FIR and IIR Filtering of Nerve Signals

1. Plot of `cleanSnap_uV`, `contamSnapBaseline_uV`, and `contamSnapPowerline_uV`. See **Appendix A** for code.

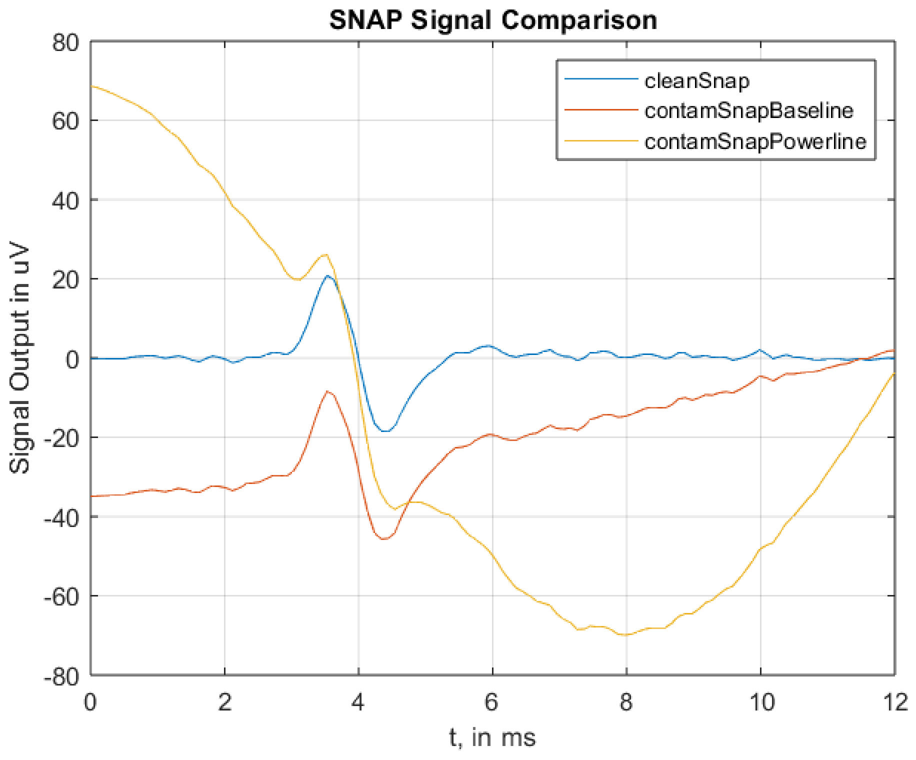

Figure 1: All three signals sampled at $F_s = 10,000$ Hz

Here we can see that the clean signal, `cleanSnap` is a relatively normal SNAP signal with minimal noise. `contamSnapBaseline` adds a low-frequency baseline interference, which here looks as if the output has been lowered and is contantly increasing. `contamSnapPowerline` adds a low-frequency power-line interference, which seems to greatly distort the output.

2. Created analysis function `analyzeSNAP` in Python. See **Appendix B** for code.

3. Plotted the clean SNAP in a separate figure. Running the analysis on the clean SNAP:
   Amplitude (peak to peak) = 39.47167015363368 uV
   Latency = 0.0035 sec
   See code for plot and analysis in **Appendix C**.

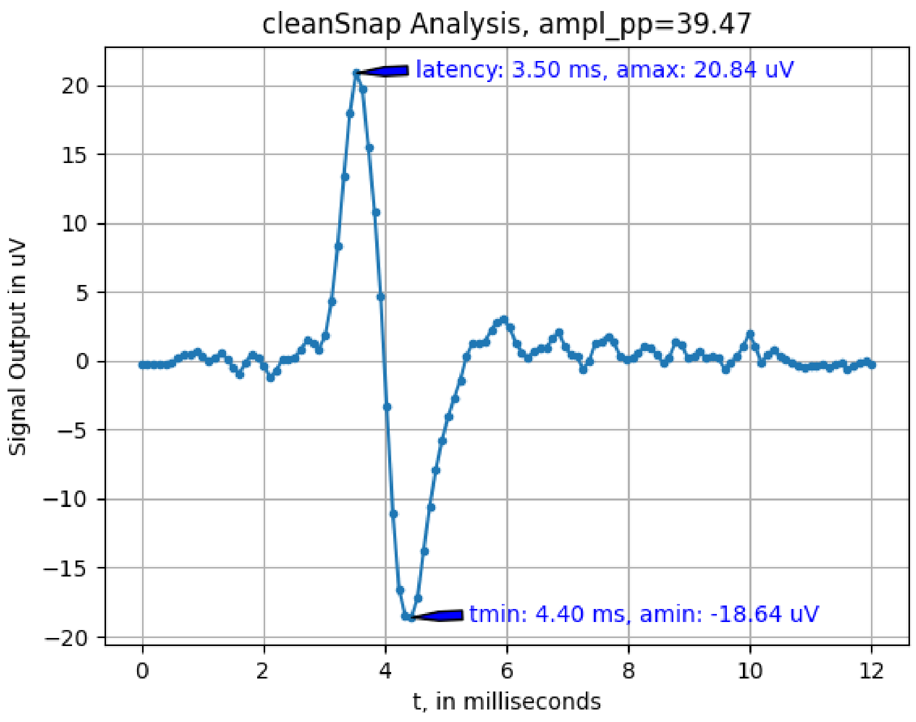

Figure 2: Plot of cleanSnap_uV, with ampl in title and annotated latency

Parameters for this graph are seemingly correct, as they match the original data points.

See next page for Problem 4.

4. Designed an IIR filter by using a 6th-order Butterworth highpass filter with cutoff frequency of 100 Hz. See code for this in **Appendix D**.

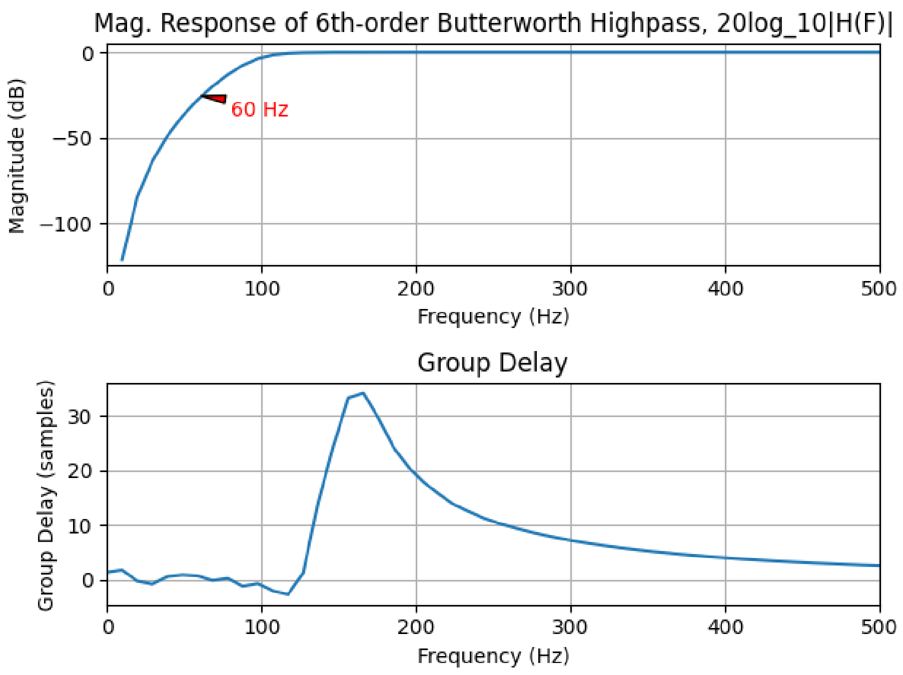

Figure 3: Magnitude Response and Group Delay for IIR Filter

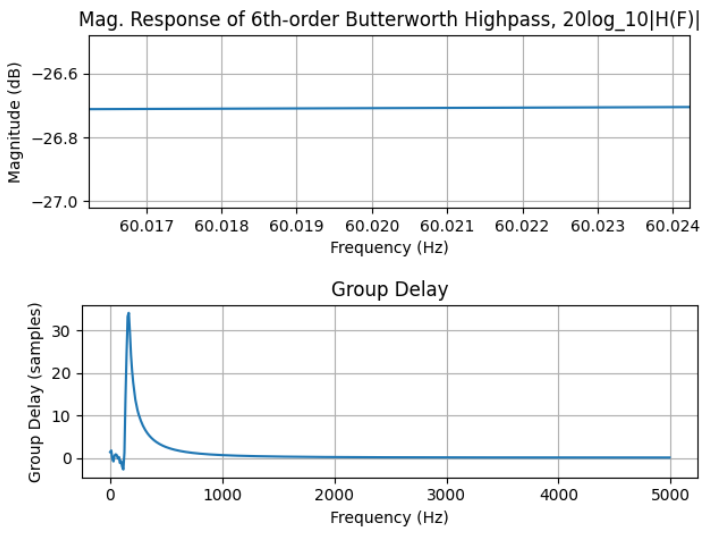

Figure 4: Zoomed In Mag. Response at 60 Hz and Zoomed Out Group Delay

Discussion. Taking a look at the figure with the extremely zoomed in magnitude response at 60 Hz (Figure 4), the filter will suppress the 60 Hz power-line interference at about -26.7 dB. This value was derived visually from the zoomed in graph. In more detail, it is an assumption based on the default linear interpolation from `matplotlib`, as in the code there are only 512 points (default worN) produced for magnitude response. In the actual array, there is never is an exact data point at 60 Hz.

