Guia 3 - Ejercicio 1

Table of Contents

Código fuente y resultados

```
clear all;
close all;

numericaltour;
```

Local Fourier analysis of sound

First we load a sound, with a slight sub-sampling

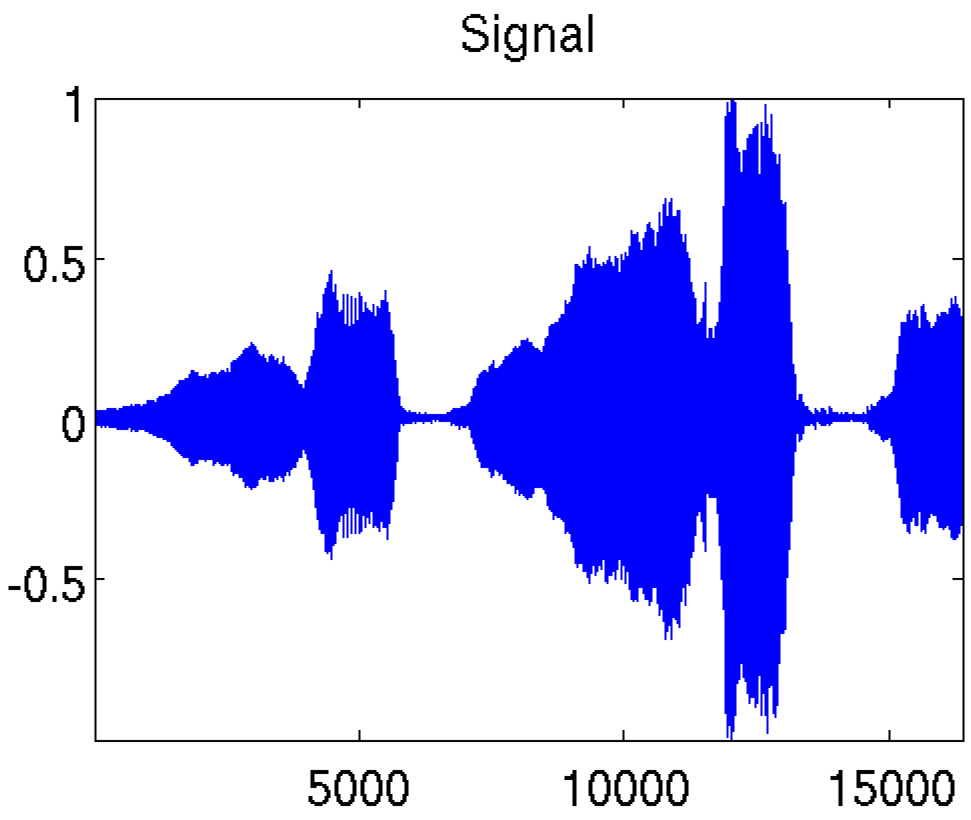
```
n = 1024*16;
options.n = n;
[x,fs] = load_sound('bird', n);
```

You can actually play a sound.

```
sound(x(:)',fs);
```

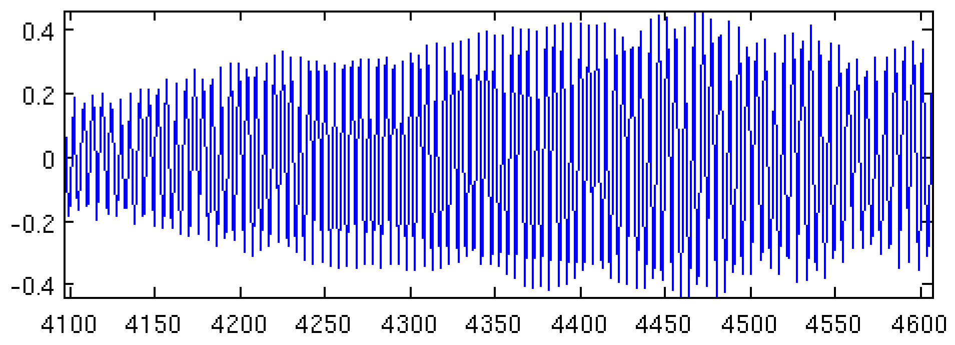
We can display the sound.

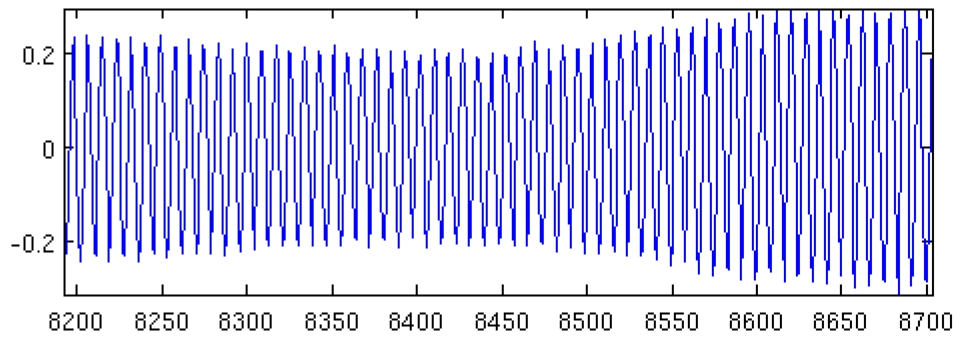
```
figure();
clf;
plot(1:n,x);
axis('tight');
set_graphic_sizes([], 20);
title('Signal');
```



Local zooms on the sound show that it is highly oscilating.

```
figure();
p = 512;
t = 1:n;
clf;
sel = n/4 + (0:p-1);
subplot(2,1,1);
plot(t(sel),x(sel)); axis tight;
sel = n/2 + (0:p-1);
subplot(2,1,2);
plot(t(sel),x(sel)); axis tight;
```



