BE1 _ Optimal Linear Filtering _ TRAN Gia Quoc Bao, ASI 2nd year

Table of Contents

Introduction:

In this BE we will apply and see the effects of optimal linear filtering to filtering broadband signals with broadband noise. We will use the Wiener with an attempt to improve its adaptivity and use the power spectral density (PSD) as a criterion to verify the performance.

Default commands

```
close all;
clear all;
clc;
```

I. The EMG signal

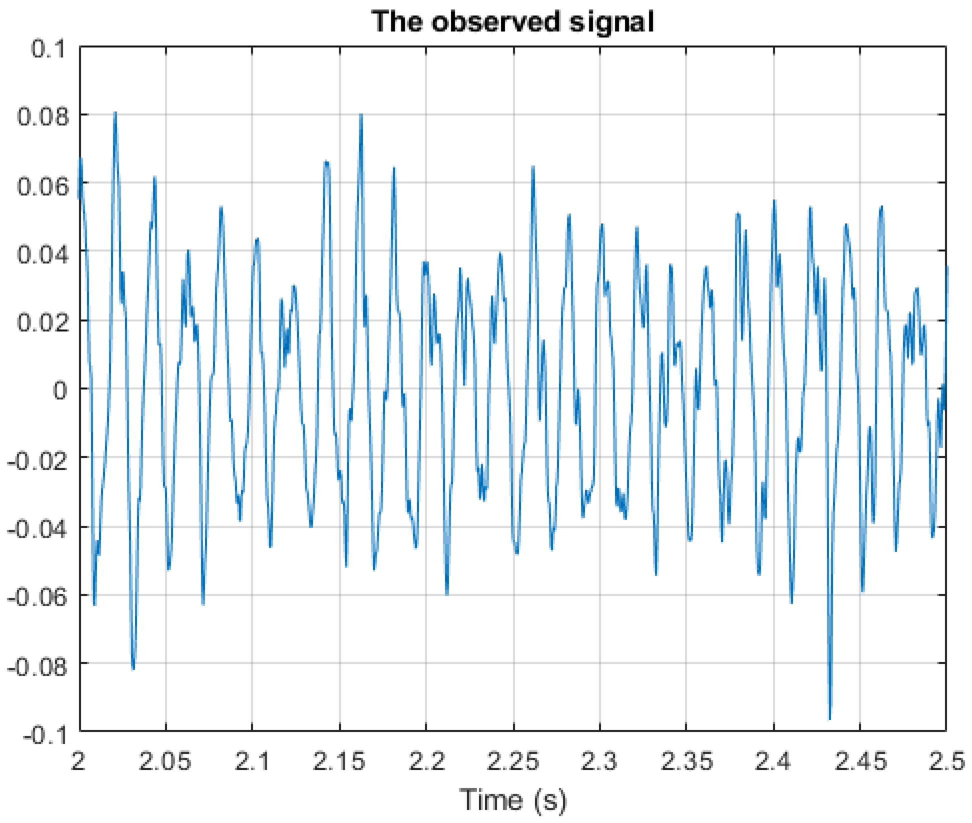
Load & prepare the signal

```
load('demipronation_extension1_lead3.mat') ;
sig_y = emg_data - mean(emg_data) ; % detrend the signal, as we
 hypothesise it to be centered.
clear emg_data; % to relieve the memory
Fs = 2000; % sampling period in Hz
% Fs = 2025; % to test later the difference of 2 filters
N = length(sig_y); % get the size
time = [(0 : N - 1)/Fs]'; % the discrete time
```

Visualize the signal

```
figure('Name', 'Signal in time domain');
```

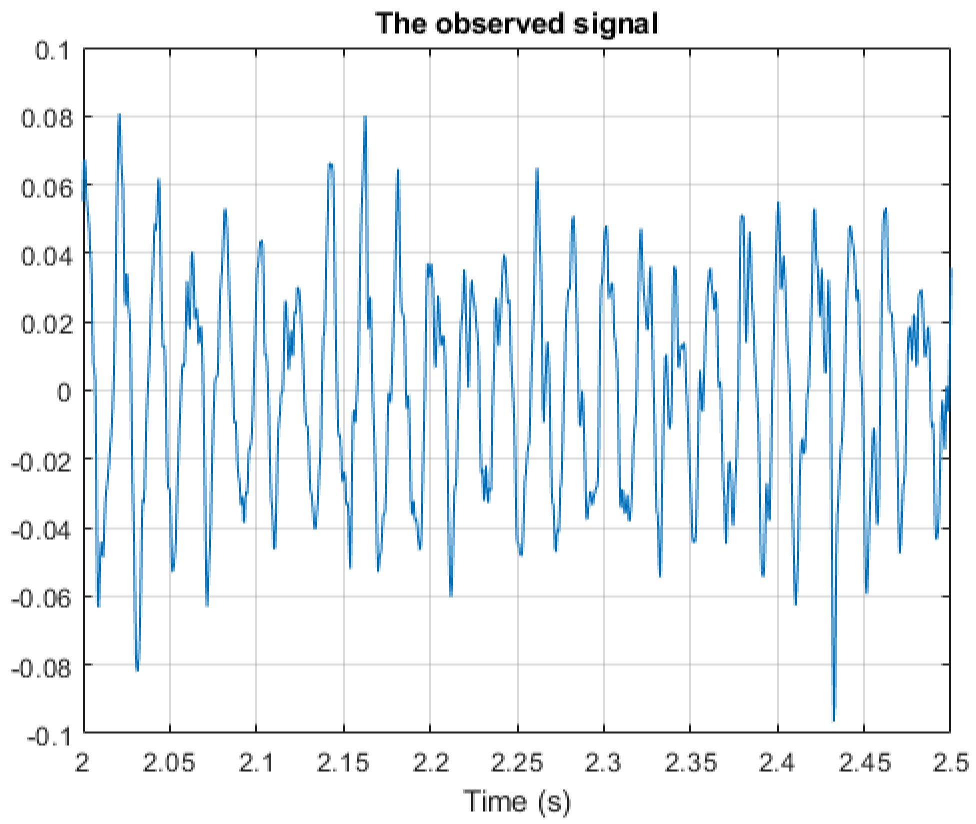
```
plot(time, sig_y);
grid on;
xlim([2 2.5]); % plot in a certain zone
xlabel('Time (s)');
title('The observed signal');

% The detrended signal has a mean of 0. It looks periodic so in the
 next
% step we would like to detect the power contained in frequencies
 using power
% spectral density (PSD).
```

