concentrated in the low and mid bands (37% and 60%), with little high-frequency content.

- Describe for each signal how the decomposition into the three bands sounds when played back. In which of the bands is the content recognizable, which bands contribute most to the identification of the content (e.g., is speech understandable if you have not heard the original before)? Show the results in the form of a table and state the parameters of the filters you used as well as the filter operation (`lfilter` or `filtfilt`).

Filter parameters: Butterworth, order N=2, `lfilter`. Low band: LPF 0-200 Hz. Mid band: BPF 200-2000 Hz. High band: HPF 2000-22050 Hz.

Signal	Original	Low Band	Mid Band	High Band	Recombined	Most Recognizable
Noise	Hissing/static	Wind	Static	Rain/static	Static, higher pitch	None — all bands sound like static variations
Music	Normal music	Deeper, bass-heavy	Similar to original, slightly lower quality	Tinny, staticky	Normal, slightly lower quality	Mid
Voice_Female	Normal female voice	Muted hum	Normal, slightly lower quality	Quiet, tin-can sound	Normal, slightly lower quality	Mid
Voice_Male	Normal male voice	Muted	Normal, slightly lower quality	Quiet, tinny	Normal, slightly lower quality	Mid
CommSig	Harsh wave/tone	Wind	Quieter version of original	Higher pitch, quieter	Quieter but similar to original	Mid

For all voice and music signals, the mid band (200–2000 Hz) is the most important for content recognition. Speech is fully understandable in the mid band alone, even without hearing the original first. The low band contributes warmth and fullness but is not sufficient for identification on its own. The high band adds clarity and crispness (sibilance in speech, cymbals in music) but sounds tinny and metallic in isolation. The recombined signals sound close to the originals but with slightly degraded quality due to phase distortion from the causal filter and power leakage from the order-2 filter rolloff.

- When measuring the percentage of out-of-band power as part of the low, mid, high filter designs, should this be done using `f=np.linspace()` or `f=np.logspace()`. What speaks for or against each of the two options?

`np.linspace` is the better choice for measuring out-of-band power percentage. With `linspace`, the frequency points are uniformly spaced, so each point represents an equal-width slice of the spectrum. This means summing $|H(e^{j\omega})|^2$ over those points gives a fair estimate of the total power in each region, since the contribution of each frequency bin is equally weighted.

With `np.logspace`, points are concentrated at lower frequencies and sparse at higher frequencies. This over-represents the low-frequency region and under-represents the high-frequency region. For the high band filter (2000–22050 Hz), `logspace` would place

very few sample points in the wide passband and many points in the narrow low-frequency stopband, giving a misleading ratio of out-of-band to in-band power.

The argument in favor of logspace is that it provides better frequency resolution where filter transitions are sharpest (near DC for the low band filter), which could capture the transition band shape more accurately. However, for the purpose of computing a power ratio, uniform spacing (linspace) gives an unbiased estimate.

4. What are the differences between using a lowpass filter from 0 to 200 Hz versus using a bandpass filter from 20 to 200 Hz for the **low band** and between using a highpass filter from 2000 to $f_s/2$ Hz versus using a bandpass filter from 2000 to 20000 Hz for the **high band**? List advantages and disadvantages in terms of separation of the bands (and power measurements), effect on quality of recombination of the signals from the low, mid, and high bands for different signals.

Low band — LPF (0-200 Hz) vs BPF (20-200 Hz):

Aspect	LPF (0-200 Hz)	BPF (20-200 Hz)
DC content	Passes DC and sub-20 Hz content	Rejects DC offset and sub-audible frequencies
Filter complexity	Simpler — order N	Effectively order 2N (steeper rolloff)
Band separation	Includes sub-audible content that inflates power measurements	Cleaner separation, more meaningful power values
Recombination	Full spectrum coverage below 200 Hz, no gaps	Frequencies 0-20 Hz are lost, creating a gap in reconstruction

High band — HPF (2000- $f_s/2$ Hz) vs BPF (2000-20000 Hz):