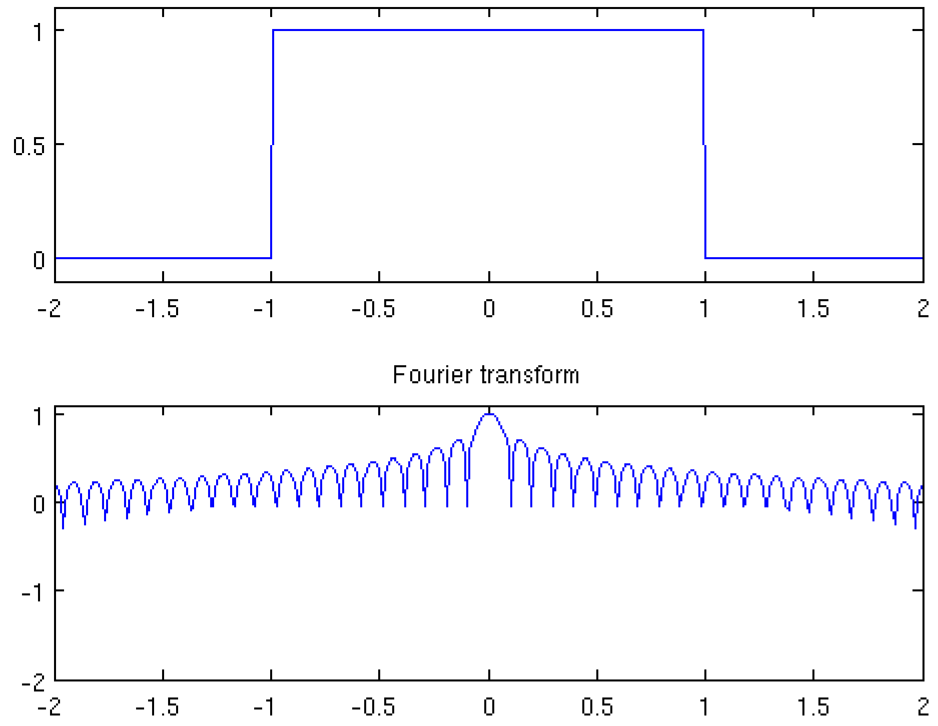


A good windowing function should balance both time localization and frequency localization.

```
t = linspace(-10,10,2048);
eta = 1e-5;
vmin = -2;
```

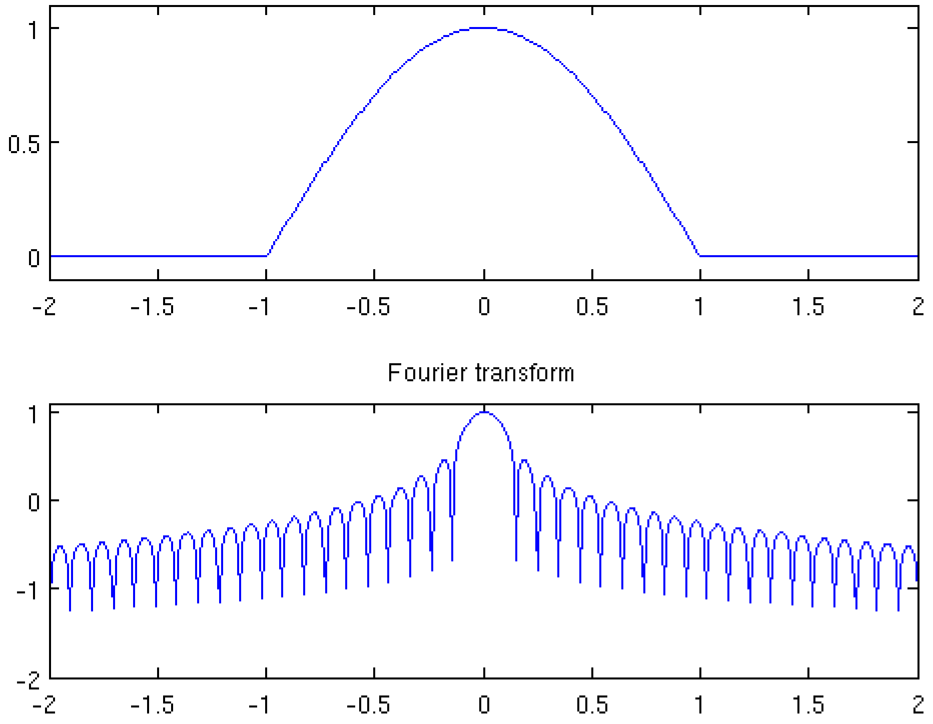
The block window has a sharp transition and thus a poor frequency localization.

```
h = double( abs(t)<1 );
hf = fftshift(abs(fft(h)));
hf = log10(eta+hf); hf = hf/max(hf);
clf;
subplot(2,1,1);
title('Block window');
plot(t, h); axis([-2 2, -.1, 1.1]);
subplot(2,1,2);
plot(t, hf); axis([-2 2, vmin, 1.1]);
title('Fourier transform');
```



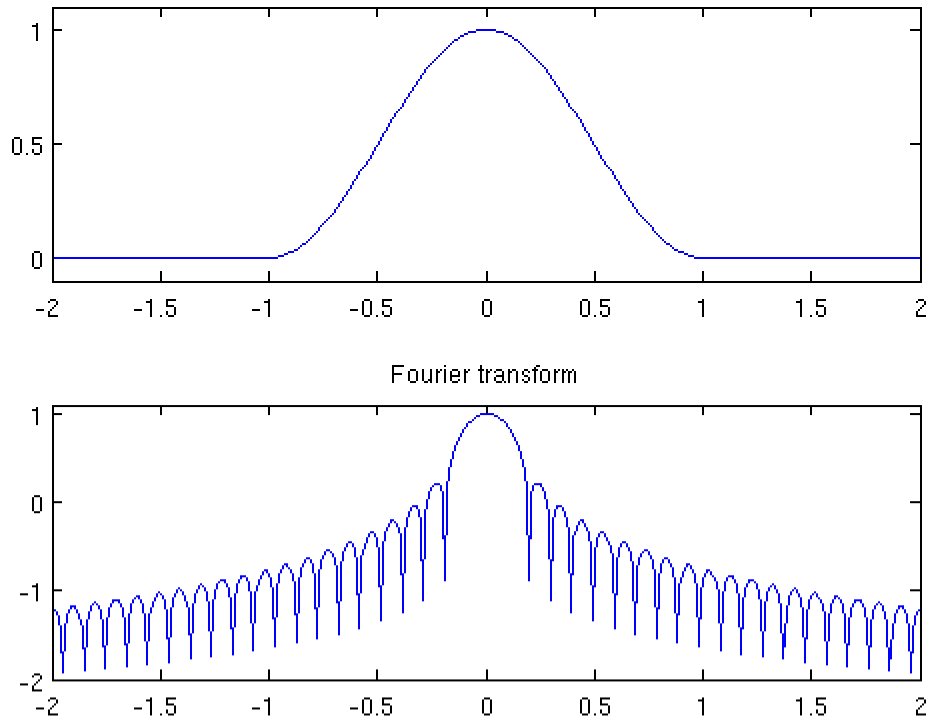
A Hamming window is smoother.

```
h = cos(t*pi()/2) .* double(abs(t)<1);
hf = fftshift(abs(fft(h)));
hf = log10(eta+hf); hf = hf/max(hf);
clf;
subplot(2,1,1);
title('Hamming window');
plot(t, h); axis([-2 2, -.1, 1.1]);
subplot(2,1,2);
plot(t, hf); axis([-2 2, vmin, 1.1]);
title('Fourier transform');
```