Discussion. Most of the SNAP energy lies in the 200 to 1000 Hz frequency band, which means that this filter will cause significant distortion in the output. The reason for this is that although the magnitude response shows 0 dB for this frequency band meaning no magnitude loss, the group delay is not constant. In fact, from 200 Hz onwards to 1000 Hz, *Figure 4* shows a decaying group delay for increasing frequencies.

This results in the output having significant distortion, as the samples resulting from frequencies closer to 200 Hz will have greater delay than when the frequency is closer to 1000 Hz. Thus, the output will have amplitudes of those lower band frequencies combine with the higher band frequencies.

See next page for Problem 5.

5. Designed an FIR highpass filter with a cutoff frequency of 100 Hz. The example code gave a 51st-order highpass filter. However, for a 51st-order filter, there will be 52 coefficient which is an even number of coefficients. For an even number of coefficients, the signal must have a zero response at the Nyquist frequency, which `cleanSnap` does not. Therefore, we use a 52nd-order filter instead.

   Plot of FIR filter is below, see code in **Appendix E**.

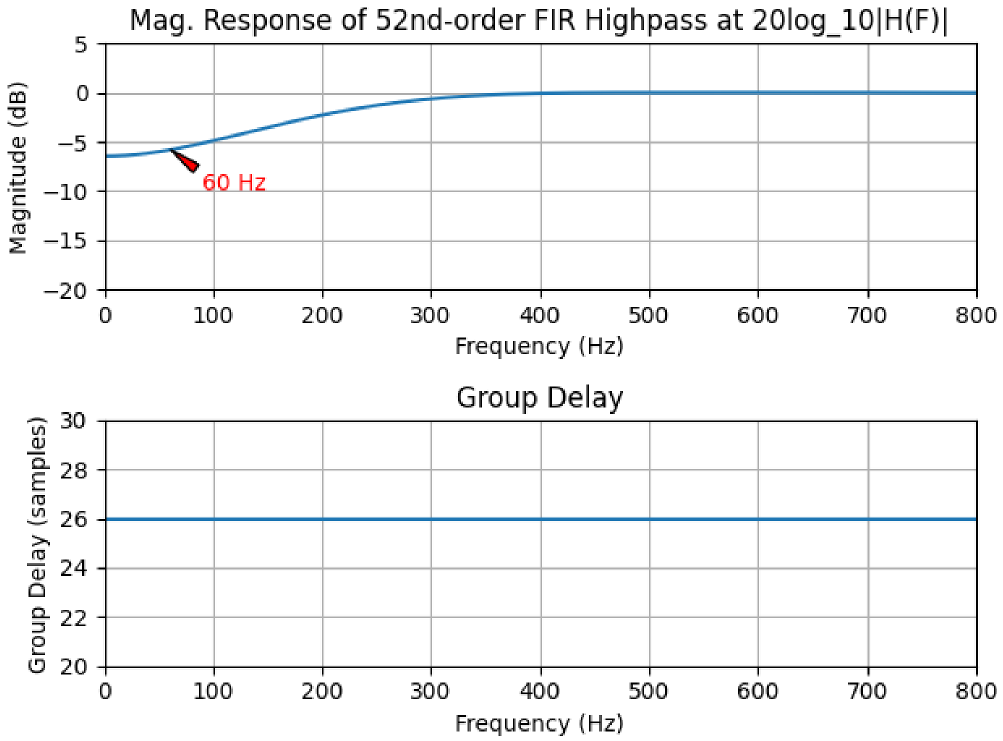

Figure 5: Mag. Response and Group Delay for FIR Filter

See the rest of Problem 5 below.

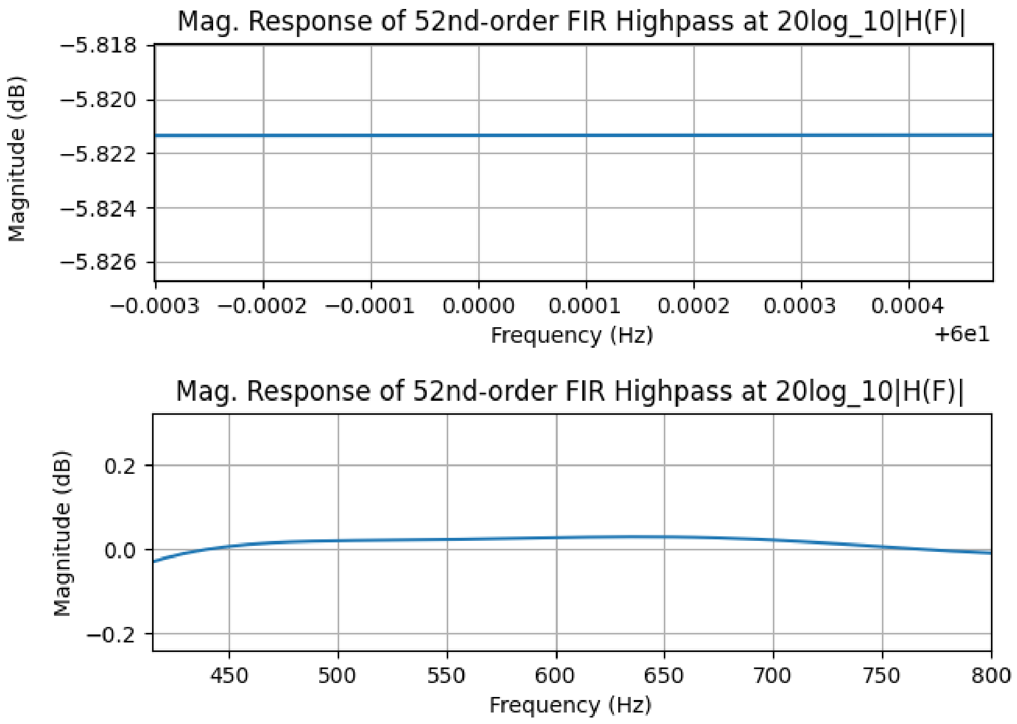

Figure 6: Mag. Response at Different Zooms for FIR Filter

Discussion. The FIR filter suppresses the 60 Hz power-line interference by about -5.82 dB. This value was derived the same way as the previous problem. Zooming the magnitude response deep at 60 Hz can be seen in *Figure 6*, and that value there is from the default linear interpolation that `matplotlib` gives us.

Discussion. Although in *Figure 5* the magnitude response appeared to be 0 dB from about 300 Hz onwards, *Figure 6* shows is that there is actually a slight gain from about 450 to 750 Hz.

Therefore, there should be a magnitude loss (200 to 300 Hz) and slight gain in the frequency range (450 to 750 Hz) containing the majority of SNAP energy (200 to 1000 Hz). An FIR filter produces a constant group delay. This is seen in *Figure 5*, where the group delay is at about 26 samples constant. Therefore, in our signal there won't be any significant distortion other than a constant delay.

6. If the efficiency metric is determined by the number of multiplies required per output sample where more multiplies is worse, the IIR filter is more computationally efficient.

   To determine this, the FIR Window method uses a 52nd-order filter and the IIR Butterworth is 6th order. This means that the FIR has 53 coefficients to multiply per output sample. Then, for the IIR there will be 7 coefficients to multiply per output sample.

   Therefore, the IIR filter is roughly $53/7 = 7.57$ times more efficient than the FIR filter.

7. Here, we run the `cleanSnap` signal through the FIR filter and IIR filter and plot them with the original in one figure with `ampl` and `latency` annotated. See code for this in **Appendix F**.