Aspect	HPF (2000-22050 Hz)	BPF (2000-20000 Hz)
Upper limit	Passes up to Nyquist (22050 Hz)	Cuts off at 20000 Hz
Filter complexity	Simpler — order N	Effectively order 2N
Band separation	Includes content above human hearing (20-22 kHz)	Only measures audible high-frequency power
Recombination	No gap at upper end, full spectrum recovered	Frequencies 20000-22050 Hz are lost

Using LPF + BPF + HPF ensures the three bands cover the entire spectrum with no gaps, producing the best possible recombination. Using all BPFs (20-200, 200-2000, 2000-20000) leaves gaps at 0-20 Hz and 20000-22050 Hz, which means the recombined signal cannot perfectly match the original. However, the all-BPF approach gives cleaner band separation and more meaningful power measurements for audible content, since it excludes DC offset and ultrasonic frequencies that are not perceptually relevant.

5. How do the differences between using `lfilter` and the zero-phase `filtfilt` filters manifest themselves? Do they matter for the power measurements? Do they matter in terms of the recombination of the signals from the tree bands? Do they matter in the same way for all signals. Hint: Try the communication signal and one of the speech signals using both filter types and using either a lowpass filter (0 to 200 Hz) or a bandpass filter (20 to 200 Hz) for the **low band**. What conclusions can you draw?

Signal	<code>lfilter</code> Recombined Power	<code>filtfilt</code> Recombined Power
Voice_Male	30.7%	107.8%
CommSig	45.1%	104.8%

Power measurements: The individual band power percentages differ between `lfilter` and `filtfilt`. Since `filtfilt` applies the filter twice (forward and backward), it effectively squares the magnitude response $|H|^2$, resulting in a steeper rolloff and more attenuation in the transition and stop bands. This means band powers measured with `filtfilt` are lower than with `lfilter`.

Recombination: This is where the biggest difference appears. With `lfilter`, the recombined power is only 30–45% of the original — significant power is lost. This happens because `lfilter` is a causal filter that introduces frequency-dependent phase shifts. When the three filtered bands are summed, these phase shifts cause partial destructive interference, so the recombined signal does not reconstruct the original well.

With `filtfilt`, the recombined power is close to 100% (107.8% and 104.8%). Since `filtfilt` is zero-phase, all three bands maintain their original phase relationships. When summed, they reconstruct the original signal almost perfectly. The slight overshoot above 100% is due to overlap in the transition bands of the order-2 filters, where adjacent filters both pass some energy, resulting in a small gain.

Signal dependence: The effect is consistent across signals — `filtfilt` always produces better recombination. However, the magnitude of the `lfilter` recombination loss varies by signal. `Voice_Male` (30.7%) loses more than `CommSig` (45.1%), likely because the voice signal has more energy concentrated near the filter crossover frequencies where phase distortion is most severe.

Conclusion: For applications where signal reconstruction from sub-bands is important, `filtfilt` is clearly superior due to its zero-phase property. For simple power measurements in isolated bands, `lfilter` is acceptable, though `filtfilt` gives slightly different values due to its steeper effective rolloff.

Bonus Question

(up to 20 points)

Repeat Tasks 2 to 6 for Chebyshev Type 1 filters with a passband ripple of 3 dB and compare the results to the analysis with the Butterworth filters.