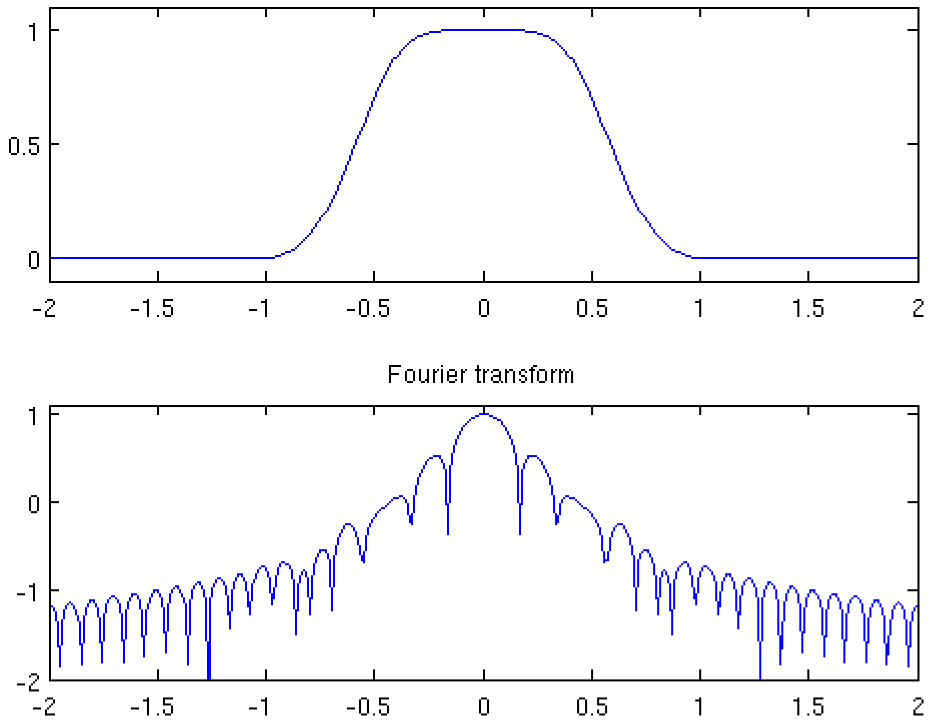
A Haning window has continuous derivatives.

```
h = (cos(t*pi())+1)/2 .* double(abs(t)<1);
hf = fftshift(abs(fft(h)));
hf = log10(eta+hf); hf = hf/max(hf);
clf;
subplot(2,1,1);
title('Haning window');
plot(t, h); axis([-2 2, -.1, 1.1]);
subplot(2,1,2);
plot(t, hf); axis([-2 2, vmin, 1.1]);
title('Fourier transform');
```



A normalized Haning window has a sharper transition. It has the advantage of generating a tight frame STFT, and is used in the following.

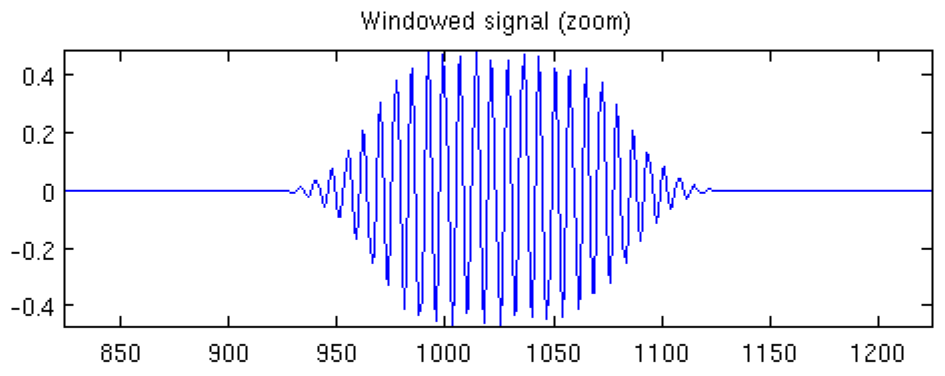
```
h = sqrt(2)/2 * (1+cos(t*pi())) ./ sqrt( 1+cos(t*pi()).^2 ) .* double(abs(t)<1);
hf = fftshift(abs(fft(h)));
hf = log10(eta+hf); hf = hf/max(hf);
clf;
subplot(2,1,1);
title('Normalized Haning window');
plot(t, h); axis([-2 2, -.1, 1.1]);
subplot(2,1,2);
plot(t, hf); axis([-2 2, vmin, 1.1]);
title('Fourier transform');
```

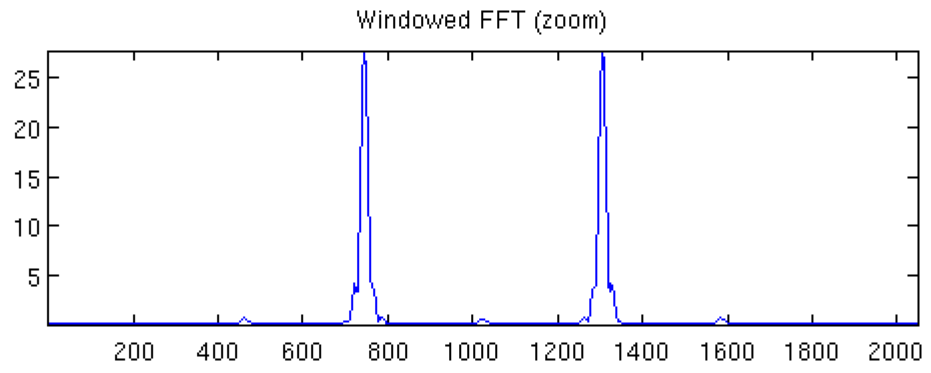


Exercice 1: (the solution is exo1.m)

Compute the local Fourier transform around a point t0 of x, which is the FFT (use the function fft) of the windowed signal x.*h where h is smooth windowing function located around t0. For instance you can use for h a Gaussian bump centered at t0. To center the FFT for display, use fftshift.

```
figure();
t0 = 9500;
wl = length(hf);
wx = x(t0-floor(wl*0.5):t0+ceil(wl*0.5)-1).*h';
subplot(2,1,1),plot(wx);
axis([wl*0.5-200 wl*0.5+200 min(wx) max(wx)])
title('Windowed signal (zoom)')
subplot(2,1,2),plot(fftshift(abs(fft(wx))));
title('Windowed FFT (zoom)')
axis('tight')
```





Short time Fourier transform.