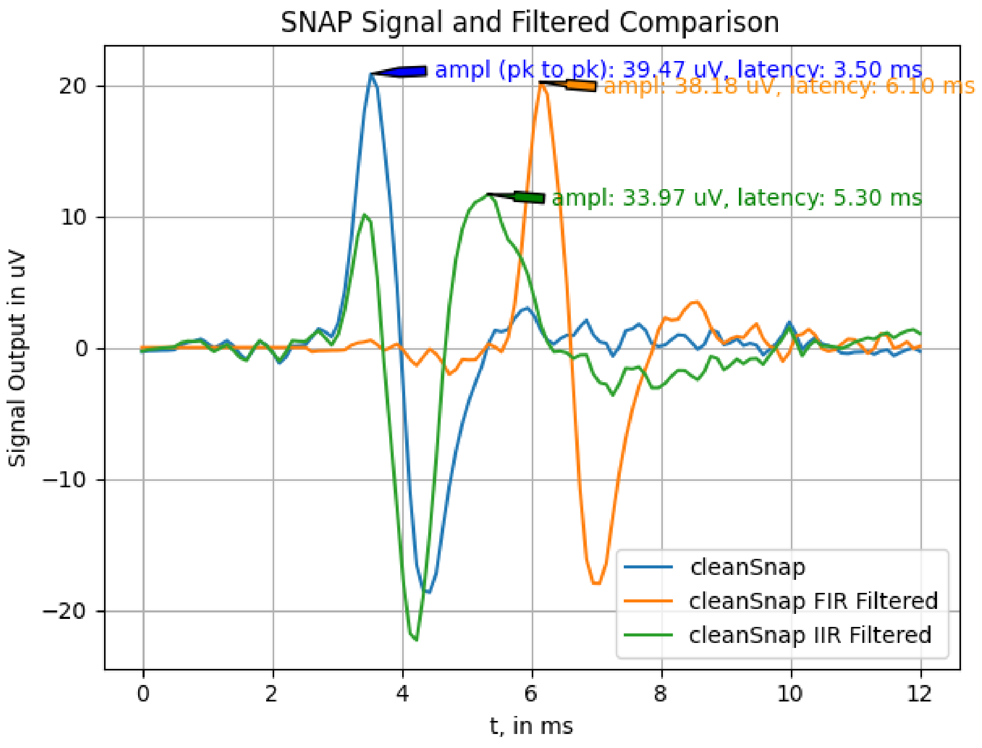

Figure 7: Mag. Response at Different Zooms for FIR Filter

See the rest of Problem 7 below.

(a) *Discussion.* Ideally, assuming `cleanSnap` is not contaminated, the signal would've been unchanged after filtering. However, as we can see that the filters have changed the output, this is not the case. In fact, this slight contamination is mentioned in *Problem 1* where the SNAP signal is described as having minimal noise.

Looking at the output from the FIR filter, the shape is roughly the same. However, as noted earlier, there is a magnitude loss and slight gain in the general SNAP energy frequency band (200 to 1000 Hz). This is observed in the output, where `ampl` is lower than in the original signal, 38.18 uV compared to 39.47 uV, showing the magnitude loss.

In addition, after the maxima and minima, the second maxima of the FIR filtered output is greater than in the original, showing the slight magnitude gain. Finally, as the group delay was constant, the position of outputs has been shifted (increasing latency), though the signal shows no significant distortion otherwise.

Looking at the output from the IIR filter, the shape is significantly altered, demonstrating the effects of the non-linear group delay. Maximas and minimas are pulled around, showing considerable distortion.

In addition, there is a significant magnitude loss at around 300 Hz and below from the IIR filter. This is observed in the `ampl`, where the IIR filtered is 33.87 uV, whereas the original is 39.47 uV

(b) *Determine.* The portion of the observed latency that's caused by the FIR filter is: $6.1 - 3.5ms = 2.6ms$.

The shift (in ms) for an FIR filter of length 21 (20th-order) is likely: $(order/2)/F_s = (20/2)/10000 = 1ms$.

This comes from the assumption that our FIR filter of length 53 (52nd-order) has a group delay of 2.6 ms, which would result from: $(52/2)/F_s = 26/10000 = 2.6ms$

Discussion. If a medical doctor is only interested in measuring the SNAP latency, a rather drastic measure could just be displaying the original `cleanSnap` with different shifts that represent the latency from the filter. This could neglect some rather important characteristics of the filters that alter the signal output though.

See next page for Problem 8.

8. Plots for the filtered contaminated signals are below. See code for this in **Appendix G**.

   (a) Latency and amplitude for both contaminated input and related filter output is annotated on all subplots.

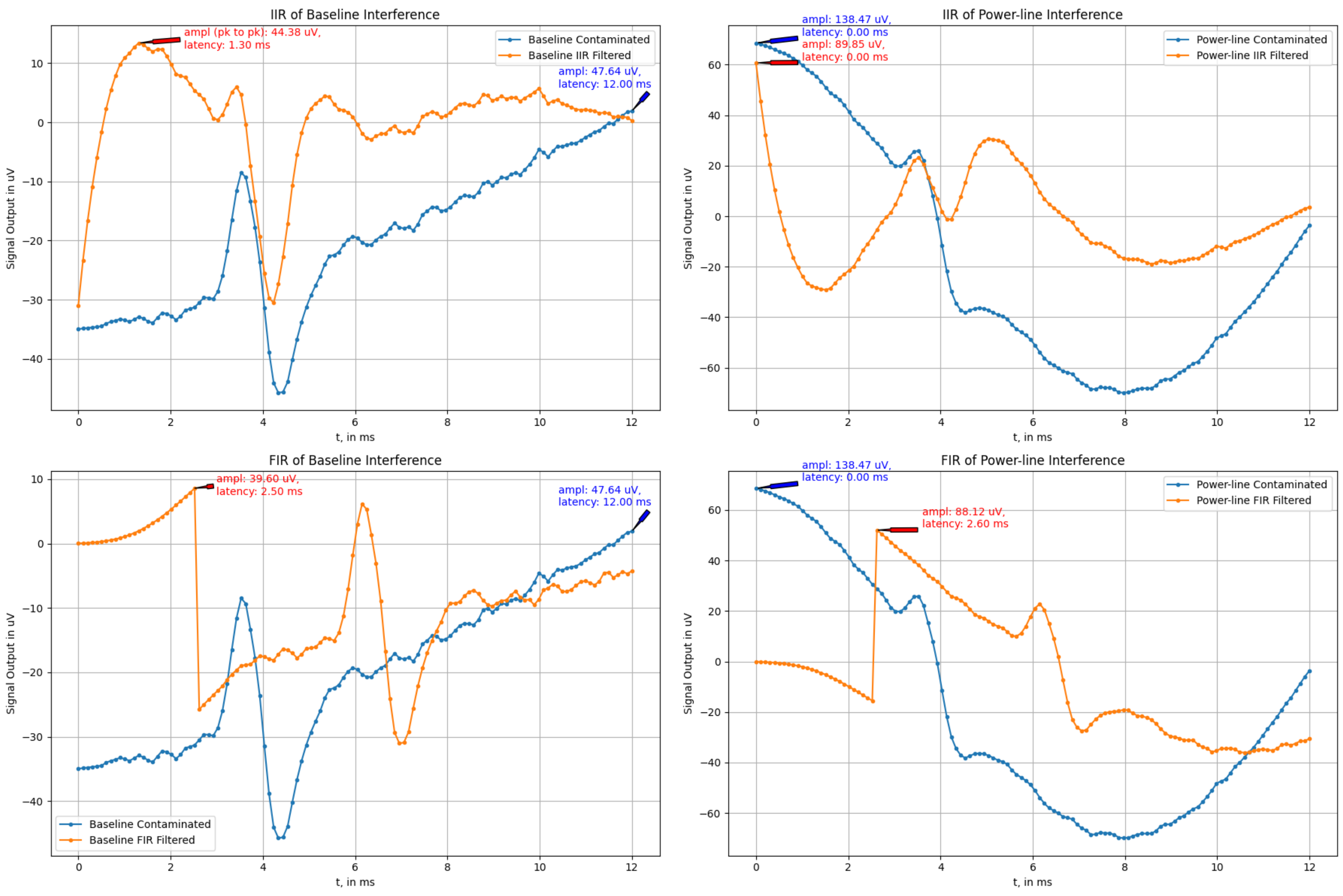

Figure 8: IIR and FIR of Baseline and Power-line Interference

Significant aliasing can be see in both FIR outputs, where the point before or after the point of latency shows an abrupt jump in continuity.