Bonus: Chebyshev Type 1 Filter Analysis

Repeating Tasks 2-6 using Chebyshev Type 1 filters with 3 dB passband ripple.

```
In [228... def design_cheby1_filter(f_low, f_high, fs, N, rp=3):

    nyquist = 0.5 * fs

    if f_low == 0:
        Wn = f_high / nyquist
        b, a = signal.cheby1(N, rp, Wn, btype='low')
    elif f_high >= nyquist:
        Wn = f_low / nyquist
        b, a = signal.cheby1(N, rp, Wn, btype='high')
    else:
        Wn = [f_low / nyquist, f_high / nyquist]
        b, a = signal.cheby1(N, rp, Wn, btype='band')

    return b, a

In [229... def plot_freqresp_lin_cheby(bands, fs, f=None, ax=None, title='Chebyshev Fre
nyquist = 0.5*fs
if ax is None:
    ax = plt.gca()
if f is None:
    f = np.linspace(0, nyquist, 2000)
for name, low, high, order, color in bands:
    b, a = design_cheby1_filter(low, high, fs, order)
    w = 2 * np.pi * f / fs
    _, Hejw = signal.freqz(b, a, worN=w)
    H_dB = 20 * np.log10(np.maximum(np.abs(Hejw), 1e-12))
    ax.plot(f, H_dB, color=color, label=name)
ax.set_xlabel('$f$ [Hz]')
ax.set_ylabel('$|H(e^{j2\pi f/f_s})|$ [dB]')
ax.set_title(title)
ax.legend()
return ax

def plot_freqresp_log_cheby(bands, fs, f=None, ax=None, title='Chebyshev Fre
nyquist = 0.5*fs
if ax is None:
    ax = plt.gca()
if f is None:
    f = np.logspace(np.log10(10), np.log10(nyquist), 2000)
for name, low, high, order, color in bands:
```

```

        b, a = design_cheby1_filter(low, high, fs, order)
        w = 2 * np.pi * f / fs
        _, Hejw = signal.freqz(b, a, worN=w)
        H_dB = 20 * np.log10(np.maximum(np.abs(Hejw), 1e-12))
        ax.semilogx(f, H_dB, color=color, label=name)
    ax.set_xlabel('$f$ [Hz]')