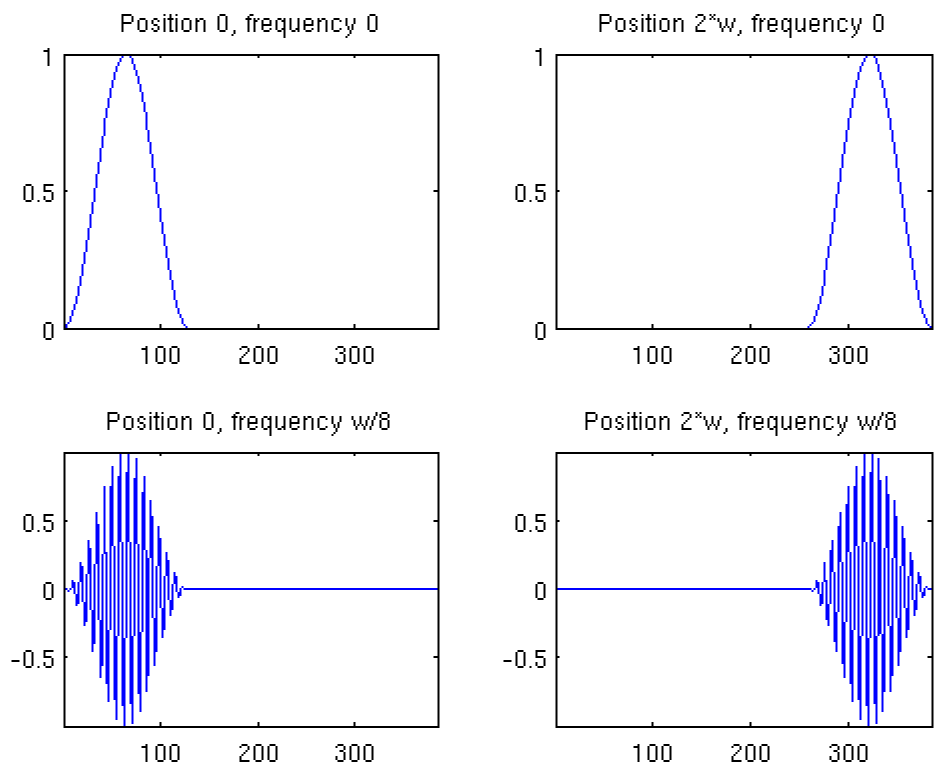
The only parameters of the transform are the size of the window and the overlap.

```
% size of the window
w = 64*2;
% overlap of the window
q = w/2;
```

Gabor atoms are computed using a Haning window. The atoms are obtained by translating in time and in frequency (modulation) the window.

```
t = 0:3*w-1;
t1 = t-2*w;
f = w/8;
% Position 0, frequency 0.
g1 = sin( pi*t/w ).^2 .* double(t<w);
% Position 2*w, frequency 0.
g2 = sin( pi*t1/w ).^2 .* double( t1<w & t1>=0 );
% Position 0, frequency w/8
g3 = g1 .* sin( t * 2*pi/w * f);
% Position 2*w, frequency w/8
g4 = g2 .* sin( t * 2*pi/w * f);
% display
figure()
clf;
```

```
subplot(2,2,1);
plot(g1); axis('tight');
title('Position 0, frequency 0');
subplot(2,2,2);
plot(g2); axis('tight');
title('Position 2*w, frequency 0');
subplot(2,2,3);
plot(g3); axis('tight');
title('Position 0, frequency w/8');
subplot(2,2,4);
plot(g4); axis('tight');
title('Position 2*w, frequency w/8');
```

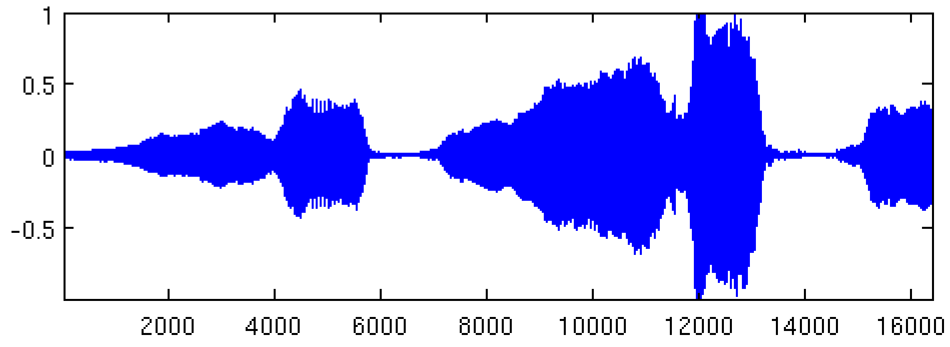


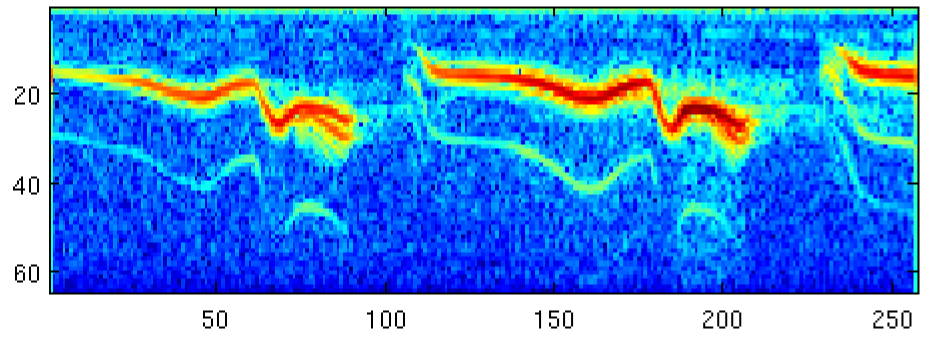
We can compute a spectrogram of the sound to see its local Fourier content. The number of windowed used is (n-noverlap)/(w-noverlap)

```
S = perform_stft(x,w,q, options);
```

