02471 Machine Learning for Signal Processing - Fall 24

Problem set 2

Name	Student ID
Jakob Ketmig	s194264

December 8, 2024

1 Signal Processing is Linear Algebra

The goal is to construct $\mathbf{X}$ and θ such that it is possible to calculate the convolution using $\mathbf{y} = \mathbf{X}\theta$. Recall, the filtering operation is defined as:

$$y_n = \sum_{l=0}^{L-1} \theta_l x_{n-l} \tag{1}$$

To clarify, 0-indexing is assumed. The total number of rows in $\mathbf{X}$ then has to be the length of $\mathbf{y}$, which is $N + L - 1$, while the total length of θ is the length of the signal L corresponding to the columns. In other words, if we shift the discrete time signal x_n such that each row fulfills the convolution and zero-pad accordingly, then:

$$\mathbf{X}_{(N+L-1)\times L} = \begin{bmatrix} x_0 & 0 & \cdots & 0 \\ x_1 & x_0 & \cdots & 0 \\ x_2 & x_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ x_{N-1} & x_{N-2} & \ddots & x_0 \\ 0 & x_{N-1} & \ddots & x_1 \\ \vdots & 0 & \ddots & x_2 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & x_{N-1} \end{bmatrix} \quad , \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{L-1} \end{bmatrix} \tag{2}$$

2 Biased vs. unbiased parameter estimation

The Mean Square Error (MSE) is given by $MSE = \mathbb{E}[(\hat{\theta} - \theta_0)^2]$, where $\hat{\theta}$ is the estimator of the parameter while θ_0 is the true value of the parameter. Next, the bias term is defined as $\text{Bias}(\hat{\theta}) = \mathbb{E}[\hat{\theta}] - \theta_0$, while the variance is $\text{Var}(\hat{\theta}) = \mathbb{E}\left[(\hat{\theta} - \mathbb{E}[\hat{\theta}])^2\right]$. Decomposing it into the bias-variance starts off with a clever trick:

$$\begin{aligned}
\text{MSE} &= \mathbb{E}\left[(\hat{\theta} - \theta_0)^2\right] && (3)\\
&= \mathbb{E}\left[((\hat{\theta} - \mathbb{E}[\hat{\theta}]) + (\mathbb{E}[\hat{\theta}] - \theta_0))^2\right] \quad (\mathbb{E}[\hat{\theta}]'\text{s adds to } 0) && (4)\\
&= \mathbb{E}\left[\left(\hat{\theta} - \mathbb{E}\left[\hat{\theta}\right]\right)^2 + 2\left(\hat{\theta} - \mathbb{E}[\hat{\theta}]\right)\left(\mathbb{E}[\hat{\theta}] - \theta_o\right) + \left(\mathbb{E}[\hat{\theta}] - \theta_o\right)^2\right] && (5)\\
&= \mathbb{E}\left[\left(\hat{\theta} - \mathbb{E}[\hat{\theta}]\right)^2\right] + \mathbb{E}\left[2\left(\hat{\theta} - \mathbb{E}[\hat{\theta}]\right)\left(\mathbb{E}[\hat{\theta}] - \theta_0\right)\right] + \mathbb{E}\left[\left(\mathbb{E}[\hat{\theta}] - \theta_0\right)^2\right] && (6)\\
&= \mathbb{E}\left[\left(\hat{\theta} - \mathbb{E}[\hat{\theta}]\right)^2\right] + \left(\mathbb{E}[\hat{\theta}] - \theta_0\right)^2 \quad \left(\text{using } \mathbb{E}\left[\hat{\theta} - \mathbb{E}[\hat{\theta}]\right] = 0 \text{ and } (\mathbb{E}[\hat{\theta}] - \theta_0) = c \text{ and } \mathbb{E}[c] = c\right) && (7)\\
&= \text{Var}(\hat{\theta}) + \text{Bias}^2(\hat{\theta}) && (8)
\end{aligned}$$

In terms of minimizing the MSE consider an unbiased estimator, ie. $\mathbb{E}[\hat{\theta}] = \theta_0$ the contribution comes solely from the variance. If the sample variance is large enough it may indeed be worse than a biased estimator, as the variance can be significantly reduced by introducing a small, but controlled amount of bias.