    ax.set_ylabel('$|H(e^{j2\pi f/f_s})|$ [dB]')
    ax.set_title(title)
    ax.legend()
    return ax

def pole_zero_plot_cheby(band, fs, ax=None, title='Pole Zero Plot'):
    if ax is None:
        ax = plt.gca()
    name, low, high, order, color = band
    b, a = design_cheby1_filter(low, high, fs, order)
    theta = np.linspace(0, 2*np.pi, 360)
    ax.plot(np.cos(theta), np.sin(theta), '--', color='gray', linewidth=0.8)
    ax.axhline(0, color='black', linewidth=0.5)
    ax.axvline(0, color='black', linewidth=0.5)
    zeros = np.roots(b)
    poles = np.roots(a)
    ax.plot(np.real(zeros), np.imag(zeros), 'o', color='blue', markersize=5,
    ax.plot(np.real(poles), np.imag(poles), 'x', color='red', markersize=7,
    ax.set_xlabel("Re")
    ax.set_ylabel("Im")
    ax.set_title(title)
    ax.set_aspect('equal')
    ax.legend(fontsize=9)
    return ax

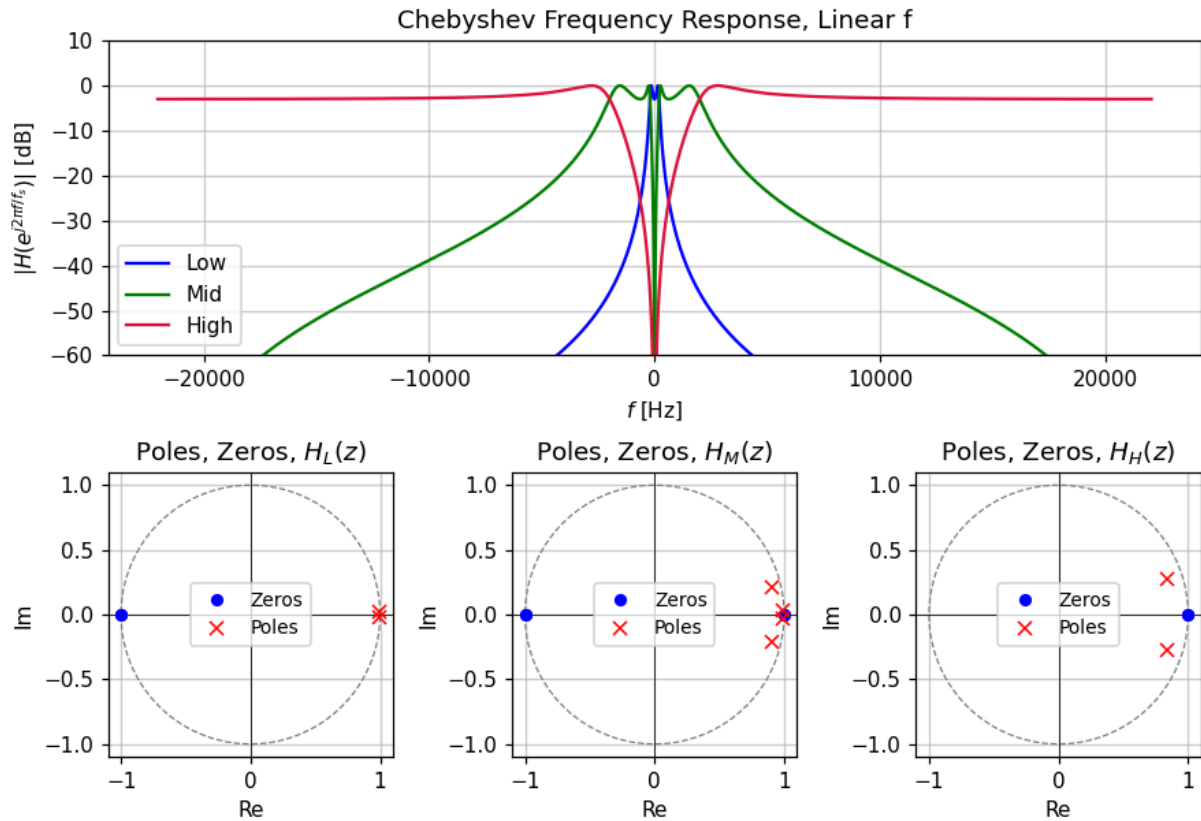
```

```

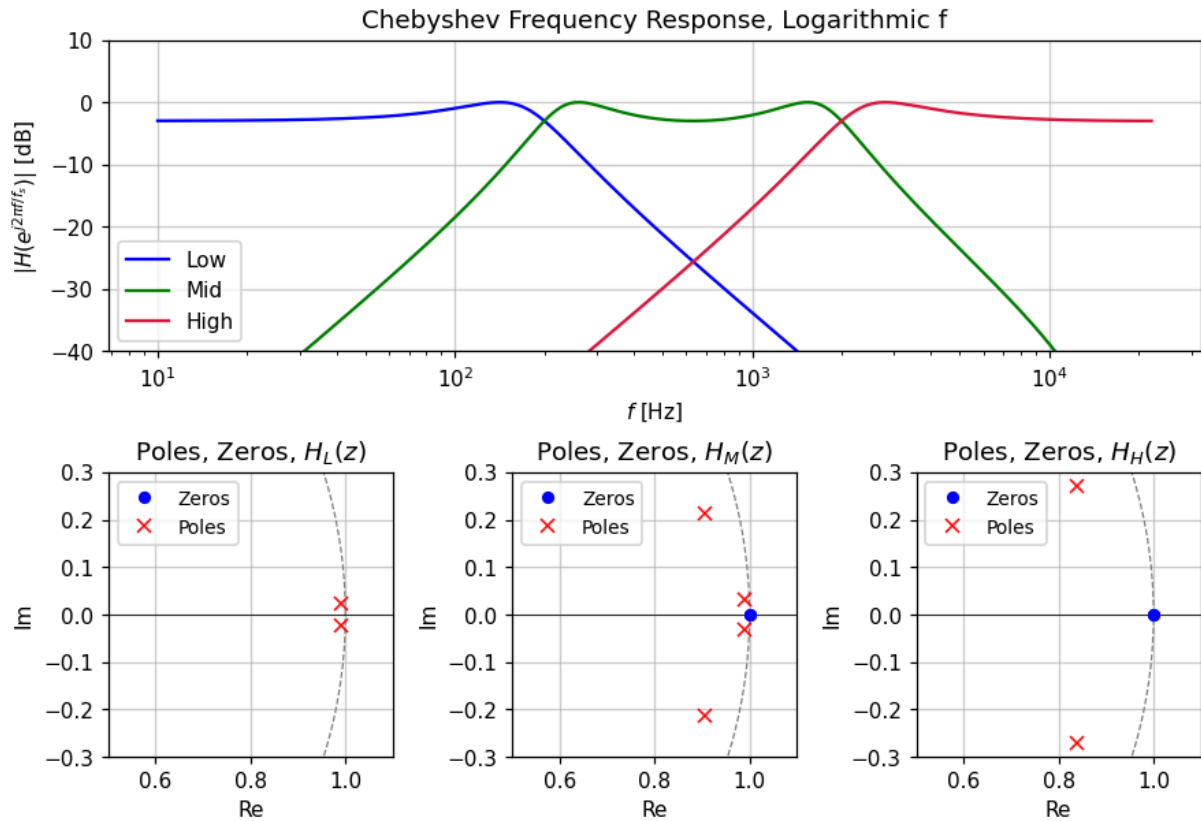
In [230... bands_cheby = [
    ("Low", 0, 200, 2, 'blue'),
    ("Mid", 200, 2000, 2, 'green'),
    ("High", 2000, 22050, 2, 'crimson')
]

f = np.arange(-fs/2, fs/2)
fig = plt.figure(figsize=(8, 5.5), layout='constrained')
spec = fig.add_gridspec(2, 3)
ax0 = fig.add_subplot(spec[0, :])
ax0 = plot_freqresp_lin_cheby(bands_cheby, fs, f)
ax0.set_ylim([-60, 10])
ax10 = fig.add_subplot(spec[1, 0])
ax10 = pole_zero_plot_cheby(bands_cheby[0], fs, title='Poles, Zeros, $H_L(z)$')
ax11 = fig.add_subplot(spec[1, 1])
ax11 = pole_zero_plot_cheby(bands_cheby[1], fs, title='Poles, Zeros, $H_M(z)$')
ax12 = fig.add_subplot(spec[1, 2])
ax12 = pole_zero_plot_cheby(bands_cheby[2], fs, title='Poles, Zeros, $H_H(z)$')
plt.show()

