Student: Baciu Vlad-Eusebiu, N. 0577953, Erasmus student

Mini-project: ECG waveform simulation and different filtering methods.

Course: Voice, Image, Coding, Media and Systems

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:

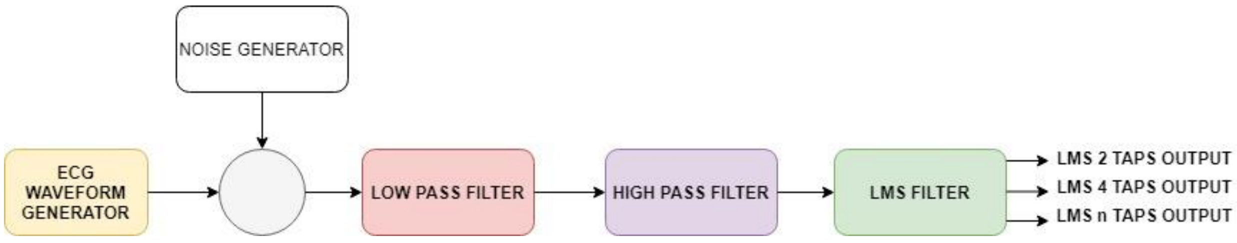

Fig 1 Filter blocks.

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 = 0.005;
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 has a frequency of **approximately** 50.0 Hz In the following picture the maximum allowed frequency offset is presented.

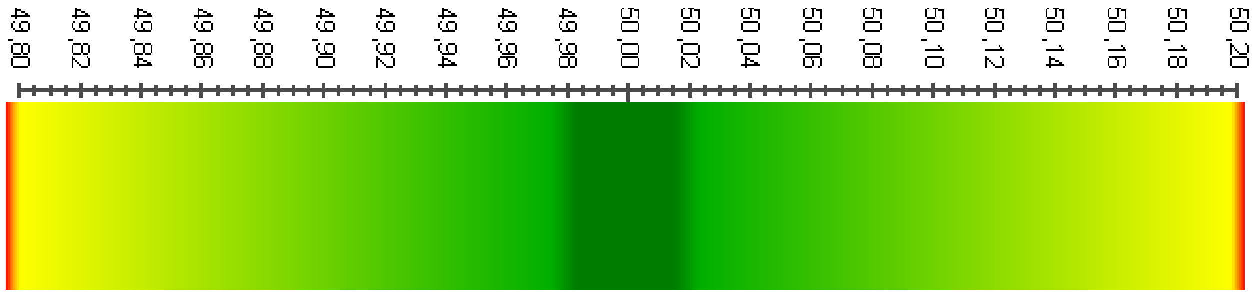

Fig 2 Grid frequency offset

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 = 0.0; % The maximum value should be +/- 0.2 Hz
f_offset2 = 0.0; % 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:

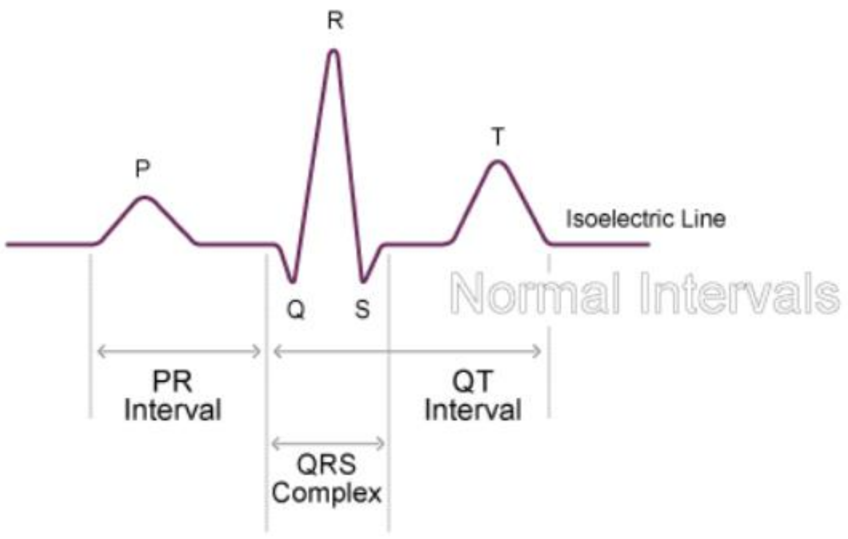

Fig 3 Electrocardiogram waveform

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])
```

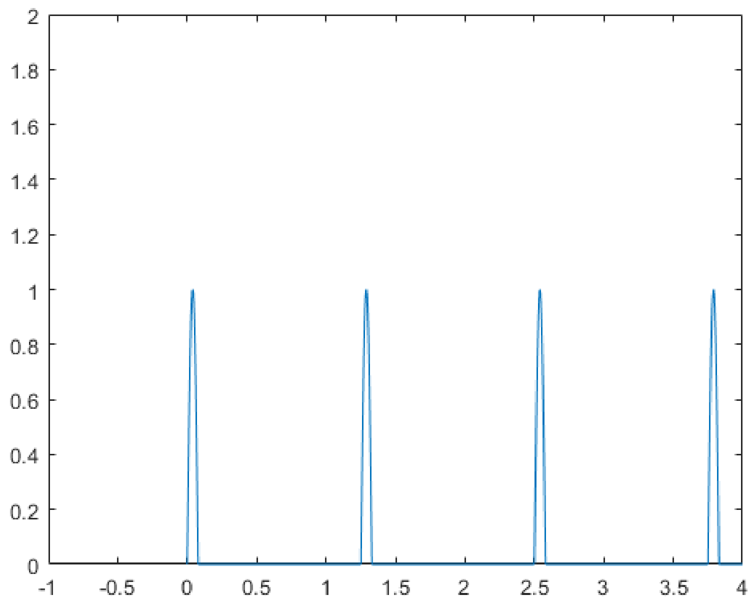

Fig 4 Construct ECG waveform. Place QRS complex.

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])
```

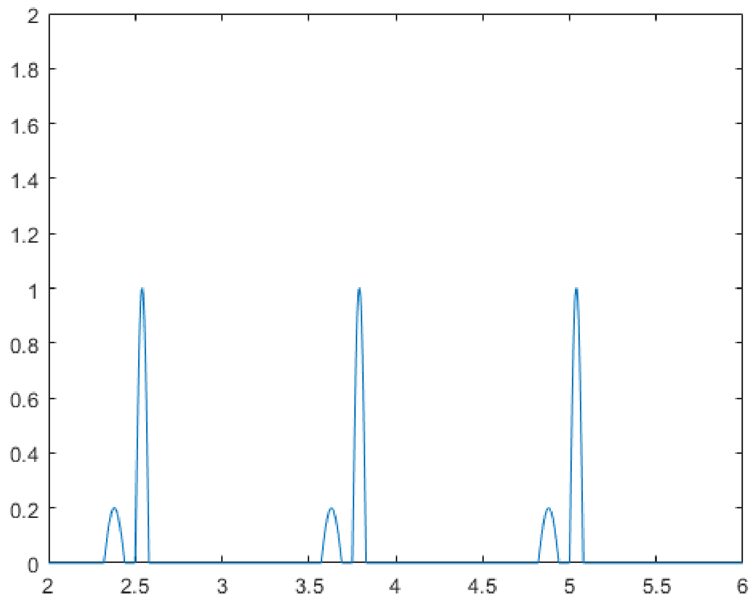

Fig 5 Construct ECG waveform. Place PR complex

Merge ECG_QT complex

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

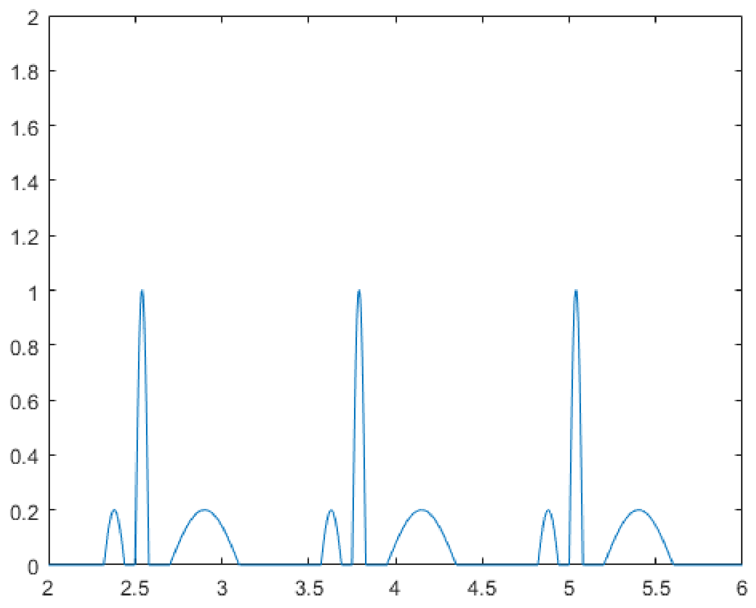

Fig 6 Construct ECG waveform. Place QT complex.