(b) The only valid `analyzeSNAP` measurement seems to be the IIR baseline interference. It has non-zero latency and though shows significant distortion, does not have abrupt jumps in continuity that would demonstrate extreme aliasing. However, it is definitely not a great measurement as the shape of the FIR output matches the non-filtered output rather closely, especially around the non-filtered output's maxima. Therefore, the IIR baseline interference measurement is rather poor.

The invalid measurements are the all non-filtered outputs (0 or 12ms latency), IIR power-line interference, and both FIR baseline and power-line interference outputs. The IIR power-line interference shows a latency of 0 ms. This is physically impossible as when interpreting the latency as the time it takes for the signal to travel and reach its maximum in the nerve. A latency of 0 ms would mean breaking the laws of physics, no travel time needed.

Both FIR outputs show abrupt jumps in continuity, showing extreme aliasing (results from a large negative and positive range in circular convolution) and making them invalid measurements. If the jump was removed, it would show the signal as being continuous and the `ampl` and `latency` values will be significantly different (baseline latency goes to maxima, power-line latency goes to 0 ms).

The cause for these invalid measurements most likely occurs from the way the contamination interacts with the filter. As seen in the baseline, there is a general trend upward in the output, whereas the power-line falls then rises. If this contamination could be accounted for and removed, this could give us more valid measurements.

See next page for Problem 9.

9. Investigation of Baseline Removal technique in figure below. See code for this in **Appendix H**.

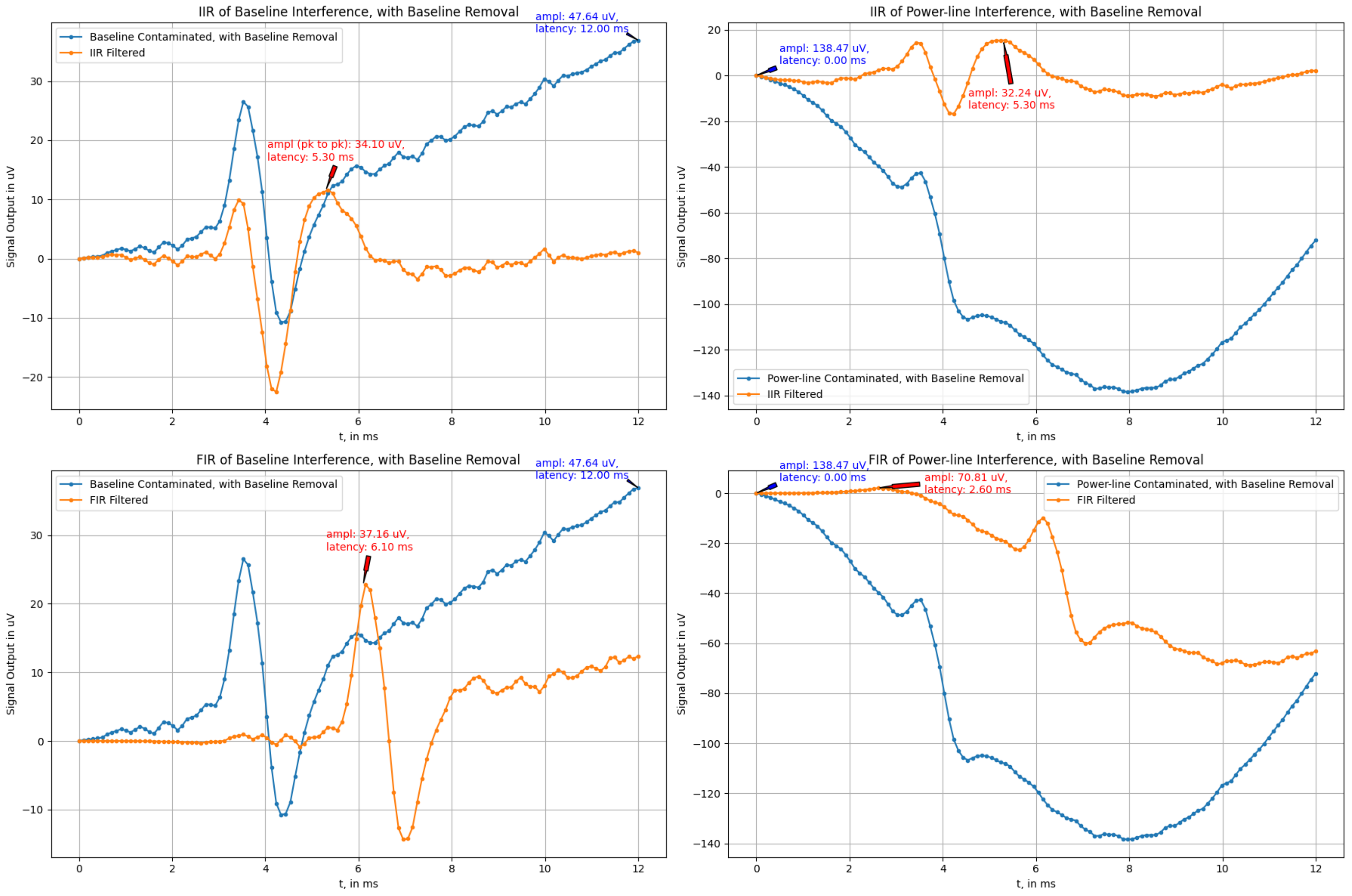

Figure 9: Baseline Removal on Baseline and Power-line Interference

Discussion. The baseline removal seems to improve the results in certain areas. Now, there is no longer any extreme aliasing in the form of jumps in continuity. However, there are similar concerns like in the previous problem about the validity of the measurements.

The only valid measurement is the FIR of baseline interference. The reason for this is that the FIR filter has no significant distortion other than a constant delay. In addition it has minimal magnitude changes. As a result, as the contamination is low-frequency baseline interference, the baseline removal can be seen as shifting all the points upwards so that the first point is at 0 μV. This avoids the previously extreme aliasing due to the large negative and positive range of values when the circular convolution was performed.

As for the invalid measurements, baseline removal can only help so much. For instance, it did not remove the general trends of the graph that distorted output values. Then, for both IIRs, the IIR still has significant distortion from group delay (phase distortion). This changes the shape of the graph, and the second maxima seems more correct as the point of latency.

The FIR of power-line interference is invalid still due to the trend that results from the power-line contamination, where dips down and back up. This causes points at smaller times to have greater outputs, incorrectly skewing the measurement.

See next page for Problem 10.

10. Investigation of removing Phase Distortion in figure below. See code for this in **Appendix I**.

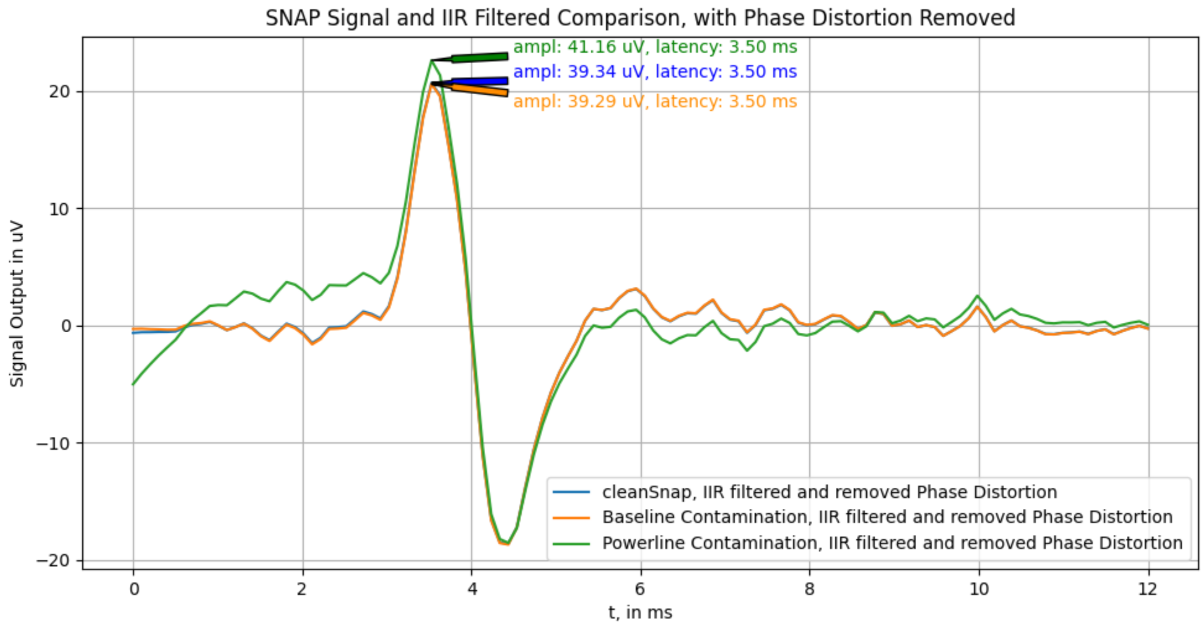

Figure 10: Phase Distortion Removal in SNAP Signal and Filtered Responses

Discussion. This technique was very effective, to where all outputs closely share the same shape and positions. In fact, the latency is the same across all outputs!