```



```
In [231]: fig = plt.figure(figsize=(8, 5.5), layout='constrained')
spec = fig.add_gridspec(2, 3)
ax0 = fig.add_subplot(spec[0, :])
ax0 = plot_freqresp_log_cheby(bands_cheby, fs)
ax0.set_ylim([-40, 10])
ax10 = fig.add_subplot(spec[1, 0])
ax10 = pole_zero_plot_cheby(bands_cheby[0], fs, title='Poles, Zeros,  $H_L(z)$ ')
ax10.set_xlim([0.5, 1.1]); ax10.set_ylim([-0.3, 0.3])
ax11 = fig.add_subplot(spec[1, 1])
ax11 = pole_zero_plot_cheby(bands_cheby[1], fs, title='Poles, Zeros,  $H_M(z)$ ')
ax11.set_xlim([0.5, 1.1]); ax11.set_ylim([-0.3, 0.3])
ax12 = fig.add_subplot(spec[1, 2])
ax12 = pole_zero_plot_cheby(bands_cheby[2], fs, title='Poles, Zeros,  $H_H(z)$ ')
ax12.set_xlim([0.5, 1.1]); ax12.set_ylim([-0.3, 0.3])
plt.show()
```



```
In [232... def outofband_power_cheby(band, fs):
    nyquist = fs/2
    f = np.linspace(0, nyquist, 5000)
    name, low, high, order, color = band
    b, a = design_cheby1_filter(low, high, fs, order)
    w = 2 * np.pi * f / fs
    _, Hejw = signal.freqz(b, a, worN=w)
    H2 = np.abs(Hejw)**2
    in_band = (f >= low) & (f <= high)
    out_band = ~in_band
    P_in = np.sum(H2[in_band])
    P_out = np.sum(H2[out_band])
    return 100*P_out/P_in

    for i in range(len(bands_cheby)):
        pP_out = outofband_power_cheby(bands_cheby[i], fs)
        print(f'{bands_cheby[i][0]}: {pP_out:1.3f}%, ({bands_cheby[i][1]}, {bands
```