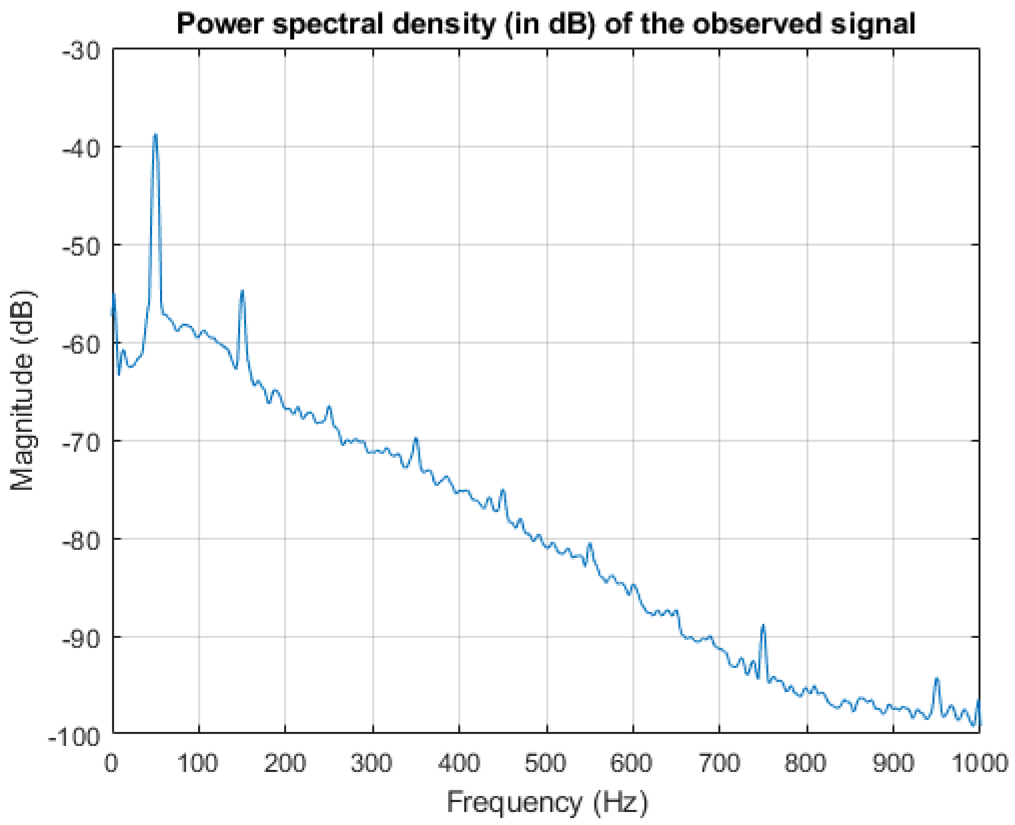


Spectral density analysis

```
Nblocks = 512; % size of the blocks for which the FFT is calculated
rec = round(3*Nblocks/4) ; % 75pc overlap between blocks

% We estimate the spectrum on Nblocks + 1 points, i.e. with a spectral
% resolution of Fs / (2 * Nblocks)
[psdy freq] = pwelch(sig_y, hanning(Nblocks), rec, 2*Nblocks, Fs);

figure('Name', 'Signal in spectral domain');
plot(freq, 10*log10(psdy)); % magnitude in decibel
grid on;
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
```

```
title('Power spectral density (in dB) of the observed signal');

% The PSD is calculated up to half the sampling frequency.

% We see some peaks at 50 Hz and its odd harmonics 150 250,....
% These are added because the electrical network has a frequency of 50
 Hz.

% We need to cancel this periodic noise without losing (much) useful
% information whose frequencies range from 20 - 400 Hz. To do this we
% cannot use traditional filters (which have certain cutoff
 frequencies)
% but the optimal linear filters. In the next parts we'll consider the
% Wiener filter.
```





