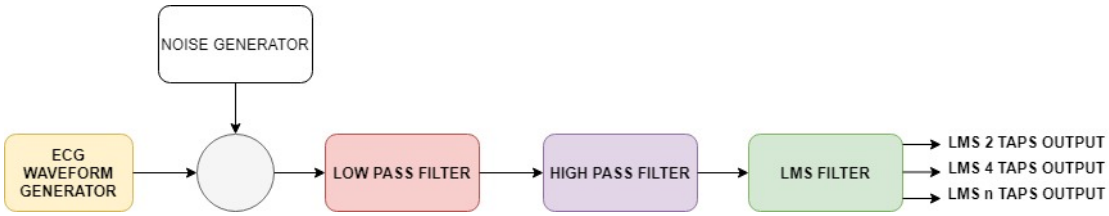
Table of Contents

Filter 50/60 Hz signal from ECG simulated data

This script demonstrate how to filter a noisy ECG signal. Over the simulated ECG signal is added baseline noise, white noise and power line hum. The filtering method is shown bellow:



There are proposed three filtering methods: using a LMS filter with 2 taps, with 4 taps and a filter with N taps. This raport was generated using Matlab publish function.

```
clear
close all
```

User controlled variables

This section includes all the user variables.

```
LPF_cutoff = 25; %ECG signal band => between 10 Hz and 25 Hz
HPF_cutoff = 1; % Baseline wander frequency is lower than 1Hz. Can be
 higher in special conditions (running)
LMS_conv = 0.009; % LMS convergence variable
```

Baseline noise is the short time variation of the baseline from a straight line caused by electric signal fluctuations, lamp instability, temperature fluctuations and other factors.

White noise is a random signal having equal intensity at different frequencies, giving it a constant power spectral density.

An oscillating voltage in a conductor generates a magnetic field that will induce a small oscillating voltage in nearby conductors, and this is how **electrical noise** in the environment shows up in the EEG. AC line current consists of sinusoidal oscillations at either 50 Hz or 60 Hz

Noise amplitudes for white noise, power line hum noise and baseline noise are defined bellow:

```
Noise_amplitude = 0.020;
Mains_interference_amplitude = 8;
Baseline_wander_amplitude = 1;


f_baseline = 0.2; % baseline noise frequency
f_interference = 50; % power line frequency
```

The sampling rate has to be wisely chosen in order to obtain an exactly 90 degree phase shift for the reference signal.

Example

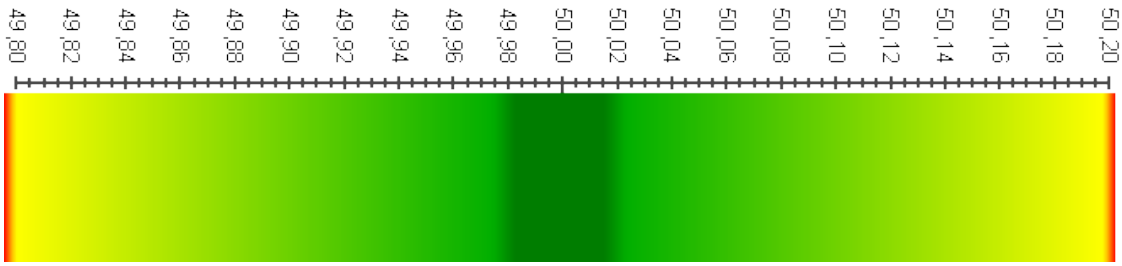
Fs=50*16 = 800 hz = 1.25 ms 50 hz -> 20ms => 20/1.25 = 16 samples/cycle => 16/4 = 4 -> 4 samples represents 90 degree phase shift

```
Fs = 50 * 16;  %sample rate
dt=1/Fs;
t=0:dt:15;
```

The reference 50/60 Hz signal is defined bellow:

```
ref = sin(2*pi*f_interference*t);
```

The European grid ranges from Portugal over Poland to Turkey. It is fed with alternating current, which has a frequency of **approximately** 50.0 Hz In the following picture the maximum allowed frequency offset is presented.

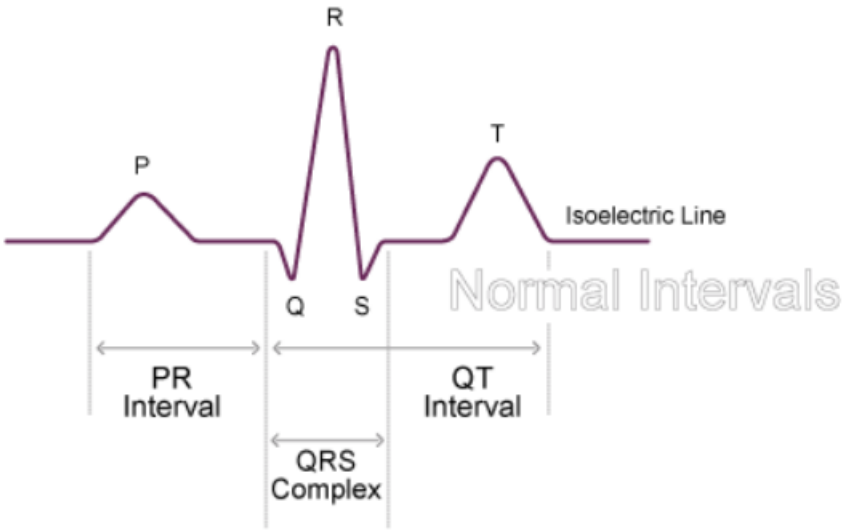


The interference noise is created as shown bellow. It contains the mains 50/60 Hz signal with an offset as mentioned previously. If the h variable is set to 1 the interference noise will also include the third harmonic of the 50/60 Hz signal.

```
h = 0;
f_offset1 = 0.7; % The maximum value should be +/- 0.2 Hz
f_offset2 = 0; % The maximum value should be +/- 0.2 Hz
interference_noise = Mains_interference_amplitude * sin
 (2*pi*(f_interference-f_offset1)*t) + ...
                     h * Mains_interference_amplitude * 0.2
 *sin(2*pi*(3*f_interference-f_offset2)*t);
```