Low: 19.011%, (0,200), N=2

Mid: 18.905%, (200,2000), N=2

High: 1.568%, (2000,22050), N=2

```
In [233... filename, filtype = 'wavFiles/Noise.wav', 'lfilter'
#filename, filtype = 'wavFiles/Music.wav', 'lfilter'
#filename, filtype = 'wavFiles/Voice_Female.wav', 'lfilter'
#filename, filtype = 'wavFiles/Voice_Male.wav', 'lfilter'
#filename, filtype = 'wavFiles/CommSig.wav', 'lfilter'

fs, data = load_and_convert(filename)
print(f'{filename}, {filtype}, Chebyshev Type 1 (3 dB ripple)')
```

```

total_power = calculate_power(data)
print(f"Total Power: {total_power:.5f}")

filtered_signals_cheby = {}
recombined_signal_cheby = np.zeros(data.size)
for name, low, high, order, color in bands_cheby:
    b, a = design_cheby1_filter(low, high, fs, order)
    if filtype == 'filtfilt':
        filtered = apply_filtfilt(data, b, a)
    else:
        filtered = apply_lfilter(data, b, a)
    filtered_signals_cheby[name] = filtered
    recombined_signal_cheby += filtered
    power = calculate_power(filtered)
    print(f"Band {name} ({low}-{high} Hz) Power: {power:.3f} ({(power/total_

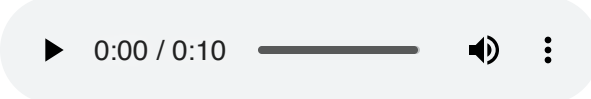
recombined_power = calculate_power(recombined_signal_cheby)
print(f'Recombined Power: {recombined_power:.3f} ({recombined_power/total_pc

filtered_signals_cheby['Orig'] = data
filtered_signals_cheby['Recomb'] = recombined_signal_cheby

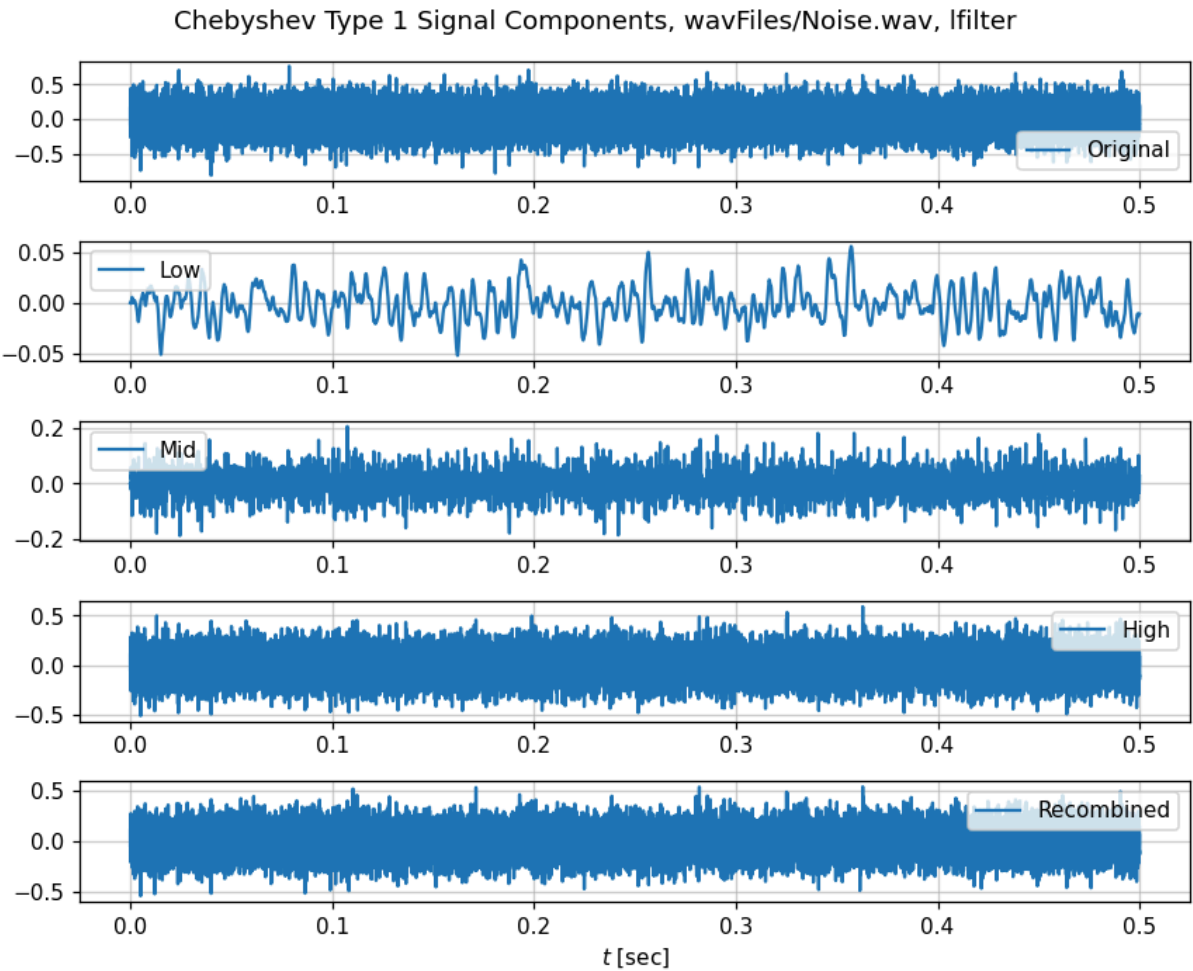
```