To see more clearly the evolution of the harmonics, we can display the spectrogram in log coordinates. The top of the spectrogram corresponds to low frequencies.

```
% display the spectrogram
clf; imageplot(abs(S)); axis('on');
% display log spectrogram
plot_spectrogram(S,x);
```





The STFT transform is decomposing the signal in a redundant tight frame. This can be checked by measuring the energy conservation.

```
% energy of the signal
e = norm(x,'fro').^2;
% energy of the coefficients
eS = norm(abs(S),'fro').^2;
disp(strcat(['Energy conservation (should be 1)=' num2str(e/eS)]));
```

```
        Energy conservation (should be 1)=1
```

One can also check that the inverse transform (which is just the transposed operator - it implements exactly the pseudo inverse) is working fine.

```
% one must give the signal size for the reconstruction
x1 = perform_stft(S,w,q, options);
disp(strcat(['Reconstruction error (should be 0)=' ...
    num2str( norm(x-x1, 'fro')./norm(x,'fro') ) ]));
```

```
        Reconstruction error (should be 0)=2.2398e-16
```

Audio Denoising

One can perform denosing by a non-linear thresholding over the transfomede Fourier domain.

First we create a noisy signal

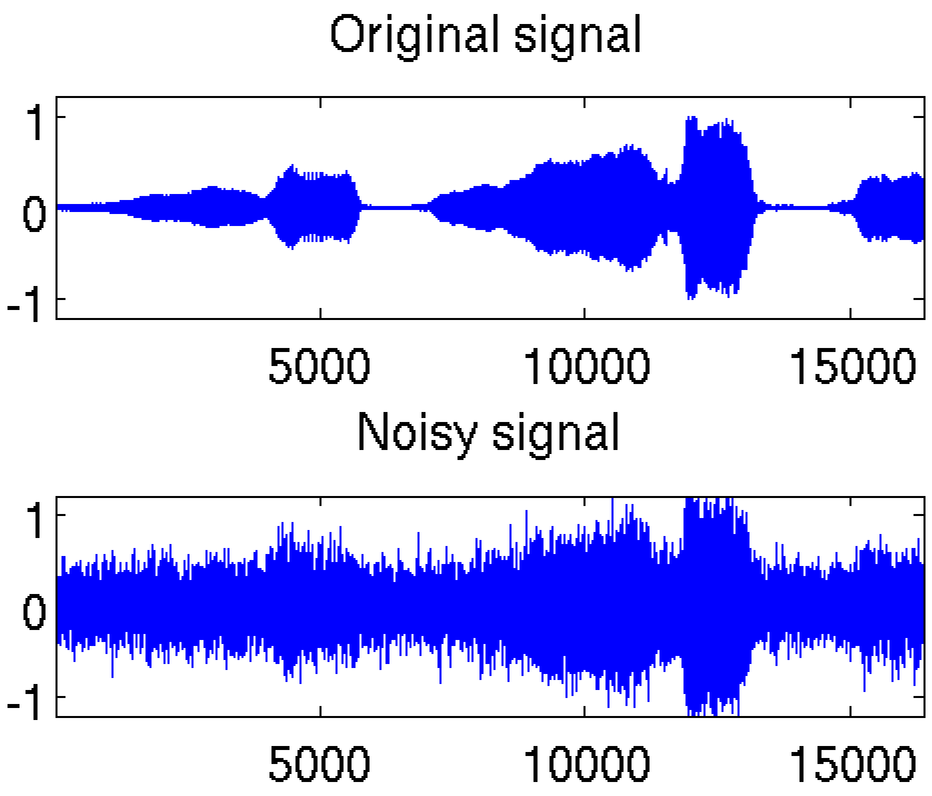
```
sigma = .2;
snoise = randn(size(x))*sigma;
xn = x + snoise;
```

Play the noisy sound.

```
sound(xn,fs);
```

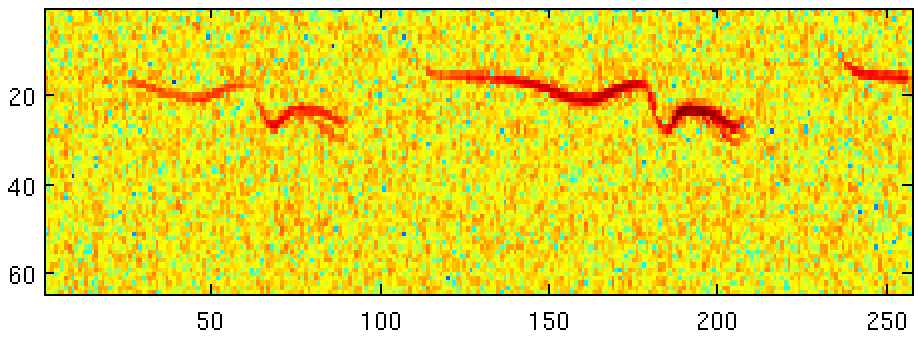
Display the Sounds.