Create ECG simulated signal

The ECG signal is simulated using according to the following picture:



P-R interval = 0.12 - 0.20 sec (3 - 5 small squares)

QRS width = 0.08 - 0.12 sec (2 - 3 small squares)

Q-T interval = 0.35 - 0.43 sec

```
Hearth_rate = 75;
ECG_period = Hearth_rate/60;
QRS_complex_duration = 0.08; % QRS complex duration is between 0.08 -
 0.12 seconds
PR_duration = 0.12;  % PR duration is between 0.12 -0.20 seconds
QT_duration = 0.40; % QT duration is between 0.35 - 0.43 seconds
```

Create QRS complex shape

```
QRS_t = 0:dt:QRS_complex_duration;
QRS_f = 1/QRS_complex_duration;
QRS_waveform = sin(2*pi*QRS_f/2*QRS_t).*(QRS_t<=QRS_complex_duration);
```

Create PR interval shape

```
PR_t = 0:dt:PR_duration;
PR_f = 1/PR_duration;
PR_waveform = 0.2*sin(2*pi*PR_f/2*PR_t).*(PR_t<=PR_duration);
```

Create QT interval shape

```
QT_t = 0:dt:QT_duration;
QT_f = 1/QT_duration;
QT_waveform = 0.2*sin(2*pi*QT_f/2*QT_t).*(QT_t<=QT_duration);
```

Create a vector that holds the exact location of the ECG pulse according to ECG_period

```
time_location = double(0==mod(t,ECG_period));
```

Create PR interval with respect to QRS complex position

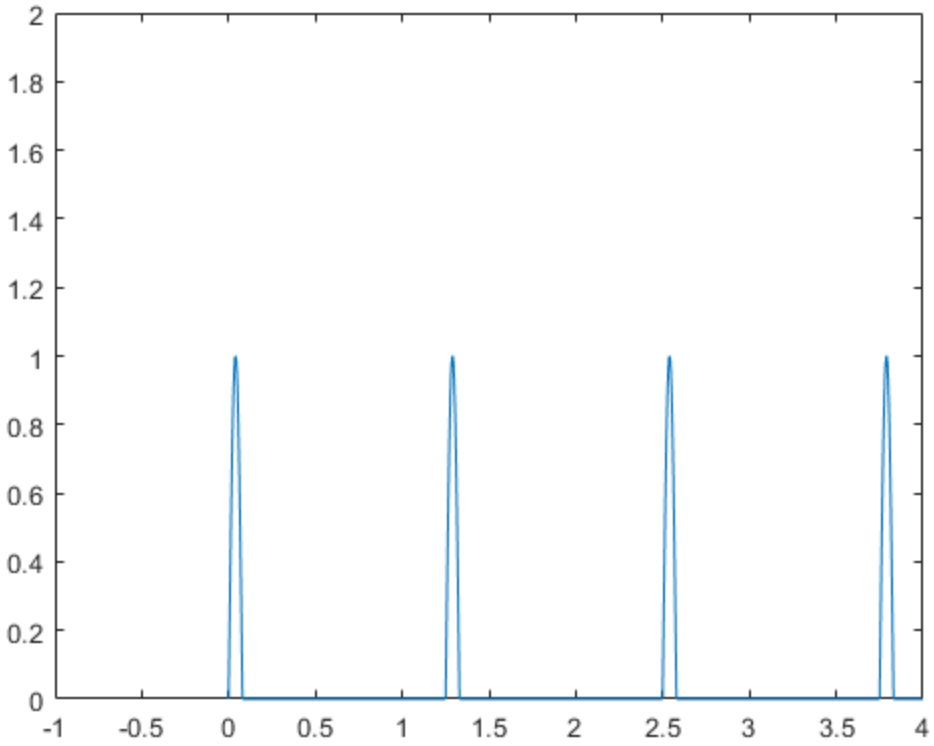
```
ECG_PR = conv(circshift(time_location,[0, -ceil(PR_duration/dt +
 PR_duration/(2*dt))]), PR_waveform);
ECG_PR = ECG_PR(1:length(ECG_PR) - length(PR_waveform) + 1);
```

Create QT interval with respect to QRS complex position

```
ECG_QT = conv(circshift(time_location,QT_duration/(2*dt)),
 QT_waveform);
ECG_QT = ECG_QT(1:length(ECG_QT) - length(QT_waveform) + 1);
```

Repeat QRS complex according to ECG_period.