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

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

Add electrode contact noise and abrupt movement noise

The following piece of code is adding noise over the noiseless signal. The noise represents the skin-electrode contact noise and the abrupt movement noise. The signal is filtered later using a reduce Hankel matrix approach.

```
F = 10e3;
t1=0:dt:0.03;
r1 = randi([1 10000],1,length(t));
time_location = double(0==mod(r1,80));
signal1 = 0.01 * sin(2*pi*900*t1) + 0.05 + 0.05 * randn(size(t1));

signal = conv(time_location, signal1);
signal = signal(1:length(signal) - length(signal1) + 1);

ECG_validation_waveform = signal+ECG_waveform;
```

```
ECG_waveform_final = ECG_waveform_final + signal;
save('ecg_waveform.mat','ECG_waveform');
figure
plot(t,ECG_validation_waveform);
axis([0 15 -1 2]);
```

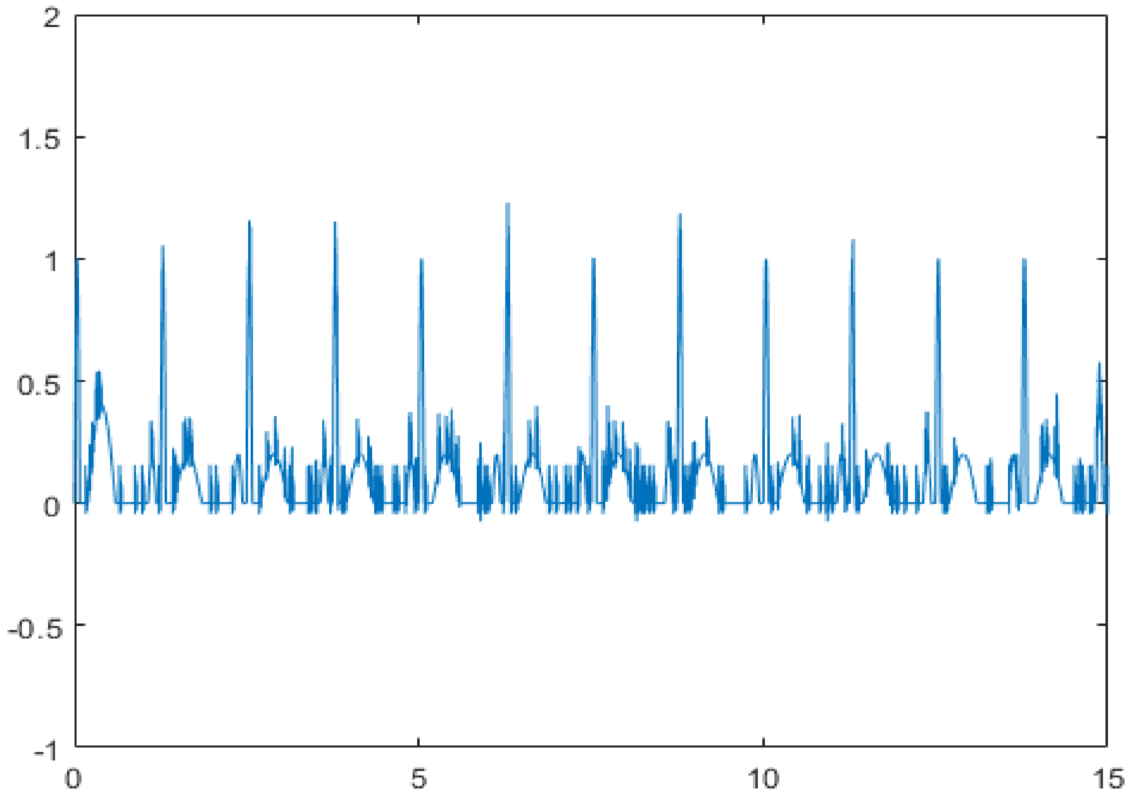

Fig 7 Simulate skin-electrode contact noise.