In Exercise 2.2.3 we showed theoretically that under $\alpha < 0$, which effectively scales down the unbiased estimator, then the biased estimator may indeed have a lower MSE that its counterpart.

Another way to put it, I think, is generally one cares not much for a good estimate on average, but rather a good estimate in a given case-study. However, I believe it is very specific based on the case, e.g. medicine it is not preferable, whereas for my background in machine learning/forecasting, where we deal with noisy data, a small bias with lower variance is typically preferred to improve generalization performance.

3 Convexity of norms

The definition from the book of a convex function is:

$$f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2) \ , \ \ \lambda \in [0,1]$$

Let the function f be the p-norm, i.e. $f = ||\cdot||_p$, which is non-negative and defined for $\mathbb{R}^l \to [0,\infty)$, then:

$$||\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2||_p \leq ||\lambda \mathbf{x}_1||_p + ||(1-\lambda)\mathbf{x}_2||_p \quad \text{(Triangle inequality [Property 3])} \tag{9}$$

$$\Leftrightarrow$$

$$||\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2||_p \leq \lambda ||\mathbf{x}_1||_p + (1-\lambda)||\mathbf{x}_2||_p \quad \text{(Homogeneity w.r.t. RHS [Property 2])} \tag{10}$$

Hence, any function that is a norm is indeed convex.

4 Correlation functions

The expected value of y_n is constant, hence it is wide sense stationary (WSS), meaning that $r_y(k) = \mathbb{E}[y_n y_{n-k}]$. Given $y_n = x_n + b$, we have:

$$\begin{aligned}
r_y(k) = \mathbb{E}[y_n y_{n-k}] &= \mathbb{E}[(x_n + b)(x_{n-k} + b)] && (11)\\
&= \mathbb{E}[x_n x_{n-k} + b x_n + b x_{n-k} + b^2] && (12)\\
&= \mathbb{E}[x_n x_{n-k}] + \mathbb{E}[b x_n] + \mathbb{E}[b x_{n-k}] + \mathbb{E}[b^2] && (13)\\
&= r_x(k) + b^2 \qquad (\mathbb{E}[x_n] = 0 \text{ and WSS}) && (14)
\end{aligned}$$

Computing $r_y(-2)$ we use eq. 2.144 from the course book, i.e. an AR(1) process $x_n = a x_{n-1} + v_n$ is given non-recursively as $r_x(k) = \dfrac{a^{|k|}}{1-a^2}\sigma_v^2$. Applying it:

$$r_y(-2) = r_x(-2) + b^2 = \frac{0.8^{|-2|}}{1-0.8^2} \cdot 1 + 0.5^2 = 2.028 \tag{15}$$

5 Wiener filtering[1]

The desired signal is the original signal we aim to recover or estimate, in this case s_n, which is noise-free. The error signal is the difference between estimated output from the filter $\hat{s}_n$ w.r.t. the original, ie. $e_n = s_n - \hat{s}_n$. The goal is to minimize this difference, of course.

In the following analysis the raw signal is omitted. Instead, we perform a fast Fourier transform (FFT) on s_n, x_n and the filtered output signal $\hat{s}_n$ and visualize their frequency response[2]:

[1]For code parts, which include this section and the following ones, the output will be presented and references to the code can be found in the appendix.

[2]The provided plot and resulting analysis includes the final choice of filter length L.