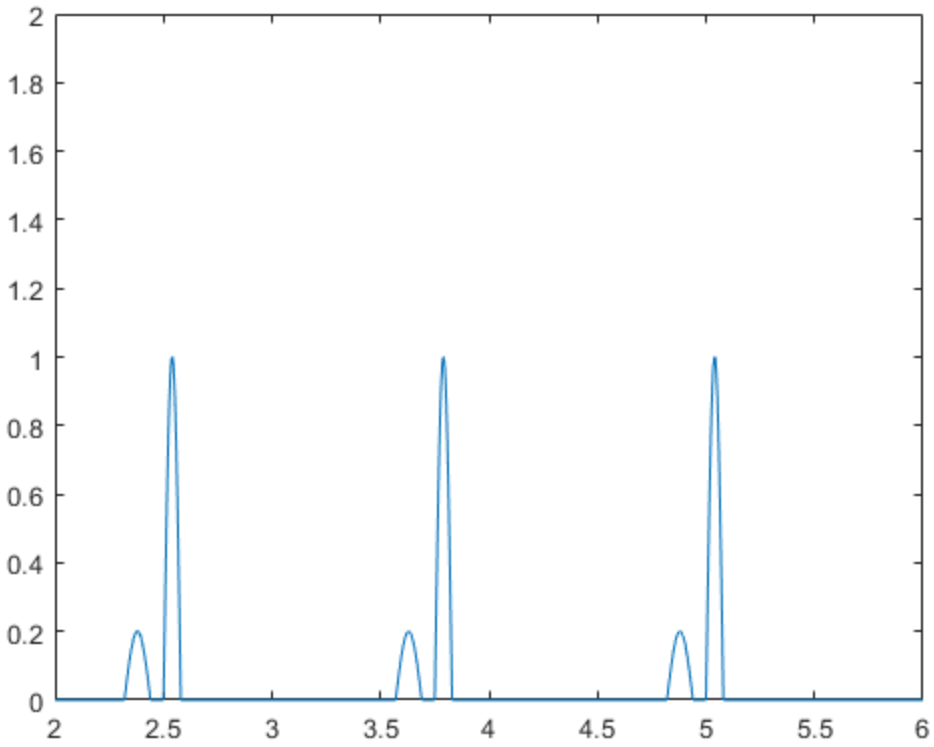
```
ECG_waveform = conv(time_location, QRS_waveform);
figure
plot(t,ECG_waveform(1:end-length(QRS_waveform)+1));
axis([-1 4 0 2])
```



Merge signals PR_QRS_QT

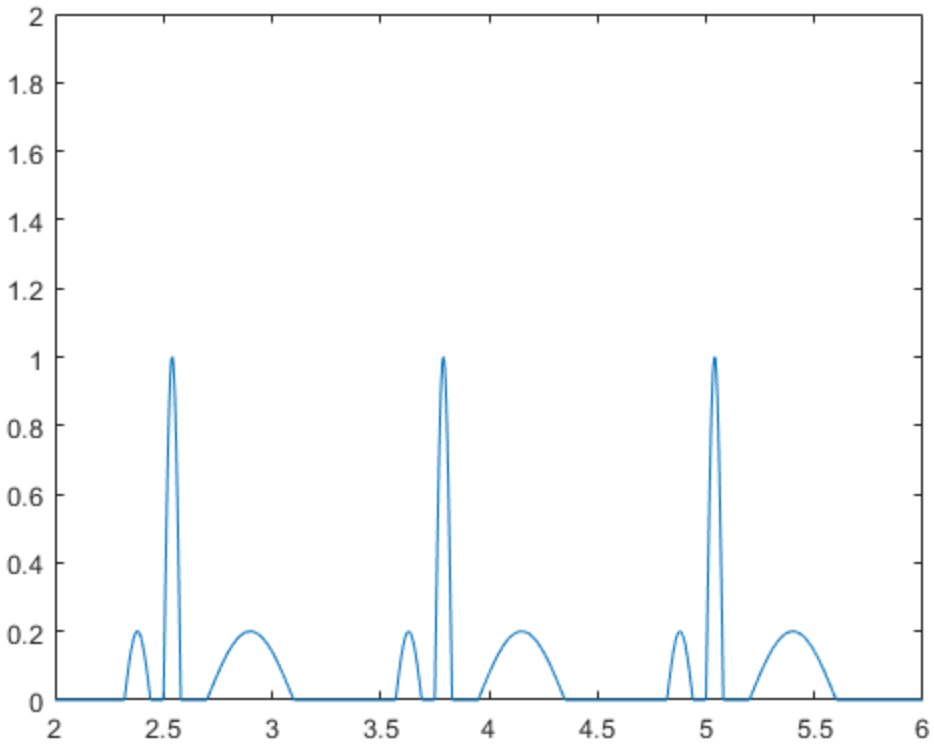
Merge ECG_PR complex

```
ECG_waveform = ECG_waveform(1:length(ECG_waveform) -
 length(QRS_waveform) + 1) + ECG_PR;
figure
plot(t,ECG_waveform);
axis([2 6 0 2])
```



Merge ECG_QT complex

```
ECG_waveform = ECG_waveform + ECG_QT;
figure
plot(t,ECG_waveform);
axis([2 6 0 2])
```