Low pass filtering and high pass filtering

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
```

Output plots

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

```
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');
```

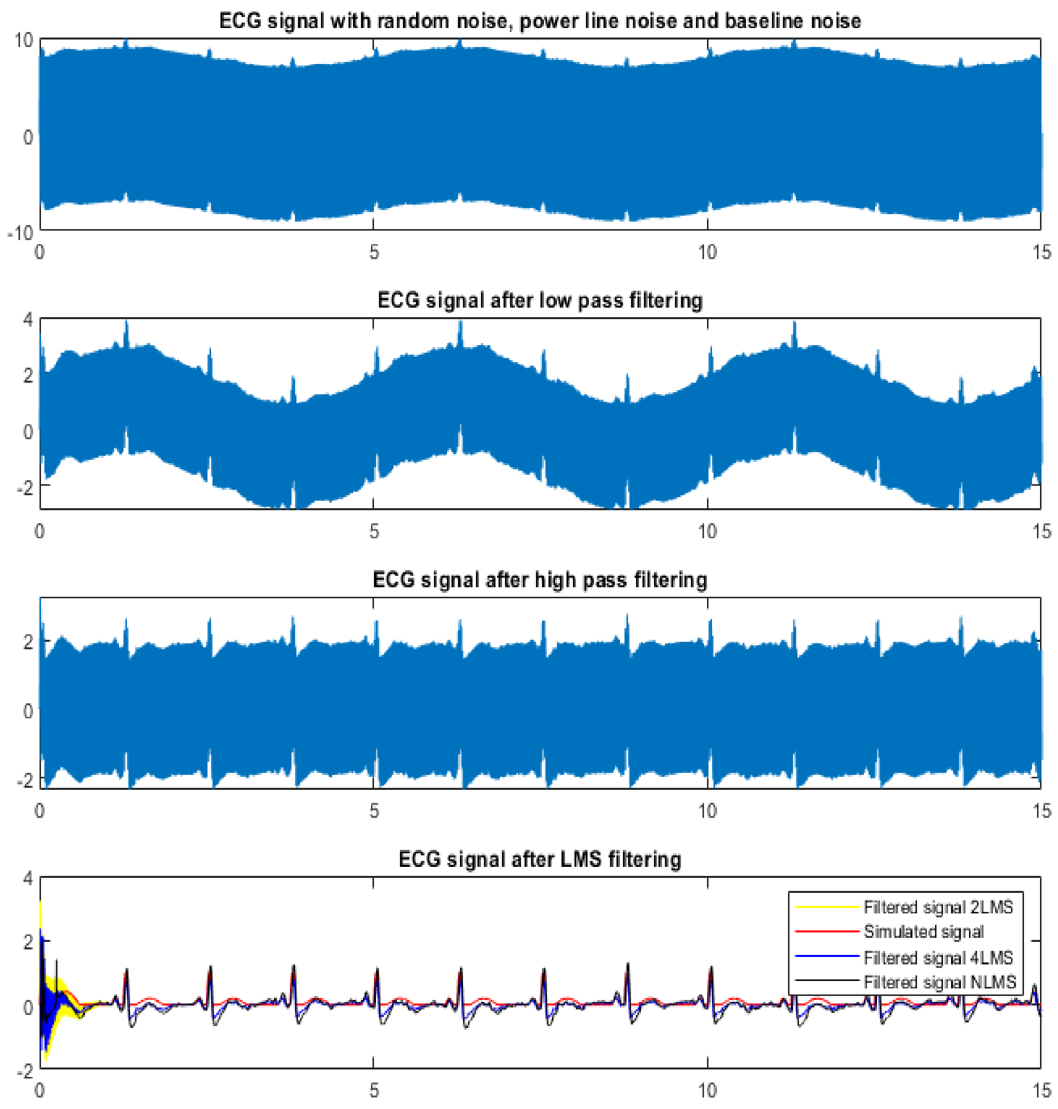

Fig 8 Noise filtering. LPF, HPF and LMS 2tap, 4tap, Ntap