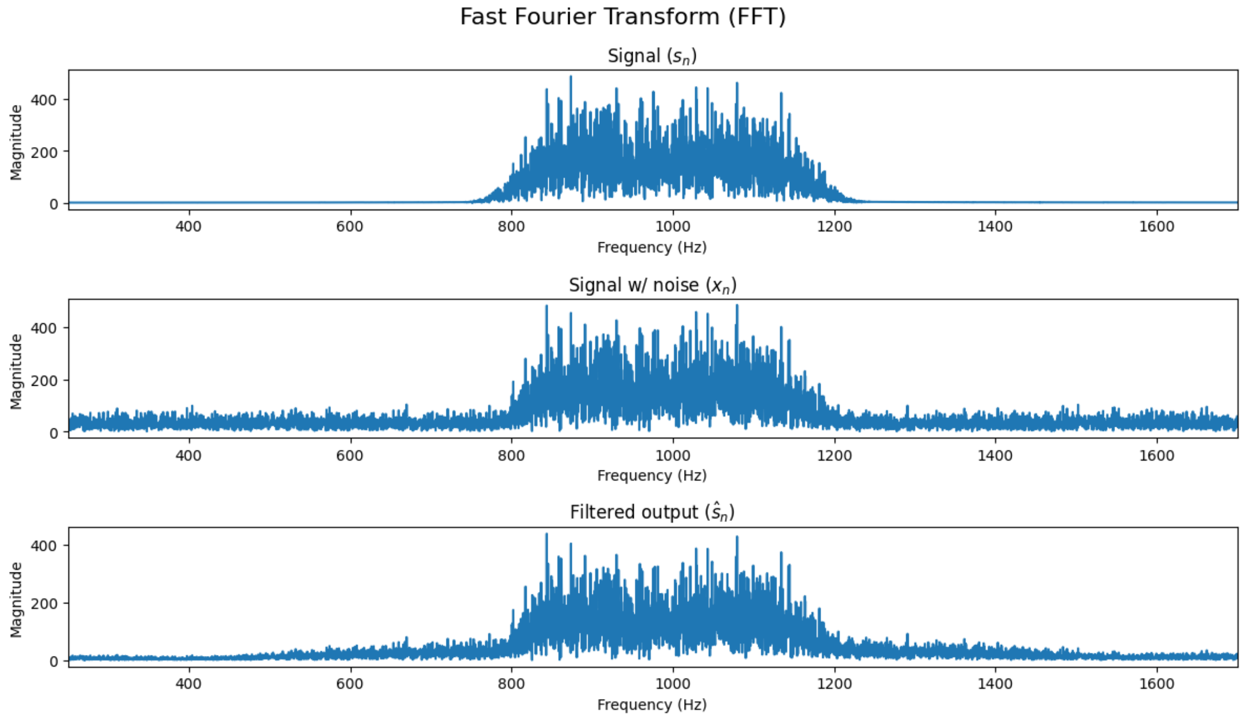
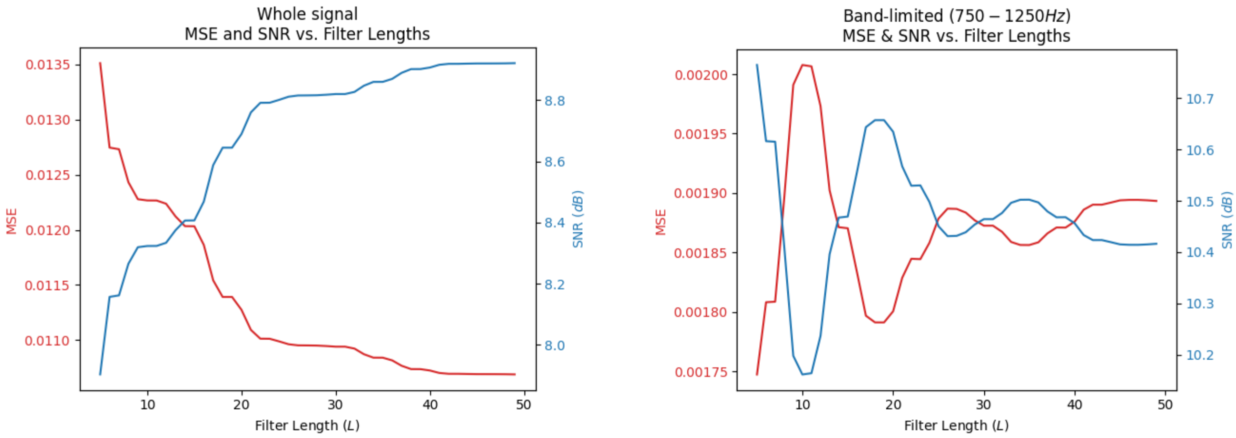


Figure 1: Frequency response of signals after FTT.

It is evident from figure 1 that the noise introduced to the signal is reduced after applying the filter, but some residual noise is left, especially in the higher and lower frequency bands outside the 750-1250 Hz range, where the primary signal resides. However, there is still some residual noise in the vicinity of the primary signal band. By tuning the filter length a better suppression of the noise around this range without significantly altering the main signal may be possible.