Add signal noise: random noise, interference noise and baseline noise

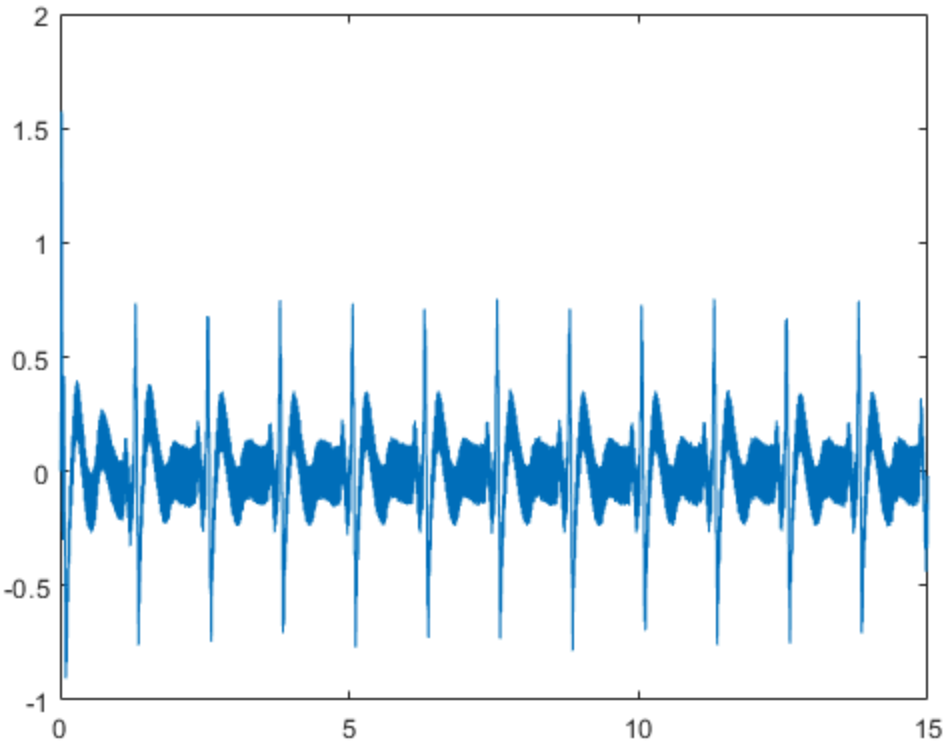
```
ECG_waveform_noise = ECG_waveform + Noise_amplitude * randn(size(t));
ECG_waveform_interference = ECG_waveform_noise + interference_noise;
ECG_waveform_final = ECG_waveform_interference +
 Baseline_wander_amplitude * sin(2*pi*f_baseline*t);

save('ecg_waveform.mat','ECG_waveform');

Options = tfestOptions;
Options.Display = 'on';
Options.WeightingFilter = [];

%tf2 = tfest(mydata, 6, 5, Options)
%idd_data = iddata(ECG_waveform_noise,ECG_waveform,dt);
% h = tf(tf1.Numerator,tf1.Denominator,dt);
%
% lsim_out = lsim(h,ECG_waveform_final,t);
```

Low pass filtering and high pass filtering

The filtering method uses a LPF and a HPF. A 6 pole filter is used first. The output can be seen bellow:

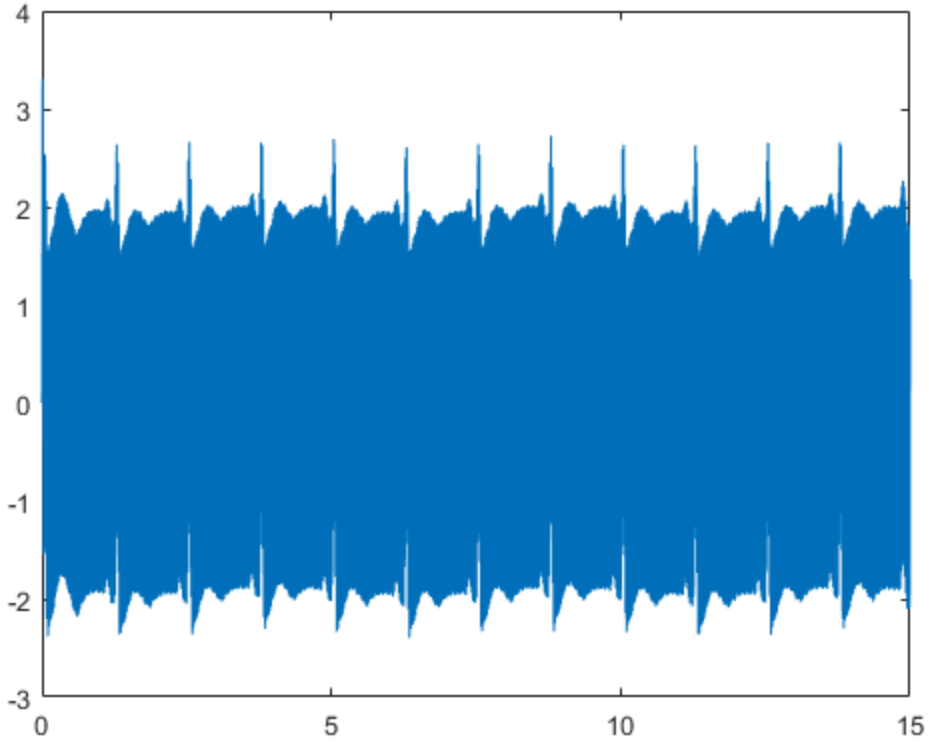
```
[b,a] = butter(6, LPF_cutoff/(Fs/2));
%freqz(b,a)
ECG_LPF = filter(b, a, ECG_waveform_final );
```

```
[b,a] = butter(6, HPF_cutoff/(Fs/2),'high');
%freqz(b,a)
ECG_HPF = filter(b, a, ECG_LPF );
figure
plot(t,ECG_HPF);
```