wavFiles/Noise.wav, lfilter, Chebyshev Type 1 (3 dB ripple)
 Total Power: 0.03645
 Band Low (0-200 Hz) Power: 0.000 (0.8%)
 Band Mid (200-2000 Hz) Power: 0.003 (7.0%)
 Band High (2000-22050 Hz) Power: 0.019 (52.9%)
 Recombined Power: 0.019 (51.2%)

In [234... sel = ['Orig', 'Low', 'Mid', 'High', 'Recomb']
 Selection = 4 # choose 0...4
 ipd.Audio(filtered_signals_cheby[sel[Selection]], rate=fs)

Out [234... 

In [235... t = np.arange(int(0.5 * fs)) / fs
 fig, ax = plt.subplots(5, 1, figsize=(8, 6.5))
 fig.suptitle(f'Chebyshev Type 1 Signal Components, {filename}, {filtype}')
 ax[0].plot(t, filtered_signals_cheby["Orig"][:len(t)], label='Original')
 ax[0].legend()
 ax[1].plot(t, filtered_signals_cheby["Low"][:len(t)], label='Low')
 ax[1].legend()
 ax[2].plot(t, filtered_signals_cheby["Mid"][:len(t)], label='Mid')
 ax[2].legend()
 ax[3].plot(t, filtered_signals_cheby["High"][:len(t)], label='High')
 ax[3].legend()
 ax[4].plot(t, filtered_signals_cheby["Recomb"][:len(t)], label='Recombined')
 ax[4].set_xlabel('\$t\$ [sec]')
 ax[4].legend()
 plt.tight_layout()
 plt.show()



Bonus: Task 6 Discussion Questions — Chebyshev Type 1 (3 dB ripple)

Question 1: Power Distribution

The following table shows the power distribution across the three bands for each signal. All filters are Chebyshev Type 1, order $N=2$, 3 dB passband ripple. Low band: LPF 0-200 Hz. Mid band: BPF 200-2000 Hz. High band: HPF 2000-22050 Hz.

Signal	Filter	Low (0-200 Hz)	Mid (200-2000 Hz)	High (2000-22050 Hz)	Recombined
Noise	lfilter, N=2	0.8%	7.0%	52.9%	51.2%
Music	lfilter, N=2	25.5%	44.3%	7.0%	44.5%
Voice_Female	lfilter, N=2	3.9%	56.4%	14.1%	41.8%
Voice_Male	lfilter, N=2	19.5%	69.2%	1.6%	42.7%

Signal	Filter	Low (0-200 Hz)	Mid (200-2000 Hz)	High (2000-22050 Hz)	Recombined
CommSig	lfilter, N=2	28.3%	49.7%	4.2%	46.1%

Compared to Butterworth, the Chebyshev filters consistently measure lower power percentages in each band. This is due to the 3 dB passband ripple — the Chebyshev filter attenuates some in-band frequencies by up to 3 dB, reducing measured power even within the passband. The overall distribution trends remain the same: noise is dominated by the high band, voice signals concentrate in the mid band, and the male voice has more low-band power than the female voice. The most dramatic difference is in the noise high band — Butterworth measures 90.0% while Chebyshev measures only 52.9%, because the passband ripple causes significant attenuation across the wide high band for a flat-spectrum signal like white noise.