After rigorously inspecting the qualitative similarity between the signal and the filtered output, I decided to compute the mean-squared error (MSE) and the signal-to-noise ratio (SNR) and plot it over a range of L's, as it is very hard to find a definitive difference. However, I also decided to further specify this to the area of interest, namely the range of frequencies inhabited by the original signal, ca. 750-1250 Hz. My reasoning is that the MSE and SNR metrics when considering the full frequency range could be overemphasizing noise reduction in areas outside this area, leading to the choice of a longer filter lengths. These can be found in figure 2.



(a) Whole signal.

(b) Band-limited for signal of interest.

Figure 2: Quantitative analysis of choice of filter length, L.

Inspecting 2a believe the effect of a larger filter length L diminishes somewhere around 25-30, whereas beforehand it seems to have a noticeable gain. This is mirrored in the MSE as well.

Instead, looking at 2b indicates using L around 18 might be the most beneficial. To strike a balance between the two, I believe my final answer is I would use a filter length of $L = 20$ as I give a bit more weight to the quality of the signal. The frequency response of the filter looks like:

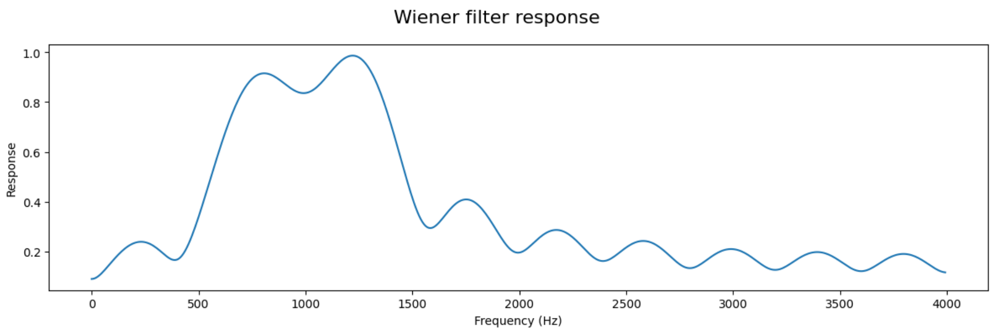


Figure 3: Frequency response of Wiener filter ($L = 20$).

Finally, I will write some general considerations I had in choosing my final filter length L. Some of them I had prior to my quantitative approach.

- A larger L led to the "dip", or also more formally referred to as the "attenuation", in the primary signal's frequency range illustrated in the filter response, see figure 3. While this is not ideal, as we try to preserve the original signal, I understand it as the Wiener filter trying to compromise between attenuating the noise at the expense of attenuating some of the signal, as well, which may result in an overall cleaner output signal. However, it provided the suppression or "cutoff" regarding frequencies outside of the primary signal band. It is also more computationally expensive, I suppose.
- Too low of an L led to a lot more residual noise inside the primary band range and outside of it, as well.

6 Time-frequency analysis of audio

The choice of parameters in my analysis come down to a trade-off between the time and frequency resolutions of the signal provided. Generally, I read online that a 50% overlap is a good starting point and my window size of 516 samples is a moderate resolution of the frequency and time localization. I chose this as there are about an average of 3 evenly spaced sounds per 2 seconds, so it should be able to capture the each sound. The trade-off for a larger window is it would decrease the time resolution and make the beat less distinct, whereas a smaller one would make it harder to distinguish the instruments on the spectrogram.

In terms of hop-size, the trade-off is the continuity of the time-frequency representation, so the goal is to not have gaps in data that could affect the analysis. If the sounds were closer to each other, ie. a fast transition, the window size would have to increase for the analysis to distinguish it as the resolution increases. This, on the other hand, would also require more computational power.

Lastly, for the window function I read online that the Hanning window is a good starting point. I tried e.g. Blackman, but I couldn't find any differences, really. That being said, the following figure 4 illustrates my resulting spectrogram alongside the signal itself.