From an embedded system solution a 2 pole filter is preffered. A 2 pole filter would be much more easier to implement in hardware. Also for a higher order filter (e.g 6) the output ECG signal tends to ring.

```
[b,a] = butter(2, LPF_cutoff/(Fs/2));
%freqz(b,a)
ECG_LPF = filter(b, a, ECG_waveform_final );

[b,a] = butter(2, HPF_cutoff/(Fs/2),'high');
%freqz(b,a)
ECG_HPF = filter(b, a, ECG_LPF );
figure
plot(t,ECG_HPF);
```



LMS 2 tap

The LMS filtering approach is a narrow band filter that works only with frequencies near f_interference

For the 2 tap filter - adjust b1 and b2 coefficients for in phase ref(i) and 90 degree phase shift ref(i-4) signal For instance, if interference signal from ECG_HPF would be in phase with the ref signal then b2 would be 0 and b1 would be Mains_interference_amplitude/ref_amplitude.

```
b1 = 0;
b2 = 0;

for i=5:length(t)

    ECG_out(i) = ECG_HPF(i) - (b1*ref(i) + b2*ref(i-4)) ;
    %b1 = b1 + LMS_conv*ECG_out(i)*sign(ref(i));
    %b2 = b2 + LMS_conv*ECG_out(i)*sign(ref(i-4));
    b1 = b1 + LMS_conv*ECG_out(i)*ref(i);
    b2 = b2 + LMS_conv*ECG_out(i)*ref(i-4);

end
```

LMS 4 taps

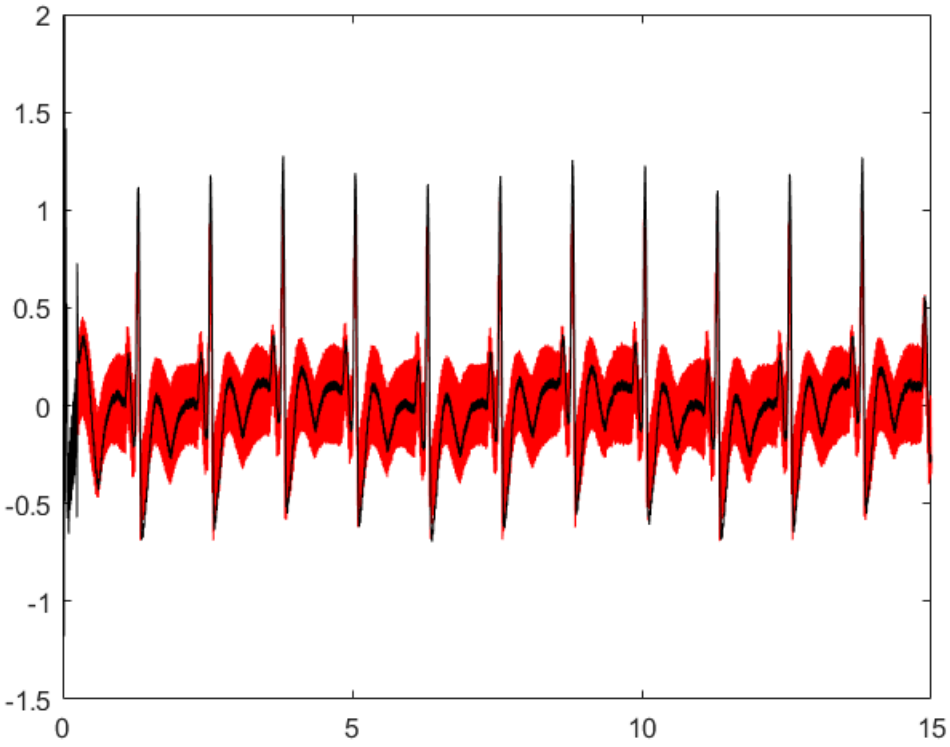
```
a1 = 0;
a2 = 0;
```

```
a3 = 0;
a4 = 0;
for i=13:length(t)

    ECG_4tap(i) = ECG_HPF(i) -  (a1*ref(i) + a2*ref(i-4) + a3*ref(i-8)
 + a4*ref(i-12));
    a1 = a1 + LMS_conv*ECG_4tap(i)*(ref(i));
    a2 = a2 + LMS_conv*ECG_4tap(i)*(ref(i-4));
    a3 = a3 + LMS_conv*ECG_4tap(i)*(ref(i-8));
    a4 = a4 + LMS_conv*ECG_4tap(i)*(ref(i-12));

end
```

LMS n taps

```
%figure
n = 20;
h = zeros(1,n+1);
offset = 0:-1:-n;
for i=n+1:length(t)
    % keep a history buffer
    buffer = ref(i+offset);
    ECG_ntap(i) = ECG_HPF(i) - dot(h,buffer);
    h = h + LMS_conv * ECG_ntap(i) * buffer;
end

%plot(t,ECG_ntap,'red');

n = 200;
h = zeros(1,n+1);
offset = 0:-1:-n;
for i=n+1:length(t)
    % keep a history buffer
    buffer = ref(i+offset);
    ECG_ntap(i) = ECG_HPF(i) - dot(h,buffer);
    h = h + LMS_conv * ECG_ntap(i) * buffer;
end
%hold on
%plot(t,ECG_ntap,'black');
```