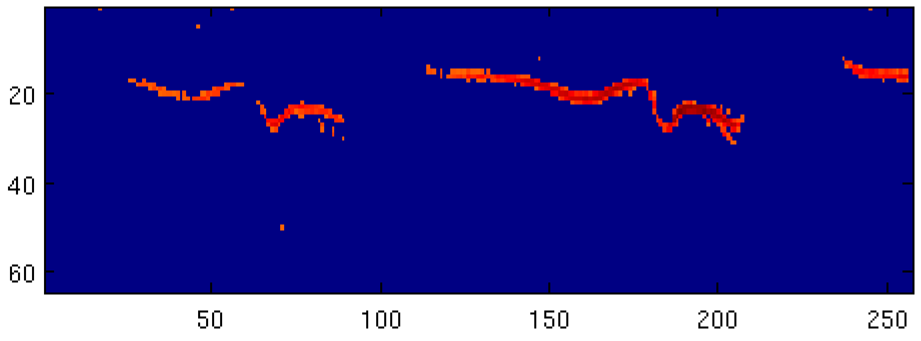
```
clf;
figure()
subplot(2,1,1);
plot(x); axis([1 n -1.2 1.2]);
set_graphic_sizes([], 20);
title('Original signal');
subplot(2,1,2);
plot(xn); axis([1 n -1.2 1.2]);
set_graphic_sizes([], 20);
title('Noisy signal');
```



One can threshold the spectrogram.

```
% perform thresholding
Sn = perform_stft(xn,w,q, options);
SnT = perform_thresholding(Sn, 2*sigma, 'hard');
% display the results
figure()
subplot(2,1,1);
plot_spectrogram(Sn);
subplot(2,1,2);
plot_spectrogram(SnT);
```



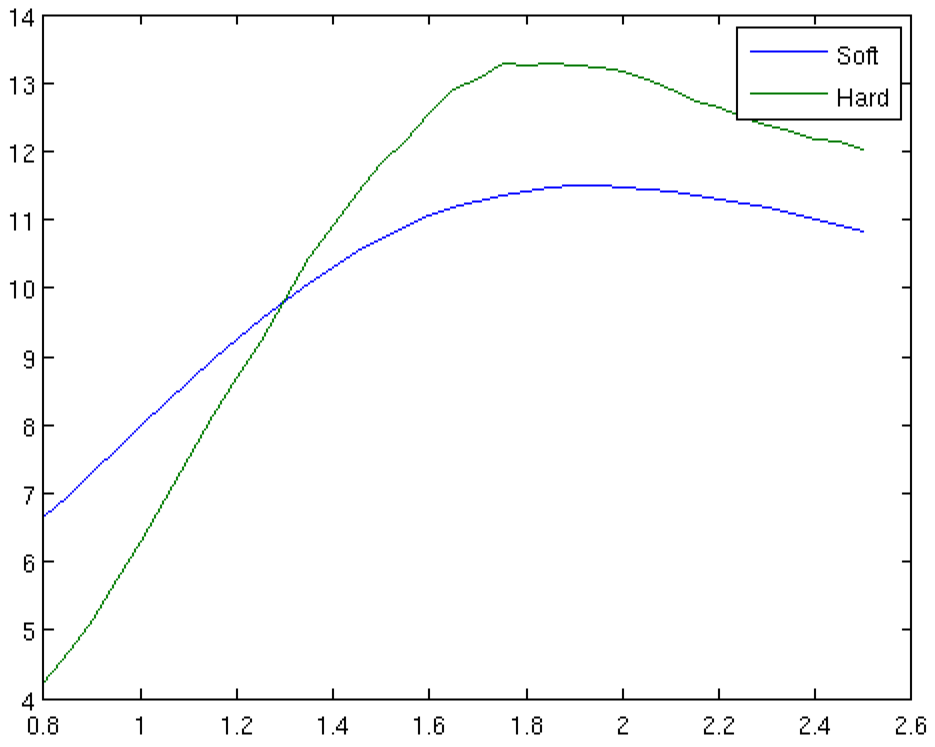


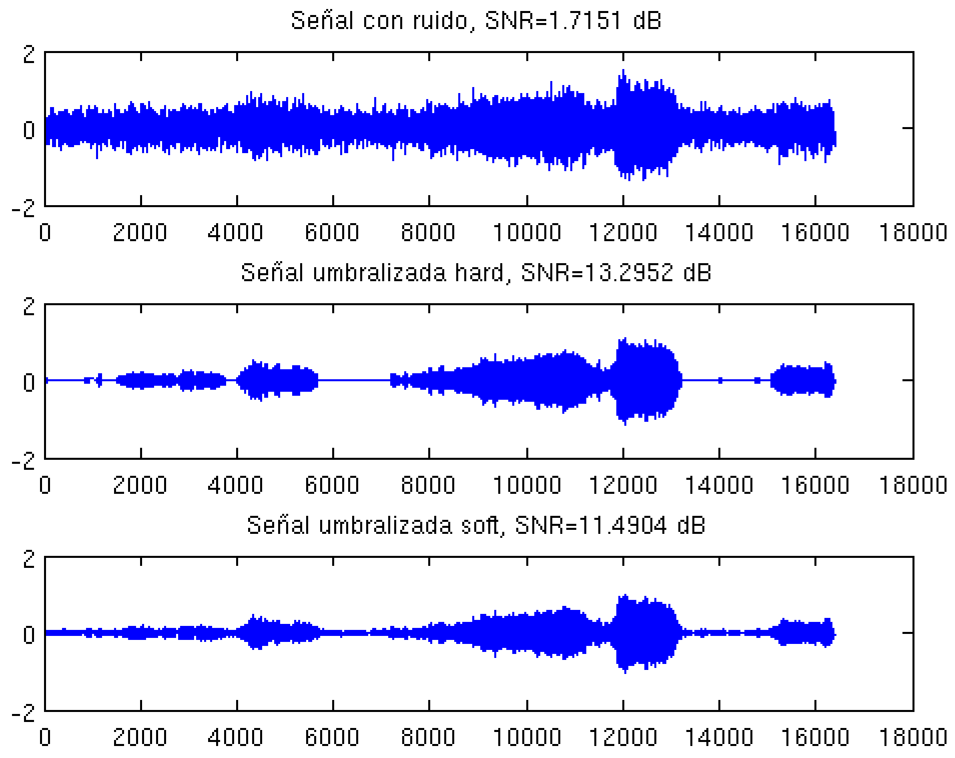
Exercice 2: (the solution is exo2.m)