There are some minor changes to the amplitude and latency however. For baseline contamination, it has 0.05 uV lower `ampl` when compared to the IIR of `cleanSnap`. The time before the maxima also shows very small differences in the signal output between the two.

For power-line contamination, the shape is close, though in the time before the maxima, the signal is somewhat shifted up. After the minima, the signal is somewhat shifted down. It has an `ampl` higher by 1.92 uV.

However, when compared to the earlier results (without removing phase distortion) on the reference signal, these differences are significant, and for the better. Again, the latency after removing phase distortion is identical, whereas previous results had latency wildly vary depending on contamination and filter. Amplitudes are also only a difference of a uV after removing phase distortion, whereas they varied wildly depending on contamination and filter before.

Thank you for reading!
Also sorry if some of the text reads a bit meandering or loopy.

Appendix

All Python files have the following imports and variables:

```
import numpy as np
import scipy
import matplotlib.pyplot as plt
import pymatreader as pmr

data = pmr.read_mat('NerveData_Project3.mat') # Load data from .mat file
print(f"types {[k for k in data]}")           # Find the keys in dictionary

fs_hZ                       = int(data["fs_hZ"])
cleanSnap_uV                = np.array(data["cleanSnap_uV"])
contamSnapPowerline_uV      = np.array(data["contamSnapPowerline_uV"])
contamSnapBaseline_uV       = np.array(data["contamSnapBaseline_uV"])
```

Appendix A - Problem 1 MATLAB Code

Jump to Problem 1

```
% 1.) Plot the data
% All data is plotted at Fs = 10,000 Hz
% 12 ms data total, uV for data, time vector is in ms
t1 = 12; % in ms
Fs = 10000; % in Hz
t1_ls = linspace(0, t1, t1 * Fs);
t1_space = linspace(0, 12, 120);
plot(t1_space, cleanSnap_uV)
hold on
plot(t1_space, contamSnapBaseline_uV)
plot(t1_space, contamSnapPowerline_uV)
hold off
ylabel("Signal Output in uV")
xlabel("t, in ms")
title("SNAP Signal Comparison")
grid on
legend('cleanSnap', 'contamSnapBaseline', 'contamSnapPowerline')
```

Appendix B - Problem 2 Python Code

Jump to Problem 2

```
# 2.) Create the analysis function
def analyzeSNAP(x, Fs):
    """
    Analyzes SNAP signal and returns several helpful parameters
    :param x:   SNAP Signal x(n), to analyze. Should be a numpy array.
    :param Fs:  Sampling frequency as an integer in Hz
    :return:    ampl, the peak to peak amplitude
                latency, the time it takes in s, for the signal
                to reach its max
                amax, the index in x where the max amplitude occurs
                amin, the index in x where the min amplitude occurs
                tmin, the time it takes in s, for the signal
                to reach its min
    """
    amax = max(x)
    amin = min(x)
    ampl = amax - amin
    latency = x.argmax() / fs_hZ
    tmin = x.argmin() / fs_hZ
    return [ampl, latency, amax, amin, tmin]
```

Appendix C - Problem 3 Python Code

Jump to Problem 3

```
cleansnap_analyze = analyzeSNAP(cleanSnap_uV, fs_hZ)
# print(f"{cleansnap_analyze}")
# ampl, latency, amax, amin, and tmin
# [np.float64(39.47167015363368), np.float64(0.0035),
#  np.float64(20.835104751011546),
#  np.float64(-18.636565402622136), np.float64(0.0044)]
plt.plot(t1_space, cleanSnap_uV, marker='.', label='cleanSnap')
plt.xlabel('t, in milliseconds')
plt.ylabel(r'Signal Output in uV')
```

```
cs_max_index = cleanSnap_uV.argmax()
cs_min_index = cleanSnap_uV.argmin()

# Annotate the latency (as well as amax)
plt.annotate(f'latency: {(cleansnap_analyze[1]*fs_hZ / 10):.2f} ms, \
             amax: {cleansnap_analyze[2]:.2f} uV',
             xy=(cs_max_index / 10, cleanSnap_uV[cs_max_index]),
             xytext=(cs_max_index / 10 + 1, cleanSnap_uV[cs_max_index] - 0.25),
             arrowprops=dict(facecolor='b', headwidth=4, shrink=0.05),
             color='b')
plt.annotate(f'tmin: {(cleansnap_analyze[4]*fs_hZ / 10):.2f} ms, \
             amin: {cleansnap_analyze[3]:.2f} uV',
             xy=(cs_min_index / 10, cleanSnap_uV[cs_min_index]),
             xytext=(cs_min_index / 10 + 1, cleanSnap_uV[cs_min_index] - 0.25),
             arrowprops=dict(facecolor='b', headwidth=4, shrink=0.05),
             color='b')
plt.title(f'cleanSnap Analysis, ampl_pp={cleansnap_analyze[0]:.2f}')
plt.grid()
plt.show()
```

Appendix D - Problem 4 Python Code

Jump to Problem 4

```
# 4.) Design a IIR highpass and plot freq response and group delay in Hz
#       Given: 6th order Butterworth highpass filter, w/cutoff frequency
#              of 100Hz
[b_iir, a_iir] = scipy.signal.butter(6,100/(Fs/2),'highpass')
# Computes frequency response
freq, h = scipy.signal.freqz(b_iir, a_iir) # default 512 points, or worN
# Convert freq. to Hz
freq_hz = freq * fs_hZ / (2 * np.pi)
# Get magnitude in decibels
mag_dec = 20 * np.log10(np.abs(h[1:])) # took out first value, as it is zero.
# print(f'min h:{np.where(h == 0)}, at 60hz: \
#               {20 * np.log10(np.abs(h[np.wher]))}')
print(f"at 60hz of freq: {freq_hz}")
```

```
# Get group delay
freq_grpdelay, grpdelay = scipy.signal.group_delay([b_iir, a_iir], fs=fs_hZ)

# Plotting time!
plt.subplot(2, 1, 1)
# took out first value in mag_dec, likewise in freq_hz
plt.plot(freq_hz[1:], mag_dec)
plt.annotate(f'60 Hz',
             xy=(60 , mag_dec[4] + 12),
             xytext=(60 + 20, mag_dec[4]),
             arrowprops=dict(facecolor='r', headwidth=4, shrink=0.05),
             color='r')
plt.title('Mag. Response of 6th-order Butterworth Highpass, 20log_10|H(F)|')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude (dB)')
plt.grid(True)
# plt.xlim(0, 500)  # freq zoom
plt.ylim(-125, 5)  # mag zoom

# Plot the group delay
plt.subplot(2, 1, 2)
plt.plot(freq_grpdelay, grpdelay)
plt.title('Group Delay')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Group Delay (samples)')
plt.grid(True)
# plt.xlim(0, 500) # freq zoom

# Show the plots
plt.tight_layout()
plt.show()
```