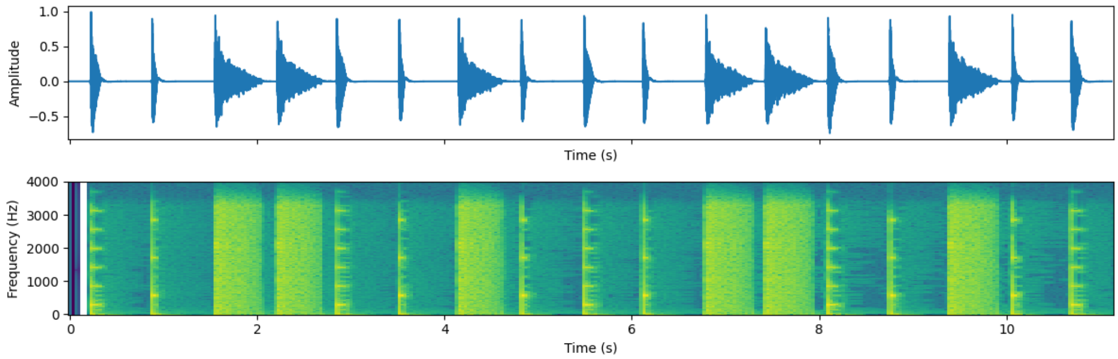


Figure 4: Spectrogram of .wav file.

There are 3 distinct sounds in the signal from the sound file. In essence, a 'boop', 'beep' and a highhat-like sound. This is also evident from the amplitudal plot, which follows the order above for the first three excitations on the plot. Upon further investigation the spectrogram plot illustrates that the sounds contain 7 frequencies for the 'boop', 3 for the 'beep', while the highhat is dispersed thoughout.

7 Adaptive filtering

7.1 Explanations

Description of the system in no particular order:

- (**h**) is the noise after it has passed through some unknown filtering, which is given by the modelling the surrounding environment, and reaches the primary microphone.
- (**d**) is the reference noise signal at the source. It acts as input to the adaptive filtering algorithm (LMS).
- (**a**) is the approximation of noise at the microphone. It is derived by learning a weighted form of (d).
- (**b**) Error signal, ie. approximation of the source after the noise has been subtracted from the primary signal. This guides the adaptive process.
- (**w**) Filtering coefficients, or weights (θ) learned by the adaptive algorithm to cancel out the noise.
- (**c**) Update of the weights based on error and input-term combinations.

The principle of the system is to stop the update of the weights when our error-term is close enough to the source signal. Here, the adaptive algorithms rely on the error term being uncorrelated with the source signal's input noise, otherwise this would result in filtering out the signal itself, instead.

7.2 Filtering exercise

Loading in the file and applying the LMS, NLMS and RLS algorithms and evaluating their MSE based on the filter of my choice, which is the band-pass BPIR filter. Here I have used the following parameters $\mu_{LMS} = 0.09$, $\mu_{NLMS} = 0.1$, $\delta_{NLMS} = 0.001$, $\beta_{RLS} = 0.995$ and $\lambda_{RLS} = 0.001$, which I found by tweaking them slightly after following the slides' general guidelines.