Question 2: Listening Observations

Filter parameters: Chebyshev Type 1, order N=2, 3 dB passband ripple, lfilter. Low band: LPF 0-200 Hz. Mid band: BPF 200-2000 Hz. High band: HPF 2000-22050 Hz.

Signal	Original	Low Band	Mid Band	High Band	Recombined	Most Recognizable
Noise	Hissing/static	Wind	Static	Rain/static	Static	None — all bands sound like static variations
Music	Normal music	Deep, bass-heavy	Similar to original, slightly lower quality	Tinny, staticky	Slightly lower quality	Mid
Voice_Female	Normal female voice	Muted hum	Normal, slightly lower quality	Quiet, tin-can sound	Slightly lower quality	Mid
Voice_Male	Normal male voice	Muted	Normal, slightly lower quality	Quiet, tinny	Slightly lower quality	Mid
CommSig	Harsh wave/tone	Wind	Quieter version of original	Higher pitch, quieter	Quieter but similar	Mid

The listening results are nearly identical to the Butterworth filters. The same frequency bands are being isolated, so the perceptual content in each band does not change. The mid band remains the most recognizable for all voice and music signals. The 3 dB

passband ripple is not easily perceived by ear — it introduces slight amplitude variation within the passband but does not change which frequencies are passed or rejected. The primary audible difference is that the Chebyshev-filtered bands may sound very slightly quieter overall due to the ripple-induced attenuation.

Question 3: linspace vs logspace for Out-of-Band Power

The answer is the same as for Butterworth filters. np.linspace is the better choice because uniformly spaced frequency points give each frequency bin equal weight in the power sum, providing an unbiased estimate. np.logspace over-represents low frequencies and under-represents high frequencies, distorting the power ratio. This reasoning applies regardless of whether the filter is Butterworth or Chebyshev — it depends only on how the frequency axis is sampled, not on the filter type.

Question 4: LPF vs BPF for Low Band, HPF vs BPF for High Band

The tradeoffs between LPF/BPF for the low band and HPF/BPF for the high band are the same as for Butterworth filters. These tradeoffs concern the choice of band edges (whether to include DC and sub-20 Hz content, whether to include 20-22 kHz content), not the filter shape. Using LPF + BPF + HPF covers the full spectrum for best recombination, while all-BPF gives cleaner band separation at the cost of small spectral gaps. This applies equally to Chebyshev and Butterworth designs.

Question 5: lfilter vs filtfilt

The phase behavior of lfilter vs filtfilt is independent of whether the filter is Butterworth or Chebyshev. lfilter introduces frequency-dependent phase shifts that cause destructive interference when bands are recombined, while filtfilt maintains zero phase for clean reconstruction. The recombined power with lfilter is in the 42-51% range for Chebyshev (compared to 30-83% for Butterworth), and filtfilt would again bring recombination close to 100%. The conclusion remains the same: filtfilt is superior for signal reconstruction, while lfilter is acceptable for isolated band power measurements.

Overall Comparison: Chebyshev vs Butterworth

Aspect	Butterworth	Chebyshev Type 1 (3 dB)
Passband response	Maximally flat	Up to 3 dB ripple
Rolloff steepness	Gradual for same order	Steeper for same order
Band power accuracy	Higher — flat passband preserves amplitude	Lower — ripple attenuates in-band content
Out-of-band rejection	Less rejection at same order	Better rejection at same order
Recombination quality	Similar with lfilter	Similar with lfilter

Aspect	Butterworth	Chebyshev Type 1 (3 dB)
Listening quality	Cleaner in-band sound	Very slightly quieter due to ripple

Butterworth filters are preferred when accurate in-band power measurements and flat passband response matter. Chebyshev Type 1 filters are preferred when steeper rolloff is needed at the same filter order, at the cost of passband ripple. For this audio analysis project, Butterworth is generally the better choice since it preserves signal amplitude more faithfully within each band.