A denoising is performed by hard or soft thresholding the STFT of the noisy signal. Compute the denosing SNR with both soft and hard thresholding, and compute the threshold that minimize the SNR. Remember that a soft thresholding should be approximately twice smaller than a hard thresholding. Check the result by listening. What can you conclude about the quality of the denoised signal ?

```
x2h = perform_stft(SnT,w,q, options);

SnT = perform_thresholding(Sn, sigma, 'soft');
figure()
subplot(2,1,1);
plot_spectrogram(Sn);
subplot(2,1,2);
plot_spectrogram(SnT);

% Reconstruyo

x2s = perform_stft(SnT,w,q, options);
sound(x2h,fs);
sound(x2s,fs);


close all;
Tsigma = 0.8:0.05:2.5;
snr_soft = 0.*Tsigma;
```

```
snr_hard = 0.*Tsigma;

% Acá está más explicado:
% https://www.ceremade.dauphine.fr/~peyre/numerical-tour/tours/denoisingwav_2_wave
% universal threshold rule
% T = sigma*sqrt(2*log(N))
for k=1:1:length(Tsigma)

    SnTh = perform_thresholding(Sn, Tsigma(k)*sigma, 'hard');
    x2h = perform_stft(SnTh,w,q, options);
    snr_hard(k) = snr(x,x2h);

    % Sugerencia del autor del tutorial: Remember that a soft thresholding
    % should be approximately twice smaller than a hard thresholding.
    SnTs = perform_thresholding(Sn, Tsigma(k)*sigma*0.5, 'soft');
    x2s = perform_stft(SnTs,w,q, options);
    snr_soft(k) = snr(x,x2s);

end



figure()
clf;
plot(Tsigma,snr_soft,Tsigma,snr_hard);
legend('Soft','Hard');

% Veo los mejores umbrales
[ms,poss] = max(snr_soft);
[mh,posh] = max(snr_hard);

SnTh = perform_thresholding(Sn, Tsigma(posh)*sigma, 'hard');
x2h = perform_stft(SnTh,w,q, options);
SnTs = perform_thresholding(Sn, Tsigma(poss)*sigma*0.5, 'soft');
x2s = perform_stft(SnTs,w,q, options);

figure()
subplot(3,1,1),plot(xn);
title(sprintf('Señal con ruido, SNR=%.4f dB',snr(x,xn)));
subplot(3,1,2),plot(x2h);
title(sprintf('Señal umbralizada hard, SNR=%.4f dB',snr(x,x2h)));
subplot(3,1,3),plot(x2s);
title(sprintf('Señal umbralizada soft, SNR=%.4f dB',snr(x,x2s)));
```


Soft
Hard

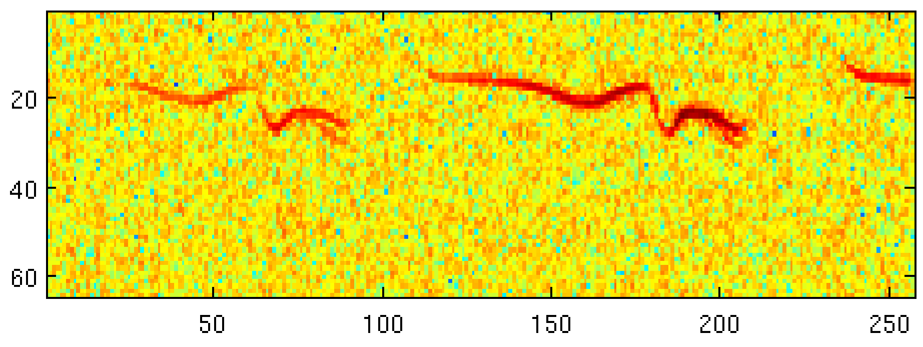

Señal con ruido, SNR=1.7151 dB
Señal umbralizada hard, SNR=13.2952 dB
Señal umbralizada soft, SNR=11.4904 dB

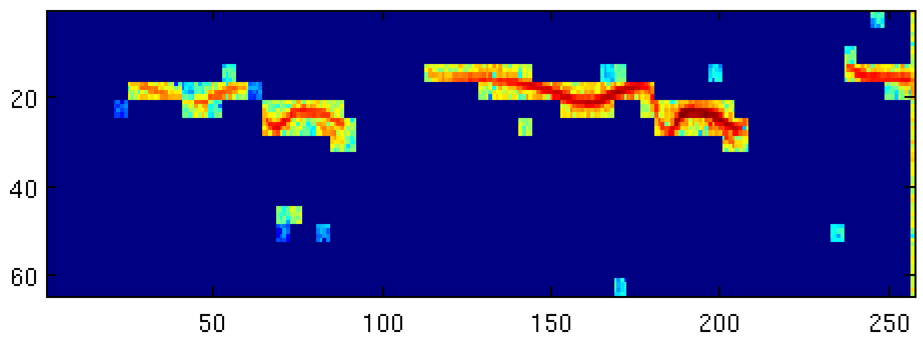
Audio Block Thresholding

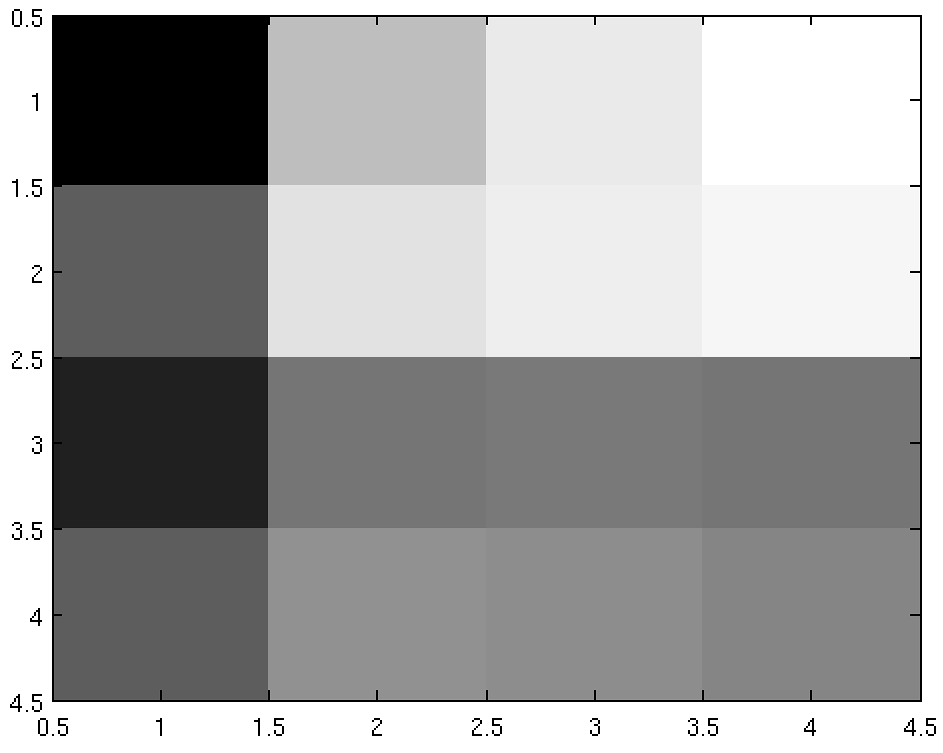
It is possible to remove musical noise by thresholding blocks of STFT coefficients.