Appendix E - Problem 5 Python Code

Jump to Problem 5

```
# 5.) Design a FIR highpass
# produces an array with (length), eg. 52 coefficients. NOT FILTER ORDER.
# Filter order is coefficients - 1. So 52 coefficients -> 51st-order
# cannot work with an even length, as an even length = even # of coefficients
# which must have a zero response at Nyquist freq. Thus, we give length=53
b_fir = scipy.signal.firwin(53, 100/(fs_hZ/2), pass_zero='highpass')

# Plot Mag and GRPDELAY

# Get frequency response of FIR
freq_fir, h_fir = scipy.signal.freqz(b_fir) # default 512 points, or worN
# Convert freq to Hz
freq_hz_fir = freq_fir * fs_hZ / (2 * np.pi)
# Get magnitude in dB
mag_db_fir = 20 * np.log10(np.abs(h_fir))

# Get group delay
freq_grpdelay_fir, grpdelay_fir = scipy.signal.group_delay([b_fir, 1], fs=fs_hZ)

# Plotting time!
plt.subplot(2, 1, 1)
plt.plot(freq_hz_fir, mag_db_fir)
plt.annotate(f'60 Hz',
             xy=(60 , -5.7),
             xytext=(60 + 30, -10),
             arrowprops=dict(facecolor='r', headwidth=4, shrink=0.05),
             color='r')
plt.title('Mag. Response of 52nd-order FIR Highpass at 20log_10|H(F)|')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude (dB)')
plt.grid(True)
plt.xlim(0, 800)  # zoom freq
plt.ylim(-20, 5)  # zoom mag
# Plot the group delay
plt.subplot(2, 1, 2)
plt.plot(freq_grpdelay_fir, grpdelay_fir)
plt.title('Group Delay')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Group Delay (samples)')
plt.grid(True)
plt.xlim(0, 800) # zoom freq
plt.ylim(20, 30)  # zoom mag
# Show the plots
plt.tight_layout()
plt.show()
```

Appendix F - Problem 7 Python Code

Jump to Problem 7

```
# 7.) Filter the reference signal
# 6th-order Butterworth
cleanSnap_uV_iir_filtered = scipy.signal.lfilter(b_iir, a_iir, cleanSnap_uV)
# 52nd-order FIR Window
cleanSnap_uV_fir_filtered = scipy.signal.lfilter(b_fir, 1, cleanSnap_uV)

# Plotting time!
plt.plot(t1_space, cleanSnap_uV)
cleansnap_analyze = analyzeSNAP(cleanSnap_uV, fs_hZ)
cleanSnap_max_index = cleanSnap_uV.argmax()
print(cleanSnap_max_index)
# annotate ampl and latency for clean signal
plt.annotate(f'ampl (pk to pk): {cleansnap_analyze[0]:.2f} uV, '
             f'latency: {(cleansnap_analyze[1]*fs_hZ / 10):.2f} ms',
             xy=(cs_max_index / 10, cleanSnap_uV[cs_max_index]),
             xytext=(cs_max_index / 10 + 1, cleanSnap_uV[cs_max_index] - 0.25),
             arrowprops=dict(facecolor='b', headwidth=4, shrink=0.05),
             color='b')

plt.plot(t1_space, cleanSnap_uV_fir_filtered)
# analyze fir filtered
firfiltered_analyze = analyzeSNAP(cleanSnap_uV_fir_filtered, fs_hZ)
csfirfilt_maxindex = cleanSnap_uV_fir_filtered.argmax()
print(csfirfilt_maxindex)
# annotate ampl and latency for fir filtered
plt.annotate(f'ampl: {firfiltered_analyze[0]:.2f} uV, '
             f'latency: {(firfiltered_analyze[1]*fs_hZ / 10):.2f} ms',
             xy=(firfiltered_analyze[1] * fs_hZ / 10, firfiltered_analyze[2]),
             xytext=(firfiltered_analyze[1] * fs_hZ / 10 + 1, \
                     firfiltered_analyze[2] - 0.9),
             arrowprops=dict(facecolor='darkorange', headwidth=2, shrink=0.03),
             color='darkorange')

plt.plot(t1_space, cleanSnap_uV_iir_filtered)
# analyze iir filtered
iirfiltered_analyze = analyzeSNAP(cleanSnap_uV_iir_filtered, fs_hZ)
csiirfilt_maxindex = cleanSnap_uV_iir_filtered.argmax()
print(csiirfilt_maxindex)
# annotate ampl and latency for fir filtered
plt.annotate(f'ampl: {iirfiltered_analyze[0]:.2f} uV, '
             f'latency: {(iirfiltered_analyze[1]*fs_hZ / 10):.2f} ms',
             xy=(iirfiltered_analyze[1] * fs_hZ / 10, iirfiltered_analyze[2]),
             xytext=(iirfiltered_analyze[1] * fs_hZ / 10 + 1, \
                     iirfiltered_analyze[2] - 0.9),
             arrowprops=dict(facecolor='green', headwidth=2, shrink=0.03),
             color='green')

plt.ylabel("Signal Output in uV")
plt.xlabel("t, in ms")
plt.title("SNAP Signal and Filtered Comparison")
plt.grid()
plt.legend(['cleanSnap', 'cleanSnap FIR Filtered', 'cleanSnap IIR Filtered'])

plt.tight_layout()
plt.show()
```