II. Periodic noise estimation / suppression

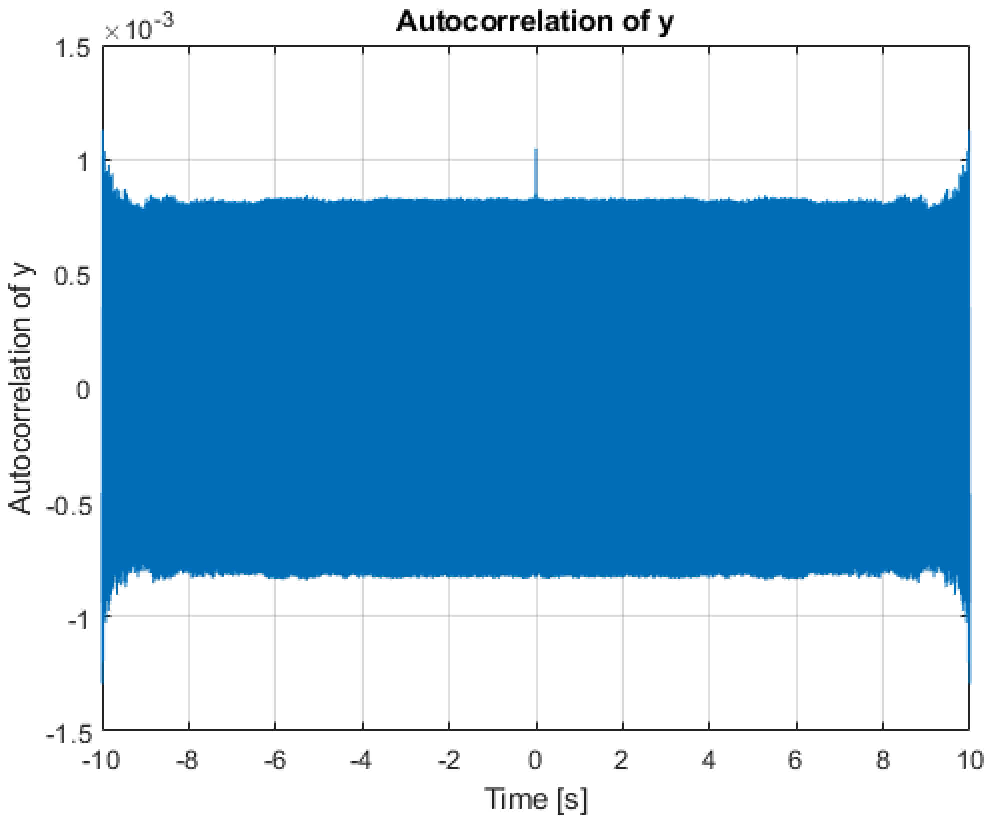
Finite Impulse Response (FIR) Wiener Filter

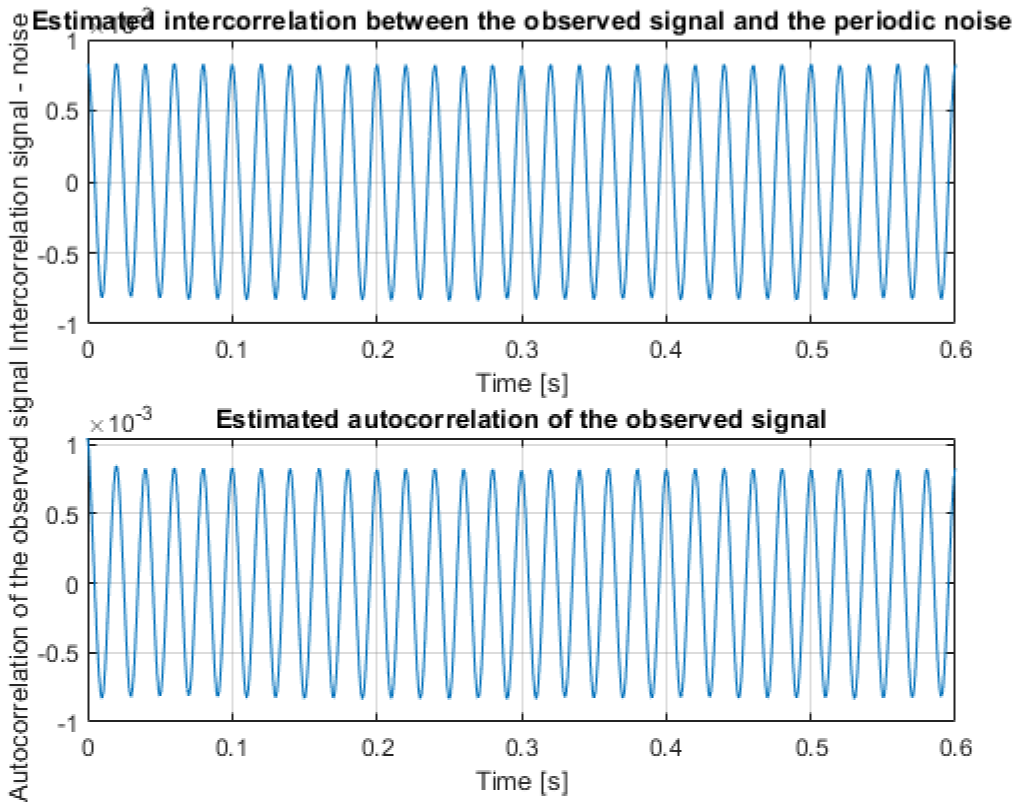
Estimate the filter's order: If we zoom the PSD, we can see that the width (in Hz) of the peaks is approximately 10 Hz (5 for each side). The peaks with lower freq have slightly wider ranges but not so different. From the relation between the band and the filter's order: 3.1/N < 0.5*(deltaF/Fs) = 0.5*(10/2000) = 1/400. So we can take:

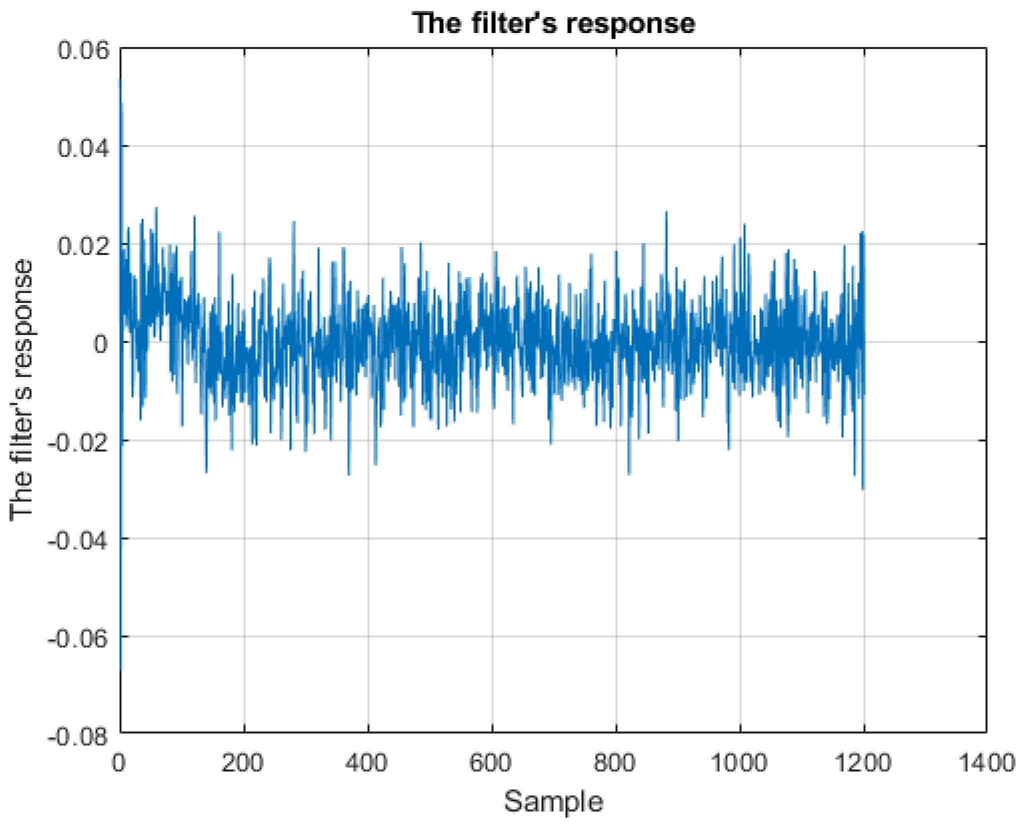
```
Nrif = 1200;

[cy, lags] = xcorr(sig_y, 'unbiased');

figure('Name', 'Autocorrelation of y');
plot(lags/Fs, cy);
grid on;
xlabel('Time [s]');
ylabel('Autocorrelation of y');
title('Autocorrelation of y');

% For the options 'biased', 'normalized' and 'coeff' the result has an
% envelope form. For the last 2, the autocorrelation at 0 lag is
 normalized
% to be 1. For the 'unbiased' option the result looks like a
 rectangle. The
```

```
% shape depends on the normalization factor (1 for unbiased).
% When calculating autocorr, the data y we have is noisy, and for
 large
% delays the estimator can become unstable. We can see that for
 'unbiased',
% at the 2 sides of the result it starts to diverge (clearly from 9 to
 10).
% But if we want to see more clearly the good value for the delay it
 is better
% to go with unbiased.

% With the necessary assumptions about x(t), vp(t), we can build the
 filter.
% To do this we need gamma_y (auto) and gamma_sy (inter).

% The 1st one is known from y(t). For the 2nd one, we cannot find it
% directly but we replace it by gamma_yyr under the assumptions that
 yr is
% obtained by delaying y by a large enough number of samples.

% By zooming in the autocorr, we see that it is safe to delay it at
 0.04 (the amplitude
% becomes constant from there) so in number of samples:
M = 0.04*Fs;
y = sig_y(M + 1 : end) ;     % y now designates the observed signal
L = length(y) ;          % signal size observed (in number of samples)
time = time(M + 1 : end) ; % time domain associated with y (in
 seconds)
yr = sig_y(1 : L) ;   % signal y delayed by M samples
clear sig_y; % now we only work with y and yr, so we sacrificed some
 information in y

% To build matrix gamma_sy we do as said earlier:
[inter_sy, lags] = xcorr(y, yr, Nrif, 'unbiased');
inter_sy = inter_sy((Nrif + 1) : end);
lags = lags((Nrif + 1) : end)/Fs;  % in seconds
% To build matrix gamma_y we find the autocorrelation of y
auto_y = xcorr(y, y, Nrif, 'unbiased');
auto_y = auto_y((Nrif + 1) : end);

% Plot the results
figure('Name', 'Correlations');
subplot(211);
plot(lags, inter_sy);
grid on;
xlabel('Time [s]');
ylabel('Intercorrelation signal - noise');
title('Estimated intercorrelation between the observed signal and the
 periodic noise');
subplot(212);
plot(lags, auto_y);
grid on;
xlabel('Time [s]');
ylabel('Autocorrelation of the observed signal');
```

```
title('Estimated autocorrelation of the observed signal');

% The filter's response is found by solving the Wiener-Hopf equation:
 gamma_sy = gamma_y*w
gamma_y = toeplitz(auto_y);
gamma_sy = inter_sy;
w = (inv(gamma_y))*gamma_sy;

% Plot the results
figure('Name', 'Response');
plot(w);
grid on;
xlabel('Sample');
ylabel("The filter's response");
title("The filter's response");

% From here we can filter the signal
vpEst = filter(w, 1, y);
% And then subtract the estimated noise from the signal for the
 estimated x
xEst = y - vpEst;
```