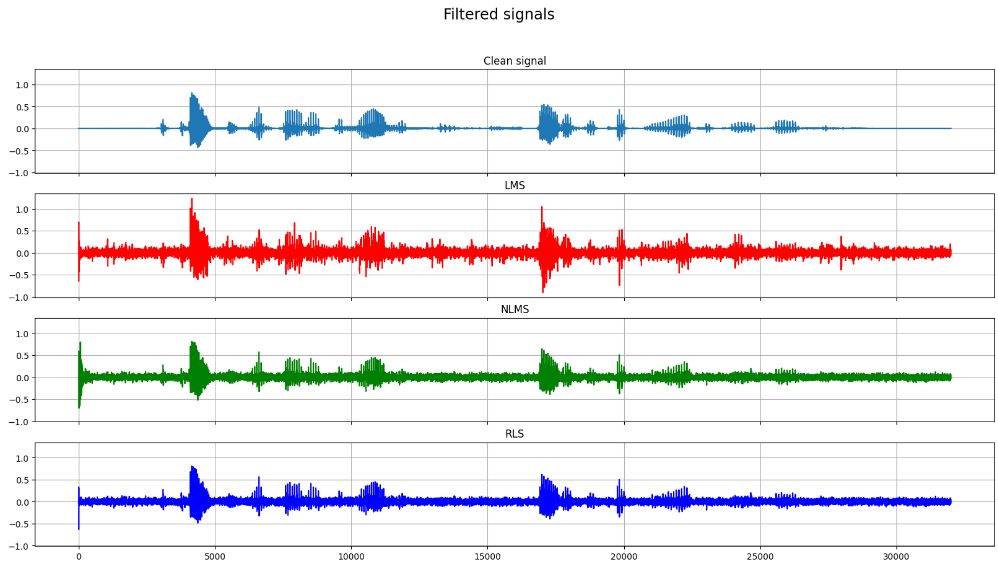


Figure 5: Filtered signal using LMS, NLMS and RLS, respectively.

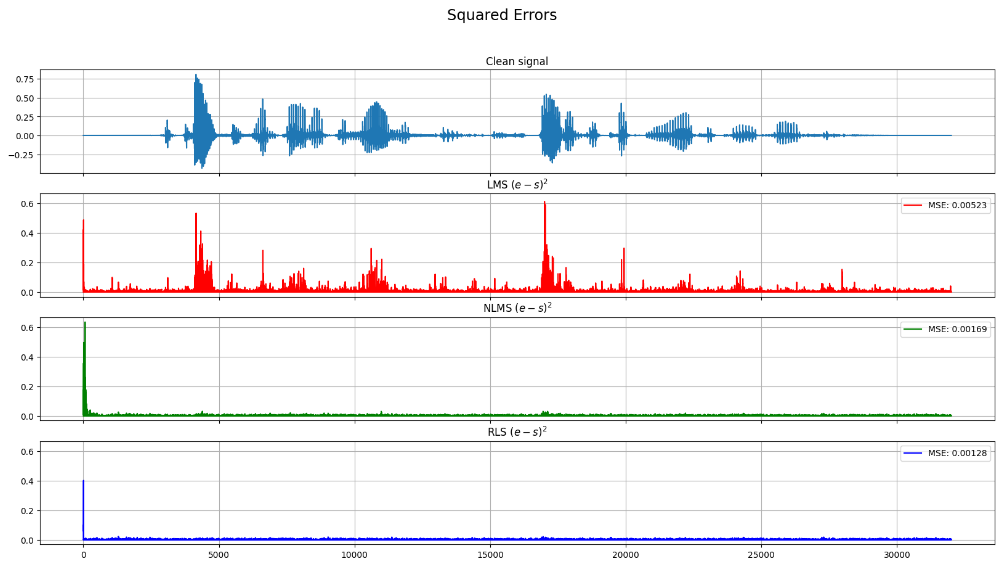


Figure 6: Mean Squared Error of filters on the signal.

Evidently, the MSE of the RLS is the lowest, see figure 6. NLMS is definitely better than LMS. Especially the start, as mentioned in the lecture, contributes to it, but for the LMS it's also generally noticable from figure 5 that it does not compete with the two other algorithms at all.