Appendix G - Problem 8 Python Code

Jump to Problem 8

```
# 8.) Filter the contaminated signals
#       IIR Baseline | IIR Powerline
#       FIR Baseline | IIR Powerline

# baseline
```

```
baseline_analyze = analyzeSNAP(contamSnapBaseline_uV, fs_hZ)
baseline_fir_filtered = scipy.signal.lfilter(b_fir,
                                             1,
                                             contamSnapBaseline_uV)
baseline_fir_analyze = analyzeSNAP(baseline_fir_filtered, fs_hZ)

baseline_iir_filtered = scipy.signal.lfilter(b_iir,
                                             a_iir,
                                             contamSnapBaseline_uV)
baseline_iir_analyze = analyzeSNAP(baseline_iir_filtered, fs_hZ)

# powerline
powerline_analyze = analyzeSNAP(contamSnapPowerline_uV, fs_hZ)
powerline_fir_filtered = scipy.signal.lfilter(b_fir,
                                              1,
                                              contamSnapPowerline_uV)
powerline_fir_analyze = analyzeSNAP(powerline_fir_filtered, fs_hZ)

powerline_iir_filtered = scipy.signal.lfilter(b_iir,
                                              a_iir,
                                              contamSnapPowerline_uV)
powerline_iir_analyze = analyzeSNAP(powerline_iir_filtered, fs_hZ)

# 8a.) 2x2 subplot (contains contaminated input standalone
#                   and filtered output)
figure, axs = plt.subplots(2, 2, figsize=(18, 12))

# print(contamSnapBaseline_uV.argmax())
# print(contamSnapBaseline_uV[119])
# print(contamSnapBaseline_uV)

# iir baseline
axs[0, 0].plot(t1_space, contamSnapBaseline_uV, marker='.')
axs[0, 0].annotate(f'ampl: {baseline_analyze[0]:.2f} uV,\n'
                   f'latency: {(baseline_analyze[1] *fs_hZ / 10 + 0.1):.2f}'
                   f' ms',
                   xy=(baseline_analyze[1] * fs_hZ / 10 + 0.1,
                       baseline_analyze[2]),
                   xytext=(baseline_analyze[1] * fs_hZ / 10 - 1.5,
                           baseline_analyze[2] + 4),
                   arrowprops=dict(facecolor='blue',
                                   headwidth=2,
                                   shrink=0.05),
                   color='blue')
axs[0, 0].plot(t1_space, baseline_iir_filtered, marker='.')
axs[0, 0].annotate(f'ampl (pk to pk): {baseline_iir_analyze[0]:.2f} uV,\n'
                   f'latency: {(baseline_iir_analyze[1]*fs_hZ / 10):.2f} ms',
                   xy=(baseline_iir_analyze[1] * fs_hZ / 10,
                       baseline_iir_analyze[2]),
                   xytext=(baseline_iir_analyze[1] * fs_hZ / 10 + 1,
                           baseline_iir_analyze[2] - 0.9),
                   arrowprops=dict(facecolor='red',
                                   headwidth=2,
                                   shrink=0.05),
                   color='red')
axs[0, 0].set_ylabel("Signal Output in uV")
axs[0, 0].set_xlabel("t, in ms")
axs[0, 0].set_title("IIR of Baseline Interference")
axs[0, 0].grid(True)
axs[0, 0].legend(['Baseline Contaminated', 'Baseline IIR Filtered'])

# iir powerline
axs[0, 1].plot(t1_space, contamSnapPowerline_uV, marker='.')
axs[0, 1].annotate(f'ampl: {powerline_analyze[0]:.2f} uV,\n'
                   f'latency: {(powerline_analyze[1] *fs_hZ / 10):.2f} ms',
                   xy=(powerline_analyze[1] * fs_hZ / 10,
                       powerline_analyze[2]),
                   xytext=(powerline_analyze[1] * fs_hZ / 10 + 1,
                           powerline_analyze[2] + 3),
                   arrowprops=dict(facecolor='blue',
                                   headwidth=2,
                                   shrink=0.05),
                   color='blue')
axs[0, 1].plot(t1_space, powerline_iir_filtered, marker='.')
axs[0, 1].annotate(f'ampl: {powerline_iir_analyze[0]:.2f} uV,\n'
                   f'latency: {(powerline_iir_analyze[1]*fs_hZ / 10):.2f} ms',
                    xy=(powerline_iir_analyze[1] * fs_hZ / 10,
                        powerline_iir_analyze[2]),
                    xytext=(powerline_iir_analyze[1] * fs_hZ / 10 + 1,
                            powerline_iir_analyze[2] + 1),
                    arrowprops=dict(facecolor='red',
                                    headwidth=2,
                                    shrink=0.05),
                    color='red')
axs[0, 1].set_ylabel("Signal Output in uV")
axs[0, 1].set_xlabel("t, in ms")
axs[0, 1].set_title("IIR of Power-line Interference")
axs[0, 1].grid(True)
axs[0, 1].legend(['Power-line Contaminated', 'Power-line IIR Filtered'])

# FIR baseline
axs[1, 0].plot(t1_space, contamSnapBaseline_uV, marker='.')
axs[1, 0].annotate(f'ampl: {baseline_analyze[0]:.2f} uV,\n'
                   f'latency: {(baseline_analyze[1] *fs_hZ / 10 + 0.1):.2f}'
                   f' ms',
                   xy=(baseline_analyze[1] * fs_hZ / 10 + 0.1,
                       baseline_analyze[2]),
                   xytext=(baseline_analyze[1] * fs_hZ / 10 - 1.5,
                           baseline_analyze[2] + 4),
                   arrowprops=dict(facecolor='blue',
                                   headwidth=2,
                                   shrink=0.05),
                   color='blue')
axs[1, 0].plot(t1_space, baseline_fir_filtered, marker='.')
axs[1, 0].annotate(f'ampl: {baseline_fir_analyze[0]:.2f} uV,\n'
                   f'latency: {(baseline_fir_analyze[1]*fs_hZ / 10):.2f} ms',
                   xy=(baseline_fir_analyze[1] * fs_hZ / 10,
                       baseline_fir_analyze[2]),
                   xytext=(baseline_fir_analyze[1] * fs_hZ / 10 + 0.5,
                           baseline_fir_analyze[2] - 1),
                   arrowprops=dict(facecolor='red',
                                   headwidth=2,
                                   shrink=0.05),
                   color='red')
```

```
axs[1, 0].set_ylabel("Signal Output in uV")
axs[1, 0].set_xlabel("t, in ms")
axs[1, 0].set_title("FIR of Baseline Interference")
axs[1, 0].grid(True)
axs[1, 0].legend(['Baseline Contaminated', 'Baseline FIR Filtered'])

# FIR powerline
axs[1, 1].plot(t1_space, contamSnapPowerline_uV, marker='.')
axs[1, 1].annotate(f'ampl: {powerline_analyze[0]:.2f} uV,\n'
                   f'latency: {(powerline_analyze[1] *fs_hZ / 10):.2f} ms',
                   xy=(powerline_analyze[1] * fs_hZ / 10,
                       powerline_analyze[2]),
                   xytext=(powerline_analyze[1] * fs_hZ / 10 + 1,
                           powerline_analyze[2] + 3),
                   arrowprops=dict(facecolor='blue',
                                   headwidth=2,
                                   shrink=0.05),
                   color='blue')
axs[1, 1].plot(t1_space, powerline_fir_filtered, marker='.')
axs[1, 1].annotate(f'ampl: {powerline_fir_analyze[0]:.2f} uV,\n'
                   f'latency: {(powerline_fir_analyze[1]*fs_hZ / 10):.2f} ms',
                   xy=(powerline_fir_analyze[1] * fs_hZ / 10,
                       powerline_fir_analyze[2]),
                   xytext=(powerline_fir_analyze[1] * fs_hZ / 10 + 1,
                           powerline_fir_analyze[2] + 1),
                   arrowprops=dict(facecolor='red',
                                   headwidth=2,
                                   shrink=0.05),
                   color='red')
axs[1, 1].set_ylabel("Signal Output in uV")
axs[1, 1].set_xlabel("t, in ms")
axs[1, 1].set_title("FIR of Power-line Interference")
axs[1, 1].grid(True)
axs[1, 1].legend(['Power-line Contaminated', 'Power-line FIR Filtered'])

plt.tight_layout()
plt.show()
```