Estimated intercorrelation between the observed signal and the periodic noise
$\times 10^{-3}$
Intercorrelation signal - noise
Time [s]
Estimated autocorrelation of the observed signal
$\times 10^{-3}$
Autocorrelation of the observed signal
Time [s]

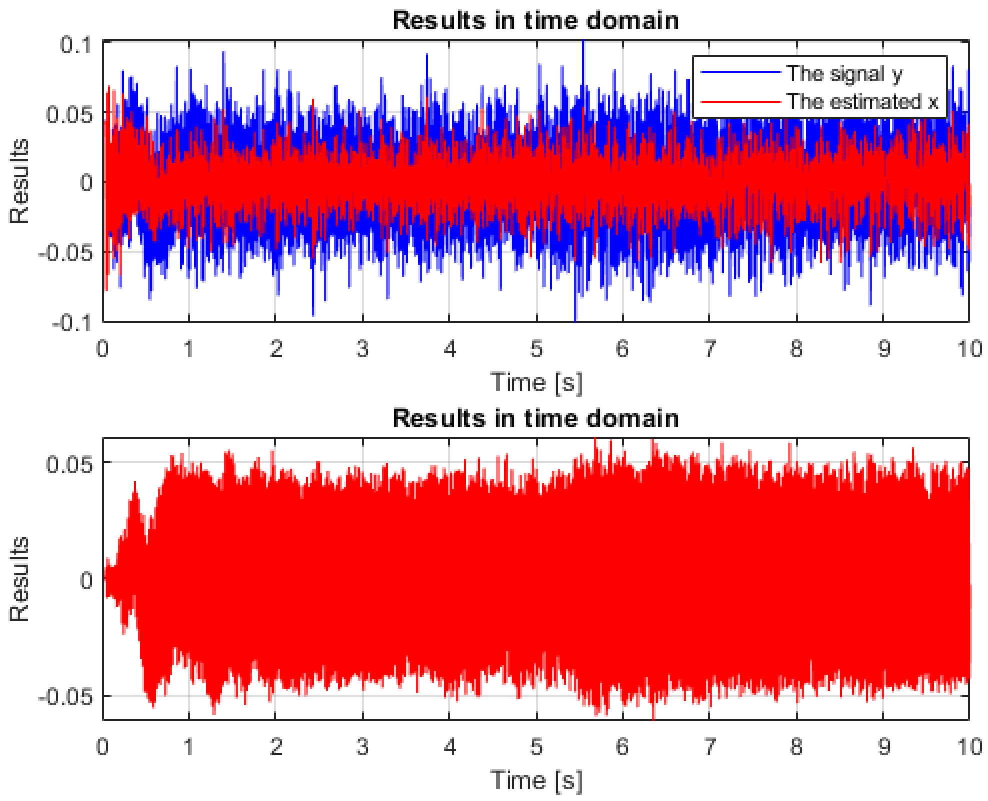

The filter's response
The filter's response
Sample

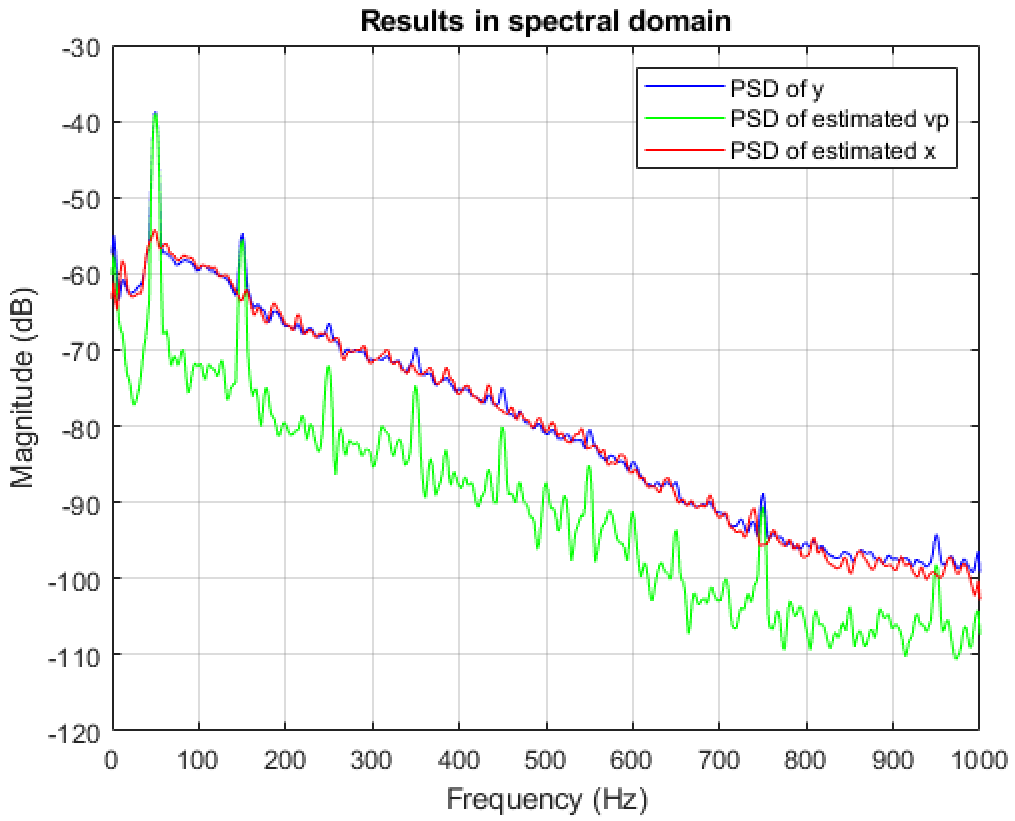
Plot the results

First we can plot the results in time domain

```
figure('Name', 'Results in  time domain');
subplot(211);
plot(time, y, 'b', time, xEst, 'r');
grid on;
xlabel('Time [s]');
ylabel('Results');
legend('The signal y', 'The estimated x');
title('Results in time domain');
subplot(212);
plot(time, vpEst, 'r');
grid on;
xlabel('Time [s]');
ylabel('Results');
title('Results in time domain');

% Not very easy to see... If we zoom in we'll see that vpEst has a
 frequency
% of almost 50 Hz, which is good (I inserted the zoomed image below).
% Let us examine them in the spectral domain.

[psdy freq] = pwelch(y, hanning(Nblocks), rec, 2*Nblocks, Fs);
[psdvpEst freq] = pwelch(vpEst, hanning(Nblocks), rec, 2*Nblocks, Fs);
[psdxEst freq] = pwelch(xEst, hanning(Nblocks), rec, 2*Nblocks, Fs);

figure('Name', 'Results in spectral domain');
plot(freq, 10*log10(psdy), 'b', freq, 10*log10(psdvpEst), 'g', freq,
 10*log10(psdxEst), 'r'); % magnitude in decibel
grid on;
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
legend('PSD of y', 'PSD of estimated vp', 'PSD of estimated x');
title('Results in spectral domain');

% Analysis: The signal is now smoothed and can be used to study. The
% estimated noise has clear peaks at 50 150 ... and so by subtracting
 this
% the x can be estimated quite well.

% About the effects of Nrif and M:

% Nrif is the filter's order. The bigger Nrif is, the more filtered
 the
% signal, but the more coefficients we must calculate. The number of
% coefficients to calculate is (N + 1)^2 + (N + 1) = N^2 + 3*N + 2.
% So we need to balance between the effectiveness and the cost. It is
% not worth increasing N if the performance is not improved
 considerably.
% After some experiments and verfication with PSD, I found out it is
 fine
```

```
% to use Nrif = 1200. Smaller values are of course possible.

% M is the number of delayed samples. The bigger M is, the more
 accurate
% the filter, because in the approximation M is >> 0. But a too big
 value
% for M will make us lose information contained in the signal. So
 again
% this is a tradeoff and we can use the PSD as a criterion for
 evauation. M
% just needs to be big enough for the PSD to be almost constant. After
 some
% experiments and verfication with PSD, I found out it is fine for M
 to be
% 80, maybe a bit smaller.
```