8 Appendix

8.1 Code

8.1.1 Wiener Filtering

```python
from scipy.io import loadmat
from xcorr import xcorr
from scipy.linalg import toeplitz
from scipy import signal
import numpy as np
import matplotlib.pyplot as plt
s_n = loadmat(f'data/problem2_5_signal.mat')['signal'].flatten()
w_n = loadmat(f'data/problem2_5_noise.mat')['noise'].flatten()

x_n = s_n + w_n

L = 20 # Filter length
Fs = 8000 # Sampling frequency (stated in problem)
L_range = range(5,50)

mse_vals = []; snr_vals = []

def calculate_snr(signal, noisy_signal):
    signal_power = np.sum(signal ** 2)
    noise_power = np.sum((signal - noisy_signal) ** 2)
    return 10 * np.log10(signal_power / noise_power)

# Compute Wiener filter
r_xx = xcorr(x_n, x_n, L-1)
R_xx = toeplitz(r_xx[L-1:])
r_dx = xcorr(s_n, x_n, L-1)
# the filter
theta = np.linalg.solve(R_xx, r_dx[L-1:])

# Filter noisy signal
s_hat_n = signal.lfilter(theta, 1, x_n)


fig, axes = plt.subplots(3, 1, figsize=(12, 7))
fig.suptitle('Fast Fourier Transform (FFT)', fontsize=16)
for i, sig in enumerate([(s_n, r"Signal ($s_n$)"), (x_n, r"Signal w/ noise ($x_n$)"), (s_hat_n, r
    "Filtered output ($\hat{s}_n$)")]):
    T = len(sig[0])  # signal duration
    t = np.arange(0, len(sig[0])) #T, 1/Fs)  # time vector
    X = np.fft.rfft(sig[0])
    f = np.fft.rfftfreq(len(sig[0]), 1/Fs)
    axes[i].plot(f, abs(X))
    axes[i].set_title(sig[1])
    axes[i].set_xlabel('Frequency (Hz)')
    axes[i].set_ylabel('Magnitude')
    axes[i].set_xlim(250, 1700)
fig.tight_layout()
plt.show()


##################
## WHOLE SIGNAL ##
##################

for k in L_range:
    # Compute Wiener filter
    r_xx = xcorr(x_n, x_n, k-1)
    R_xx = toeplitz(r_xx[k-1:])
    r_dx = xcorr(s_n, x_n, k-1)
    theta = np.linalg.solve(R_xx, r_dx[k-1:])

    s_hat_n = signal.lfilter(theta, 1, x_n)
```

```
    mse = np.mean((s_n - s_hat_n) ** 2)
    mse_vals.append(mse)

    snr = calculate_snr(s_n, s_hat_n)
    snr_vals.append(snr)

# Plotting...
fig, ax1 = plt.subplots()

color = 'tab:red'
ax1.set_xlabel('Filter Length ($L$)')
ax1.set_ylabel('MSE', color=color)
ax1.plot(L_range, mse_vals, color=color)
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()
color = 'tab:blue'
ax2.set_ylabel('SNR ($dB$)', color=color)
ax2.plot(L_range, snr_vals, color=color)
ax2.tick_params(axis='y', labelcolor=color)

plt.title('Whole signal\nMSE and SNR vs. Filter Lengths')
fig.tight_layout()
plt.show()


##################
## BAND LIMITED ##
##################
from scipy.fft import fft, ifft, fftfreq

band_start = 750  # Lower bound of the frequency range of interest
band_stop = 1250  # Upper bound of the frequency range of interest
mse_vals = []; snr_vals = []

# helper
def calculate_weighted_mse_snr(s_n, s_hat_n, Fs, band_start, band_stop):
    # FFT of original and filtered signals
    S = fft(s_n)
    D = fft(s_hat_n)

    freqs = fftfreq(len(s_n), 1/Fs) # all frequencies

    # Masking for range of interest
    band_mask = (freqs >= band_start) & (freqs <= band_stop)
    S_band = S[band_mask]
    D_band = D[band_mask]

    # Reconstructing the signal
    s_n_band = np.real(ifft(S_band, n=len(s_n)))
    s_hat_n_band = np.real(ifft(D_band, n=len(s_hat_n)))

    # MSE
    mse_band = np.mean((s_n_band - s_hat_n_band) ** 2)

    # SNR
    snr_band = calculate_snr(s_n_band, s_hat_n_band)

    return mse_band, snr_band

# weighted loop
for k in L_range:
    # Compute Wiener filter
    r_xx = xcorr(x_n, x_n, k-1)
    R_xx = toeplitz(r_xx[k-1:])
    r_dx = xcorr(s_n, x_n, k-1)
    theta = np.linalg.solve(R_xx, r_dx[k-1:])
```

```

    # Apply filter to noisy signal
    s_hat_n = signal.lfilter(theta, 1, x_n)

    # Calculate weighted MSE and SNR in the primary frequency band
    mse_band, snr_band = calculate_weighted_mse_snr(s_n, s_hat_n, Fs, band_start, band_stop)

    mse_vals.append(mse_band)
    snr_vals.append(snr_band)

# Plotting...
fig, ax1 = plt.subplots()

color = 'tab:red'
ax1.set_xlabel('Filter Length ($L$)')
ax1.set_ylabel('MSE', color=color)
ax1.plot(L_range, mse_vals, color=color)
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()
color = 'tab:blue'
ax2.set_ylabel('SNR ($dB$)', color=color)
ax2.plot(L_range, snr_vals, color=color)
ax2.tick_params(axis='y', labelcolor=color)

plt.title('Band-limited ($750-1250Hz$)\nMSE & SNR vs. Filter Lengths')
fig.tight_layout()
plt.show()
```