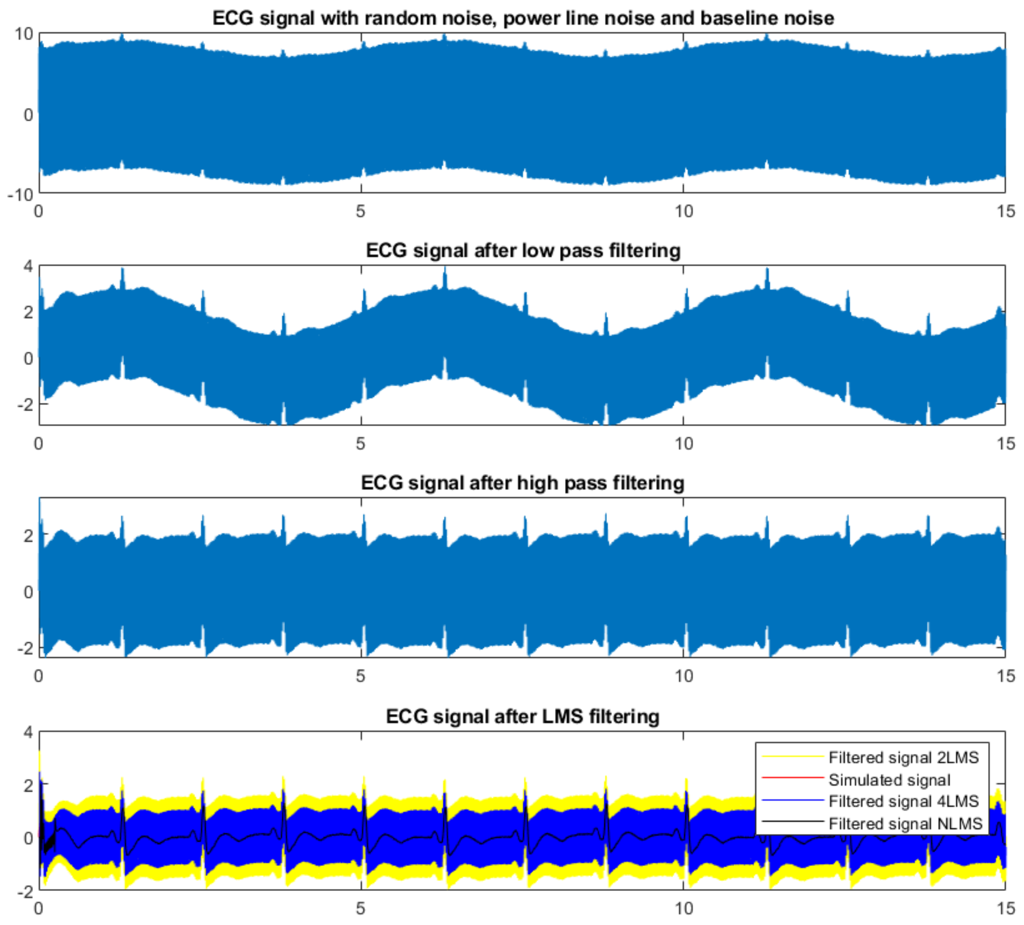

Fig 9 Noise filtering. LPF, HPF and LMS 2tap, 4tap, Ntap when a frequency offset is introduced.

Filtering with reduced Henkel Matrix

The following method is proposed in the following publication ***Filtering via Rank-Reduced Hankel Matrix, CEE 690 , ME 555 — System Identification — Fall, 2013***