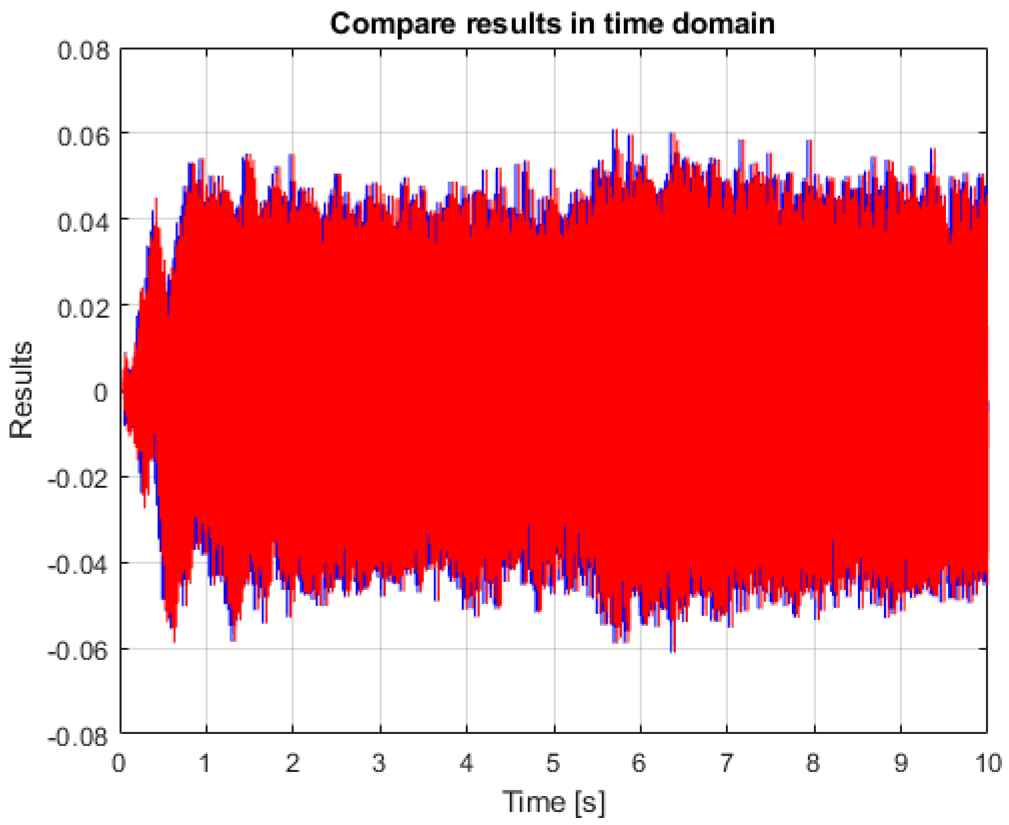


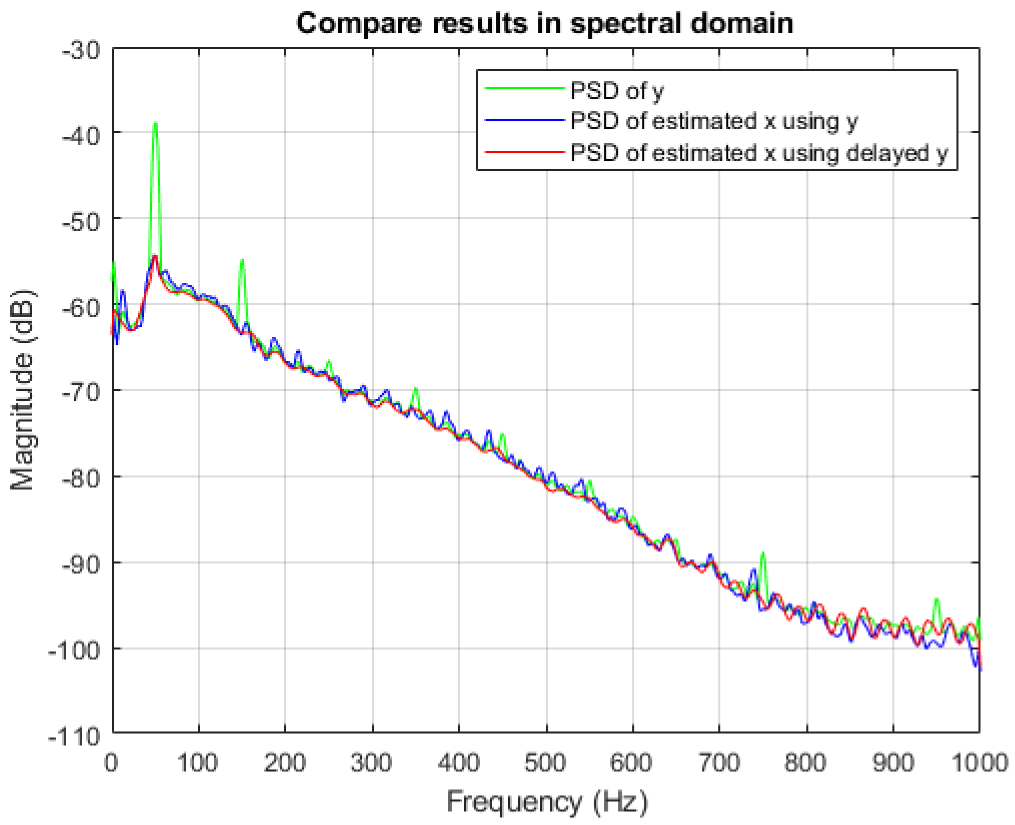
III. Prediction / suppression of periodic noise from the delayed signal

The Wiener filter for estimating (and then removing) periodic noise can be reinterpreted as a problem of predicting periodic noise from the delayed signal.

```
% We do not need to reconstruct the filter because as proved,
 gamma_sy' is
% almost gamma_yyr and gamma_y' = gamma_y

% Let us filter the delayed signal with the same filter.
vpEst2 = filter(w, 1, yr);
xEst2 = y - vpEst2;

% Compare with the last case:
figure('Name', 'Compare results in time domain');
plot(time, vpEst, 'b', time, vpEst2, 'r');
grid on;
xlabel('Time [s]');
ylabel('Results');
title('Compare results in time domain');
% Not very different. We still need the PSD to verify.

[psdxEst2 freq] = pwelch(xEst2, hanning(Nblocks), rec, 2*Nblocks, Fs);
```

```
figure('Name', 'Results in spectral domain');
plot(freq, 10*log10(psdy), 'g', freq, 10*log10(psdxEst), 'b', freq,
 10*log10(psdxEst2), 'r'); % magnitude in decibel
grid on;
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
legend('PSD of y', 'PSD of estimated x using y', 'PSD of estimated x
 using delayed y');
title('Compare results in spectral domain');

% For now, the 2nd case gave better results in the high frequency
 domain.
% But in the useful range the 2 filters gave almost the same results.
 The
% 2nd filter is still a better one. Maybe in some other cases we'll
 need the
% high frequencies.

% Here the delay does not need to be in multiple. I still let it take
 the
% same value but this is not required. This can give the advantage
 when the
% sampling frequency which we use to calculate the PSD is not
 divisible by
% the noise frequency. Because if the delay is required to be
 multiple, it
% has to lie on the noise's peaks to filter them. The 2nd filter does
 not
% require this: it still works well even if the Fs is not a multiple
 of the
% noise's frequency. The Fs is not always flexible but it is by
 default. If
% unfortunately the noise's frequency is 30 Hz then the 2nd would work
 much
% better.
% Here (the noise's freq is already given) I experimented by changing
 the Fs
% into 2025 Hz which would give the furthest remainder which is 25. I
% inserted the results below. It can be seen that the 2nd filter (red)
% works fine with all noise frequencies while the 1st one (blue) did a
 bad job.
```