8.2 Time-frequency analysis of audio

```
from scipy import signal
import scipy
import numpy as np
import librosa
import librosa.display # requires np <= 2.0
import IPython
from scipy.signal import stft
import matplotlib.pyplot as plt

synth, Fs = librosa.load('data/problem2_6.wav')
# Player for the original signal
IPython.display.Audio('data/problem2_6.wav')

# STFT params
frame_size = 516
lag = frame_size / 2 # 50% overlap
window = 'hanning'
# Plotting ...
fig, axes = plt.subplots(2, 1, figsize=(12, 4), sharex=True)
t = np.arange(len(synth))/Fs
axes[0].plot(t, synth)
axes[0].set_xlabel('Time (s)')
axes[0].set_ylabel('Amplitude')
overlap_size = frame_size - lag
f, t, S = scipy.signal.stft(synth, fs=Fs, nperseg=frame_size, noverlap=overlap_size)
S_dB = 20*np.log10(abs(S))
axes[1].pcolormesh(t, f, S_dB, shading='auto')
axes[1].set_xlabel('Time (s)')
axes[1].set_ylabel('Frequency (Hz)')
fig.tight_layout()
plt.show()
```

8.3 Adaptive filtering

```python
from scipy.io import loadmat
from convmtx import convmtx
from lms import lms
from nlms import nlms
from rls import rls

Fs = 8000 # Specified sampling freq
# Signal (Speech)
s = loadmat(f'data/problem2_7_speech.mat')['speech'].flatten()
N = len(s)
noiselevel = 1.0 # N(0,1) noise
n = np.random.randn(N) * noiselevel # 2.1.5
filters = [loadmat(f'data/problem2_7_'+_filter+'.mat')[_filter].flatten() for _filter in ['lpir',
    'hpir', 'bpir']]

# Filtering out noise (ie. (h))
h = signal.lfilter(filters[2], 1, n).flatten()
# Adding filtered noise to signal
pm = s + h

L = len(filters[0]) # dimension of the unknown vector
N = len(pm)         # number of data samples
# LMS params
mu_lms = 0.09       # step-size
# NLMS params
mu_nlms = 0.1       # step-size ( 0 < mu < 2)
delta = 1e-3        # regularization (small)
# RLS params
beta = 0.995        # forget factor
lambda_ = 1e-3      # regularization
# Execute
_, e_lms = lms(n, pm, L, mu_lms)
_, e_nlms = nlms(n, pm, L, mu_nlms, delta)
_, e_rls = rls(n, pm, L, beta, lambda_)

# Filtered signals
fig, axes = plt.subplots(4, 1, figsize=(20, 10), sharex=True, sharey=True)
colors = [None, 'r', 'g', 'b']
for i, method in enumerate([(s, 'Clean signal'), (e_lms, 'LMS'), (e_nlms, 'NLMS'), (e_rls, 'RLS')
    ]):
    axes[i].plot(method[0], color = colors[i], label = method[1])
    axes[i].grid('on')
    axes[i].set_title(method[1])
plt.suptitle('Filtered signals', fontsize = 'xx-large')
plt.show()


# Squared errors
SE_rls = (e_rls - s)**2
SE_lms = (e_lms - s)**2
SE_nlms = (e_nlms - s)**2
fig, axes = plt.subplots(4, 1, figsize=(20, 10), sharex=True)
for i, method in enumerate([(s, 'Clean signal'), (SE_lms, r'LMS $(e - s)^2$'), (SE_nlms, r'NLMS $
    (e - s)^2$'), (SE_rls, r'RLS $(e - s)^2$')]):
    axes[i].plot(method[0], color = colors[i], label = 'MSE: '+str(np.mean(method[0]))[:7])
    axes[i].grid('on')
    axes[i].set_title(method[1])
    if i != 0:
        axes[i].sharey(axes[1])
        axes[i].legend()
plt.suptitle('Squared Errors', fontsize = 'xx-large')
plt.show()
```

References