The tap order of the filter can reduce the frequency offset of the interference noise. For instance, for a f_offset of 0.7 Hz the output of a 200 tap filter and a 20 tap filter can be seen bellow:




```
% smoothdata - not mandatory, just for eye comparation
% ECG_out  = smoothdata(ECG_out);
% ECG_4tap = smoothdata(ECG_4tap);
% ECG_ntap = smoothdata(ECG_ntap);
```

System identification

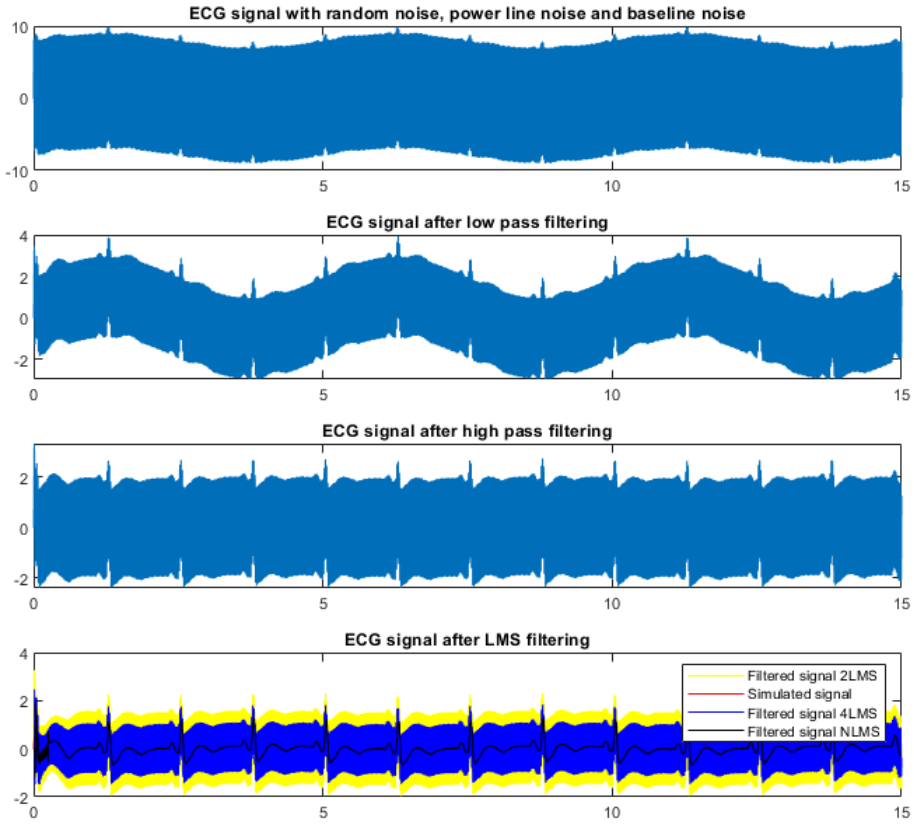
```
%data = iddata(ECG_waveform_final,ECG_waveform,dt);
%a = idtf(tf1.Numerator,tf1.Denominator);
%y = sim(tf1,ECG_waveform);
%sys = tfest(data,1);
% num = sys.Numerator;
% den = sys.den;
% sys.Report

% hz = tf(tf1.Numerator,tf1.Denominator, Fs)
% [y,t]=lsim(hz,ECG_waveform_final);
% stem(t,y);
```

Output plots

ECG signal filtering when interference frequency is equal to reference signal 50/60Hz

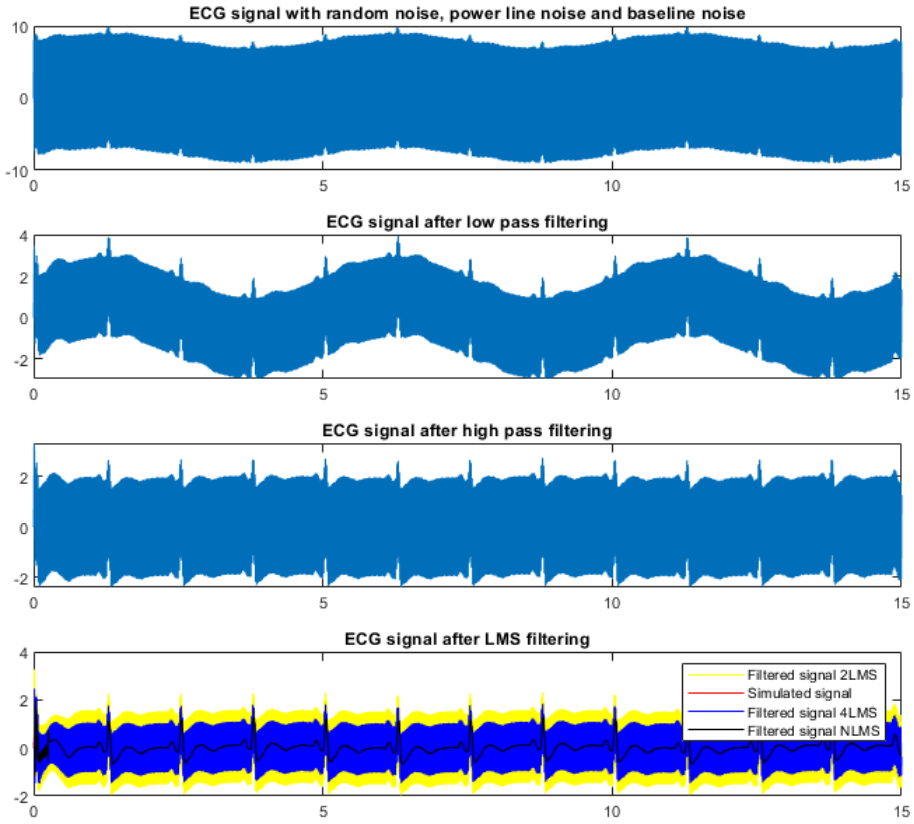
```
% figure
% plot(t,lsim_out);
figure
```