Compare results in time domain
Results
Time [s]
0.08
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
-0.08
0
1
2
3
4
5
6
7
8
9
10


Compare results in spectral domain
PSD of y
PSD of estimated x using y
PSD of estimated x using delayed y
Magnitude (dB)
Frequency (Hz)
-30
-40
-50
-60
-70
-80
-90
-100
-110
0
100
200
300
400
500
600
700
800
900
1000

Conclusion:

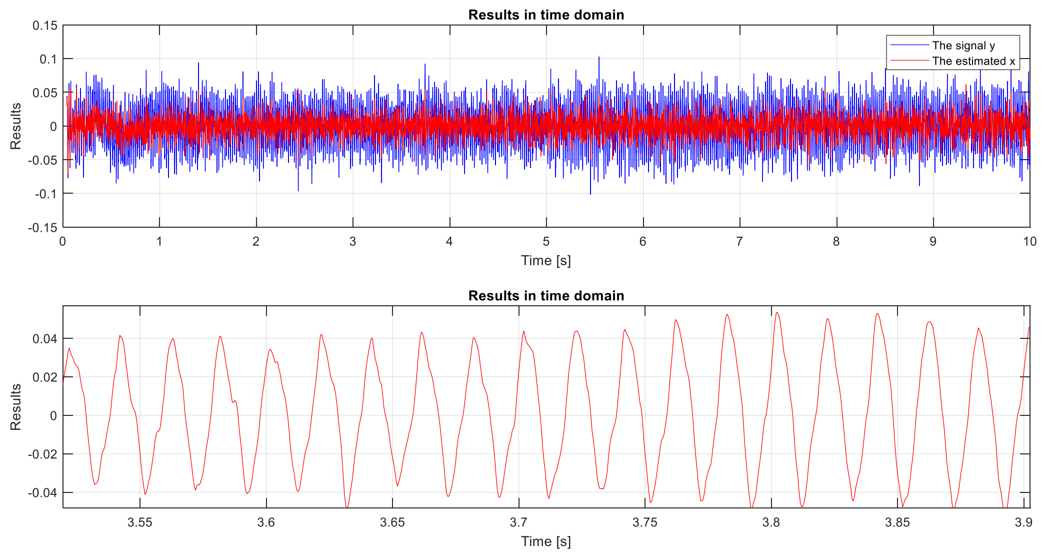
After this lab work I understood better the application of optimal linear filtering. The 2nd structure of filtering was proven to be better. The procedure to follow is first to find the PSD to identify the filter's order Nrif. Then perform autocorrelation for the delay M. Then construct the filter with the 2nd model and then verify the results with PSD.

```
% In the last page I inserted some extra images.
```

Published with MATLAB® R2019a

Some extra figures…

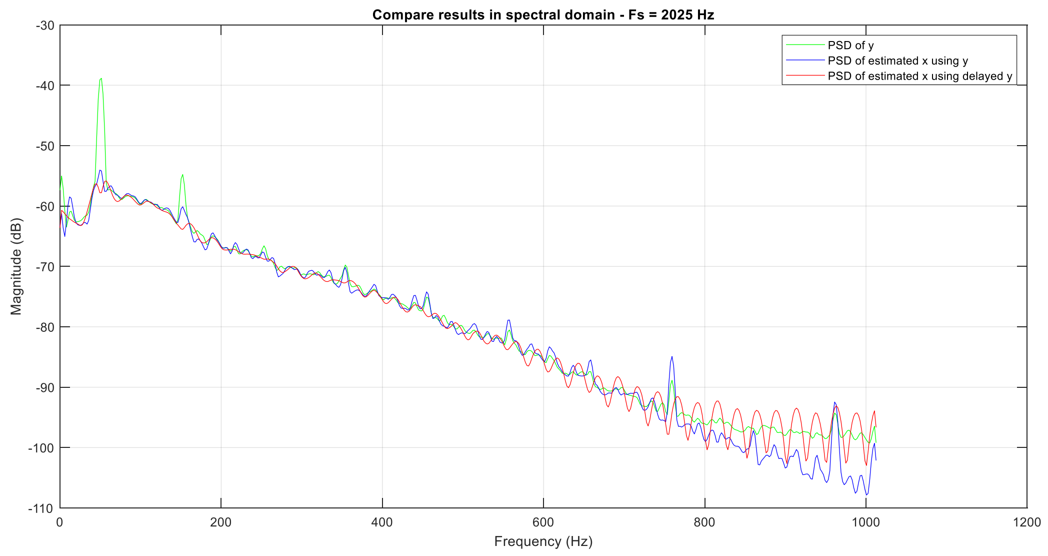
1/ Verification of estimated vp:



I zoomed the results. We see that from 3.6 to 3.8s there are 10 cycles so the period is (3.8-3.6)/10 = 0.02s and so the frequency of the estimated noise is 1/0.02 = 50 Hz. This filter works well.

2/ Experiment when Fs not divisible by noise frequency.

Here is the case Fs is not divisible by noise's frequency. The 2nd filter still does well but the 1st one could not cut the right frequencies. So the 2nd one is more adaptive to the noise frequency.


Compare results in spectral domain - Fs = 2025 Hz
PSD of y
PSD of estimated x using y
PSD of estimated x using delayed y
Magnitude (dB)
Frequency (Hz)