```
N = 2048;
y = ECG_validation_waveform(1:N);
dt = 0.05;
t = [1: N ]* dt ;
y2= y;
m = ceil(0.6* N + 1 );
n = length(y) +1 - m;
Y = zeros (m , n );
sv_ratio = 0.04;
```

Create a Henkel matrix with the input signal ECG_validation_waveform

```
for k =1: m
    Y ( k , :) = y ( k :k +n -1 );
end

[U ,S ,V] = svd (Y , 0);
```

Get a maximum index from the matrix of singluar values

```
f = find(diag(S)/ S(1 ,1) > sv_ratio );
K = max(f);
d1= diag(S)';
```

Create a rank-reduced Hankel Matrix according to K index

```
d = diag(d1(1:K));
v = V(:,1: K)';
Y = U(:,1: K )*d* v;


y = zeros (1 , N );
y (1) = Y (1 ,1);
```

Filter the signal

```
for k =2: m
    min_kn = min(k , n );
    first_col = flip(Y(1:1:k ,1: min_kn));
    y(k) = sum(diag(first_col)) / min_kn ;
end


for k =2: n
    last_col = flip(Y(m-n+k:1:m,k:n));
    y(m+k -1) = sum(diag(last_col)) / (n -k +1);
end
figure
plot (t , y2,t,y)
```

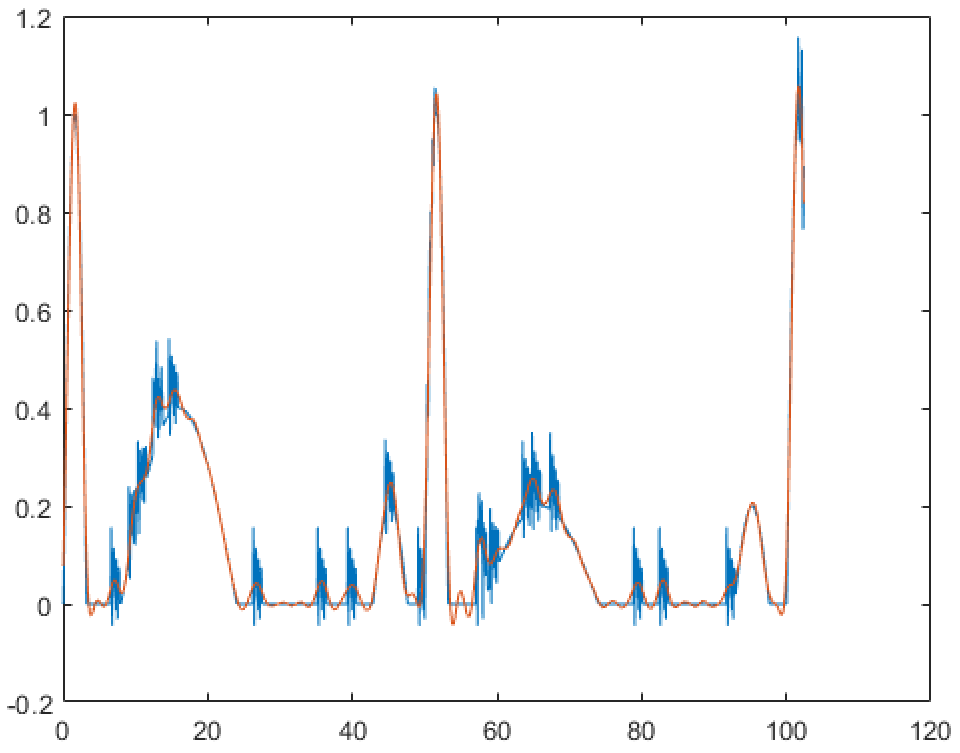

Fig 10 Reduced Hankel matrix filtering

References

http://www.medicine.mcgill.ca/physio/vlab/cardio/introecg.htm

https://homes.esat.kuleuven.be/~tvanwate/courses/alari/1617/lectureslides/chapter6-Wiener%20Filters%20and%20the%20LMS%20Algorithm-pp32.pdf

https://www.keil.com/pack/doc/CMSIS/DSP/html/group__LMS.html

http://homepages.vub.ac.be/~imarkovs/publications/cd-est.pdf

http://people.duke.edu/~hpgavin/SystemID/CourseNotes/SVD_filter-plots.pdf

Commented code

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

% y = iddata(ECG_waveform');
% z = iddata(ECG_waveform_final')
% na = 1;
% nb = 1;
% nc = 1;
% nk = 1;
% sys_armax = armax(y,[4 1])
% figure

% compare(y,sys,4);
% figure
% compare(y,sys_armax,4);
% figure
% spectrum(sys)
% y = iddata(ECG_validation_waveform');
% z = iddata((ECG_validation_waveform-ECG_waveform)')
%
%
% sys = ar(ECG_waveform,4,'ls');
%
% x1 = compare(z,sys,6)
% x2 = compare(y,sys,4);
%
% %t = iddata(t');
% noise = x2.y' - ECG_waveform;
% figure
% plot(t,x1.y);
% hold on
% plot(t,signal);
% figure
% plot(t,x2.y);
% hold on
% plot(t,ECG_validation_waveform);
% figure
% plot(t,noise);
%
% ECG_waveform = ECG_validation_waveform - noise;
% figure
% plot(t,ECG_waveform);
% %% ARMA
% Mdl = arima(1,0,1)
% EstMdl1 = estimate(Mdl,ECG_waveform');
% summarize(EstMdl1)
% EstMdl2 = estimate(Mdl,ECG_artifact');
% summarize(EstMdl2)
% X = smooth(EstMdl1,ECG_waveform);
% % [y,e,v] = simulate(Mdl,100);
% % Z = e./sqrt(v);
% % [Y,E,V] = filter(Mdl,Z)
% % plot(t,Y);

%hold on
%plot(t,ECG_ntap,'black');
```

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



%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);
```

Published with MATLAB® R2020a