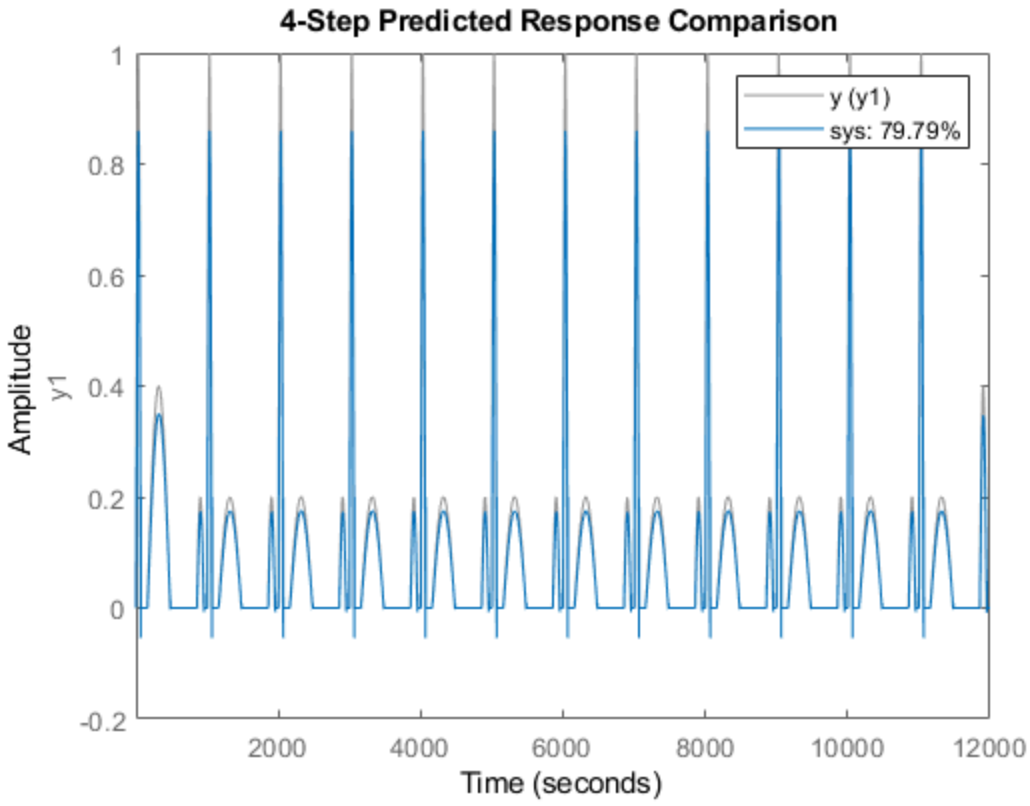
```
subplot(4,1,1);
plot(t,ECG_waveform_final);
title("ECG signal with random noise, power line noise and baseline
 noise");
subplot(4,1,2);
plot(t,ECG_LPF);
title("ECG signal after low pass filtering");
subplot(4,1,3)
plot(t,ECG_HPF);
title("ECG signal after high pass filtering");
subplot(4,1,4);
plot(t,ECG_out,'y');
title("ECG signal after LMS filtering");
hold on
plot(t,ECG_waveform,'r');
hold on
plot(t,ECG_4tap,'b');
hold on
plot(t,ECG_ntap,'black');
legend('Filtered signal 2LMS','Simulated signal', 'Filtered signal
 4LMS', 'Filtered signal NLMS');
set(gcf,'Position',[380 80 800 680]);
%saveas(gcf,'ECG_filtering.png');
```