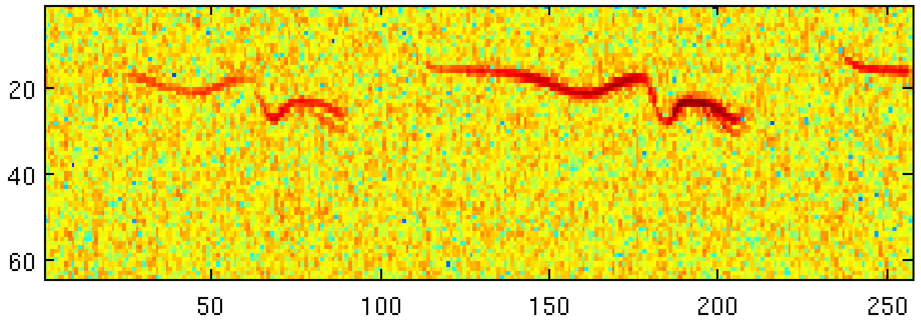
Denoising is performed by block soft thresholding.

```
% perform thresholding
Sn = perform_stft(xn,w,q, options);
SnT = perform_thresholding(Sn, sigma, 'block');
% display the results
subplot(2,1,1);
plot_spectrogram(Sn);
subplot(2,1,2);
plot_spectrogram(SnT);

% Exercice 4: (the solution is exo4.m) Trie for various block sizes and
% report the best results.

snr_block = zeros(4,4);
for k=1:1:4
    for l=1:1:4
        options.block_size = [k, l];
        SnTb = perform_thresholding(Sn, sigma, 'block', options);
        x2b = perform_stft(SnTb,w,q, options);
        snr_block(k,l) = snr(x,x2b);
    end
end

figure();
imagesc(snr_block)
colormap(gray)

[mb, posb] = max(snr_block(:));
[k,l] = ind2sub(size(snr_block),posb);

options.block_size = [k, l];

SnTb = perform_thresholding(Sn, sigma, 'block', options);
x2b = perform_stft(SnTb,w,q, options);

% display the results
figure()
subplot(2,1,1);
plot_spectrogram(Sn);
subplot(2,1,2);
plot_spectrogram(SnTb);
title('Denoise process with optimal block size');

figure()
subplot(2,1,1),plot(xn);
title(sprintf('Señal con ruido, SNR=%.4f dB',snr(x,xn)));
subplot(2,1,2),plot(x2b);
title(sprintf('Señal umbralizada hard, SNR=%.4f dB',snr(x,x2b)));
```

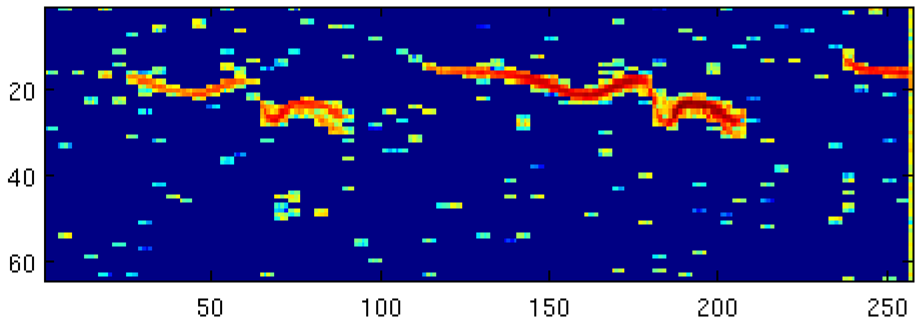

20
40
60
50
100
150
200
250


20
40
60
50
100
150
200
250

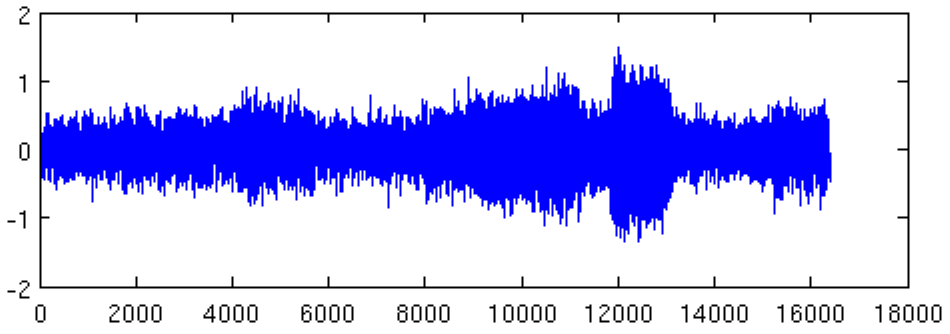

0.5
1
1.5
2
2.5
3
3.5
4
4.5
0.5
1
1.5
2
2.5
3
3.5
4
4.5



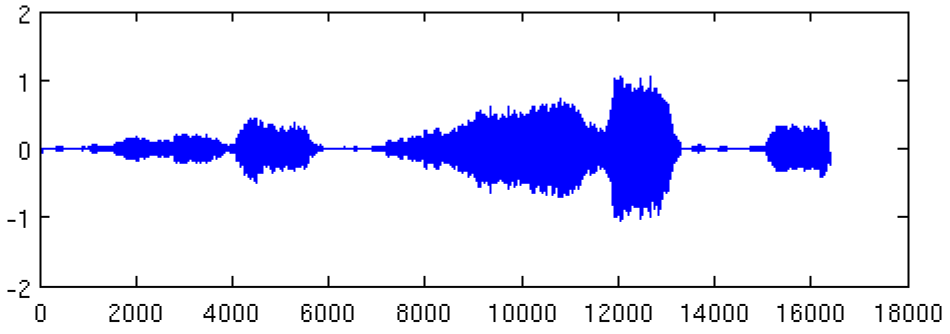
Denoise process with optimal block size



Señal con ruido, SNR=1.7151 dB



Señal umbralizada hard, SNR=14.1052 dB


2
1
0
-1
-2
0
2000
4000
6000
8000
10000
12000
14000
16000
18000