Appendix H - Problem 9 Python Code

Jump to Problem 9

```
# 9.) Investigate Baseline Removal technique x_m[n] = x[n] - x[1]
#       IIR Baseline | IIR Powerline
#       FIR Baseline | IIR Powerline

# removing baseline from contamSnapBaseline
baseline_baselineremoval = np.array([x - contamSnapBaseline_uV[0]
                                     for x in contamSnapBaseline_uV])
# removing baseline from contamSnapPowerline
powerline_baselineremoval = np.array([x - contamSnapPowerline_uV[0]
                                      for x in contamSnapPowerline_uV])

# baseline - blr = baseline removed
baseline_blr_analyze = analyzeSNAP(baseline_baselineremoval, fs_hZ)
baseline_blr_fir_filtered = scipy.signal.lfilter(b_fir,
                                                 1,
                                                 baseline_baselineremoval)
baseline_blr_fir_analyze = analyzeSNAP(baseline_blr_fir_filtered, fs_hZ)

baseline_blr_iir_filtered = scipy.signal.lfilter(b_iir,
                                                 a_iir,
                                                 baseline_baselineremoval)
baseline_blr_iir_analyze = analyzeSNAP(baseline_blr_iir_filtered, fs_hZ)

# powerline
powerline_blr_analyze = analyzeSNAP(powerline_baselineremoval, fs_hZ)
powerline_blr_fir_filtered = scipy.signal.lfilter(b_fir,
                                                  1,
                                                  powerline_baselineremoval)
powerline_blr_fir_analyze = analyzeSNAP(powerline_blr_fir_filtered, fs_hZ)

powerline_blr_iir_filtered = scipy.signal.lfilter(b_iir,
                                                  a_iir,
                                                  powerline_baselineremoval)
powerline_blr_iir_analyze = analyzeSNAP(powerline_blr_iir_filtered, fs_hZ)
```

```
# 2x2 subplot (contains baseline removed input standalone and filtered output)
figure, axs = plt.subplots(2, 2, figsize=(18, 12))

# iir baseline
axs[0, 0].plot(t1_space, baseline_baselineremoval, marker='.')
axs[0, 0].annotate(f'ampl: {baseline_blr_analyze[0]:.2f} uV,\n'
                   f'latency: {(baseline_blr_analyze[1] *fs_hZ / 10 + 0.1):.2f}'
                   f' ms',
                   xy=(baseline_blr_analyze[1] * fs_hZ / 10 + 0.1,
                       baseline_blr_analyze[2]),
                   xytext=(baseline_blr_analyze[1] * fs_hZ / 10 - 2.1,
                           baseline_blr_analyze[2] + 1.3),
                   arrowprops=dict(facecolor='blue',
                                   headwidth=2,
                                   shrink=0.05),
                   color='blue')
axs[0, 0].plot(t1_space, baseline_blr_iir_filtered, marker='.')
axs[0, 0].annotate(f'ampl (pk to pk): {baseline_blr_iir_analyze[0]:.2f} uV,\n'
                   f'latency: {(baseline_blr_iir_analyze[1]*fs_hZ / 10):.2f}'
                   f' ms',
                   xy=(baseline_blr_iir_analyze[1] * fs_hZ / 10,
                       baseline_blr_iir_analyze[2]),
                   xytext=(baseline_blr_iir_analyze[1] * fs_hZ / 10 - 1.25,
                           baseline_blr_iir_analyze[2] + 5),
                   arrowprops=dict(facecolor='red',
                                   headwidth=2,
                                   shrink=0.05),
                   color='red')
axs[0, 0].set_ylabel("Signal Output in uV")
axs[0, 0].set_xlabel("t, in ms")
axs[0, 0].set_title("IIR of Baseline Interference, with Baseline Removal")
axs[0, 0].grid(True)
axs[0, 0].legend(['Baseline Contaminated, with Baseline Removal',
                  'IIR Filtered'])

# iir powerline
axs[0, 1].plot(t1_space, powerline_baselineremoval, marker='.')
axs[0, 1].annotate(f'ampl: {powerline_blr_analyze[0]:.2f} uV,\n'
                   f'latency: {(powerline_blr_analyze[1] *fs_hZ / 10):.2f}'
                   f' ms',
                   xy=(powerline_blr_analyze[1] * fs_hZ / 10,
                       powerline_blr_analyze[2]),
                   xytext=(powerline_blr_analyze[1] * fs_hZ / 10 + 0.5,
                           powerline_blr_analyze[2] + 4.9),
                   arrowprops=dict(facecolor='blue',
                                   headwidth=2,
                                   shrink=0.05),
                   color='blue')
axs[0, 1].plot(t1_space, powerline_blr_iir_filtered, marker='.')
axs[0, 1].annotate(f'ampl: {powerline_blr_iir_analyze[0]:.2f} uV,\n'
                   f'latency: {(powerline_blr_iir_analyze[1]*fs_hZ / 10):.2f}'
                   f' ms',
                   xy=(powerline_blr_iir_analyze[1] * fs_hZ / 10,
                       powerline_blr_iir_analyze[2]),
                   xytext=(powerline_blr_iir_analyze[1] * fs_hZ / 10 - 0.75,
                           powerline_blr_iir_analyze[2] - 30),
                   arrowprops=dict(facecolor='red',
                                   headwidth=2,
                                   shrink=0.05),
                   color='red')
axs[0, 1].set_ylabel("Signal Output in uV")
axs[0, 1].set_xlabel("t, in ms")
axs[0, 1].set_title("IIR of Power-line Interference, with Baseline Removal")
axs[0, 1].grid(True)
axs[0, 1].legend(['Power-line Contaminated, with Baseline Removal',
                  'IIR Filtered'])

# FIR baseline
axs[1, 0].plot(t1_space, baseline_baselineremoval, marker='.')
axs[1, 0].annotate(f'ampl: {baseline_blr_analyze[0]:.2f} uV,\n'
                   f'latency: {(baseline_blr_analyze[1] *fs_hZ / 10 + 0.1):.2f}'
                   f' ms',
                   xy=(baseline_blr_analyze[1] * fs_hZ / 10 + 0.1,
                       baseline_blr_analyze[2]),
                   xytext=(baseline_blr_analyze[1] * fs_hZ / 10 - 2.1,
                           baseline_blr_analyze[2] + 1.3),
                   arrowprops=dict(facecolor='blue',
                                   headwidth=2,
                                   shrink=0.05),
                   color='blue')
axs[1, 0].plot(t1_space, baseline_blr_fir_filtered, marker='.')
axs[1, 0].annotate(f'ampl: {baseline_blr_fir_analyze[0]:.2f} uV,\n'
                   f'latency: {(baseline_blr_fir_analyze[1]*fs_hZ / 10):.2f}'
                   f' ms',
                   xy=(baseline_blr_fir_analyze[1] * fs_hZ / 10,
                       baseline_blr_fir_analyze[2]),
                   xytext=(baseline_blr_fir_analyze[1] * fs_hZ / 10 - 0.8,
                           baseline_blr_fir_analyze[2] + 5),
                   arrowprops=dict(facecolor='red',
                                   headwidth=2,
                                   shrink=0.05),
                   color='red')
axs[1, 0].set_ylabel("Signal Output in uV")
axs[1, 0].set_xlabel("t, in ms")
axs[1, 0].set_title("FIR of Baseline Interference, with Baseline Removal")
axs[1, 0].grid(True)
axs[1, 0].legend(['Baseline Contaminated, with Baseline Removal',
                  'FIR Filtered'])

# FIR powerline
axs[1, 1].plot(t1_space, powerline_baselineremoval, marker='.')
axs[1, 1].annotate(f'ampl: {powerline_blr_analyze[0]:.2f} uV,\n'
                   f'latency: {(powerline_blr_analyze[1] *fs_hZ / 10):.2f}'
                   f' ms',
                   xy=(powerline_blr_analyze[1] * fs_hZ / 10,
                       powerline_blr_analyze[2]),
                   xytext=(powerline_blr_analyze[1] * fs_hZ / 10 + 0.5,
                           powerline_blr_analyze[2] + 4.9),
                   arrowprops=dict(facecolor='blue',
                                   headwidth=2,
                                   shrink=0.05),
                   color='blue')
axs[1, 1].plot(t1_space, powerline_blr_fir_filtered, marker='.')
axs[1, 1].annotate(f'ampl: {powerline_blr_fir_analyze[0]:.2f} uV,\n'
                   f'latency: {(powerline_blr_fir_analyze[1]*fs_hZ / 10):.2f}'
                   f' ms',
                   xy=(powerline_blr_fir_analyze[1] * fs_hZ / 10,
                       powerline_blr_fir_analyze[2]),
                   xytext=(powerline_blr_fir_analyze[1] * fs_hZ / 10 + 1,
                           powerline_blr_fir_analyze[2] - 2),
                   arrowprops=dict(facecolor='red',
                                   headwidth=2,
                                   shrink=0.05),
                   color='red')
axs[1, 1].set_ylabel("Signal Output in uV")
axs[1, 1].set_xlabel("t, in ms")
axs[1, 1].set_title("FIR of Power-line Interference, with Baseline Removal")
axs[1, 1].grid(True)
axs[1, 1].legend(['Power-line Contaminated, with Baseline Removal',
                  'FIR Filtered'])

plt.tight_layout()
plt.show()
```

Appendix I - Problem 10 Python Code

Jump to Problem 10

```
# 10.) Remove phase distortion
# Perform filtfilt and analyzeSNAP to graph IIR filter's response
#   on all 3 signals
clean_iir_filtfilt = scipy.signal.filtfilt(b_iir,
                                           a_iir,
                                           cleanSnap_uV)
clean_ffanalyze = analyzeSNAP(clean_iir_filtfilt, fs_hZ)

baseline_iir_filtfilt = scipy.signal.filtfilt(b_iir,
                                              a_iir,
                                              contamSnapBaseline_uV)
baseline_ffanalyze = analyzeSNAP(baseline_iir_filtfilt, fs_hZ)

powerline_iir_filtfilt = scipy.signal.filtfilt(b_iir,
                                              a_iir,
                                              contamSnapPowerline_uV)
powerline_ffanalyze = analyzeSNAP(powerline_iir_filtfilt, fs_hZ)

# Plotting time!
plt.plot(t1_space, clean_iir_filtfilt)
# annotate ampl and latency for the cleanSnap that's been
#   filtfilt w/IIR filter
plt.annotate(f'ampl: {clean_ffanalyze[0]:.2f} uV, '
             f'latency: {(clean_ffanalyze[1]*fs_hZ / 10):.2f} ms',
             xy=(clean_ffanalyze[1] * fs_hZ / 10,
                 clean_ffanalyze[2]),
             xytext=(clean_ffanalyze[1] * fs_hZ / 10 + 1,
                     clean_ffanalyze[2] + 0.5),
             arrowprops=dict(facecolor='blue', headwidth=2, shrink=0.03),
             color='blue')

plt.plot(t1_space, baseline_iir_filtfilt)
plt.annotate(f'ampl: {baseline_ffanalyze[0]:.2f} uV, '
             f'latency: {(baseline_ffanalyze[1]*fs_hZ / 10):.2f} ms',
             xy=(baseline_ffanalyze[1] * fs_hZ / 10,
                 baseline_ffanalyze[2]),
             xytext=(baseline_ffanalyze[1] * fs_hZ / 10 + 1,
                     baseline_ffanalyze[2] - 2),
             arrowprops=dict(facecolor='darkorange', headwidth=2, shrink=0.03),
             color='darkorange')

plt.plot(t1_space, powerline_iir_filtfilt)
plt.annotate(f'ampl: {powerline_ffanalyze[0]:.2f} uV, '
             f'latency: {(powerline_ffanalyze[1]*fs_hZ / 10):.2f} ms',
             xy=(powerline_ffanalyze[1] * fs_hZ / 10,
                 powerline_ffanalyze[2]),
             xytext=(powerline_ffanalyze[1] * fs_hZ / 10 + 1,
                     powerline_ffanalyze[2] + 0.7),
             arrowprops=dict(facecolor='green', headwidth=2, shrink=0.03),
             color='green')

plt.ylabel("Signal Output in uV")
plt.xlabel("t, in ms")
plt.title("SNAP Signal and Filtered Comparison, with Phase Distortion Removed")
plt.grid()
plt.legend(['cleanSnap, removed Phase Distortion',
            'Baseline Contamination, removed Phase Distortion',
            'Powerline Contamination, removed Phase Distortion'])

plt.tight_layout()
plt.show()
```