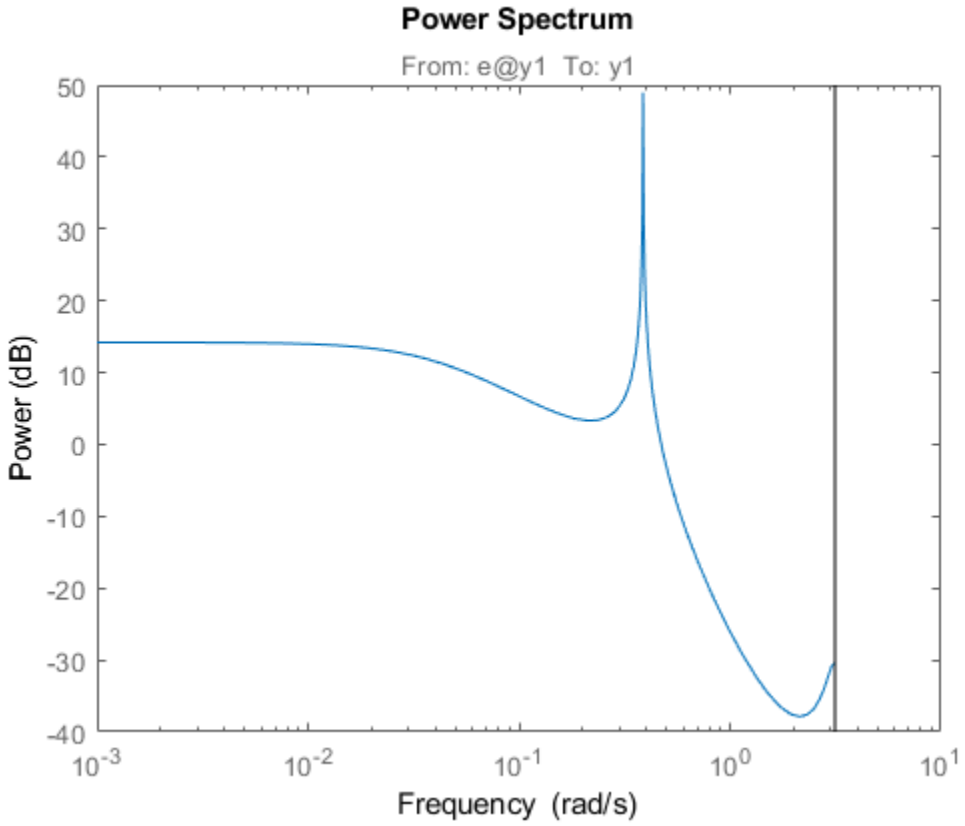


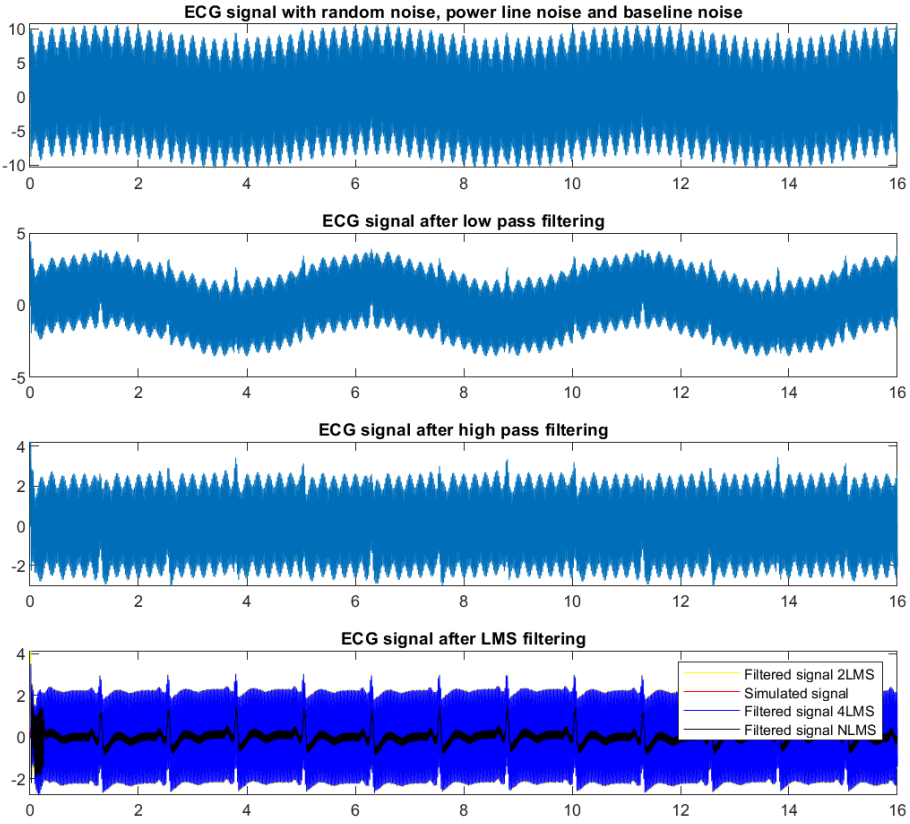
ARIMA

```
% sys = ar(ECG_waveform,4);
% covar = sys.Report.Parameters.FreeParCovariance;

y = iddata(ECG_waveform');
z = iddata(ECG_waveform_final')

figure
sys = ar(ECG_waveform_final,4,'ls');
compare(y,sys,4);
figure
spectrum(sys)


z =

Time domain data set with 12001 samples.
Sample time: 1 seconds

Outputs      Unit (if specified)
   y1
```


ECG signal with random noise, power line noise and baseline noise
ECG signal after low pass filtering
ECG signal after high pass filtering
ECG signal after LMS filtering
Filtered signal 2LMS
Simulated signal
Filtered signal 4LMS
Filtered signal NLMS


4-Step Predicted Response Comparison
y (y1)
sys: 79.79%
Amplitude
y1
1
0.8
0.6
0.4
0.2
0
-0.2
2000
4000
6000
8000
10000
12000
Time (seconds)


Power Spectrum
From: e@y1 To: y1
Power (dB)
50
40
30
20
10
0
-10
-20
-30
-40
10-3
10-2
10-1
100
101
Frequency (rad/s)



Published with MATLAB® R2020a