Seizure Detection

Naureen Ghani

December 16, 2017

Time-Frequency Analysis

How does a signal change over time? This question is often answered by using one of the following three methods:

- Apply a Fourier transform with a sliding window
- Use a wavelet transform
- Filter the signal and apply a Hilbert transform

A ***sliding window*** is a segment of data *("window")* within which the computation (often a Fourier transform) is applied. The window slides through the data, repeating the computation until the entire signal is covered. The built-in **spectrogram** function in MATLAB performs a sliding window, and allows the user to vary window lengths and window overlaps.

A ***wavelet transform*** is an alternative to the Fourier transform. This algorithm computes the similarity between each segment of a signal and a short, wave-like distribution called a ***wavelet.*** The wavelet can be scaled across many widths to capture different frequencies.

The ***Hilbert transform*** is often applied to pre-filtered signals. It can be thought of as a *convolution* (using a distribution to transform the signal) that produces a complex number at each point. These complex numbers can be re-interpreted in terms of phases and amplitudes. Thus, the Hilbert transform is an *instantaneous* Fourier transform.

Uncertainty Principle

There are limits to how precisely the frequency spectrum of a signal can be computed. In signal processing, the limiting factor is the length of the signal. Dennis Gabor, inventor of the hologram, was the first to realize that the uncertainty principle applies to signals. The exact time and frequency of a signal can never be known simultaneously: a signal cannot plot as a point on the time-frequency plane. *This uncertainty is a property of signals, not a limitation of mathematics.*

Heisenberg's uncertainty principle is often written in terms of the standard deviation of position σ_{x}, the standard deviation of momentum σ_{p}, and the Planck constant h:

$$\sigma_x \sigma_p \geq \frac{h}{4\pi} \approx 5.3 \times 10^{-35} m^2 kg.s^{-1}$$

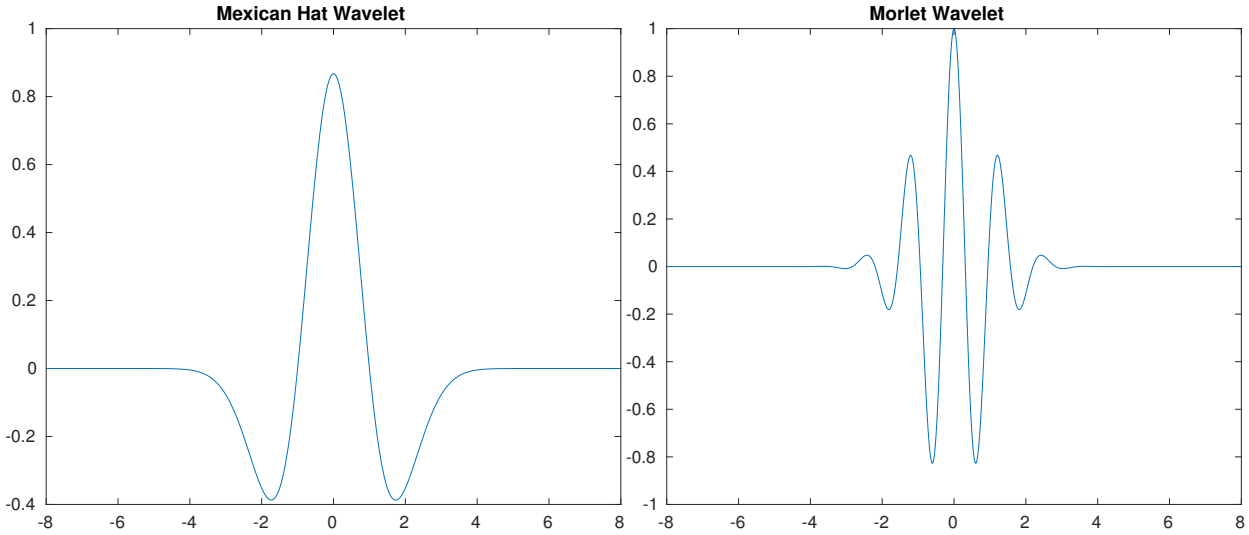
In other words, the product of the uncertainties of position and momentum is small, but not zero. If the standard deviations of the time and frequency estimates are $\sigma_{\mathrm{t}}$and σ_{f}respectively, then we can write ***Gabor's uncertainty principle*** as:

$$\sigma_t \sigma_f \geq \frac{1}{4\pi} \approx 0.8 cycles$$

Thus, the product of the standard deviations of time (ms) and frequency (Hz) must be at least 80 ms-Hz. Regardless of how the transform is computed, we pay for time information with frequency information. Specifically, the product of time uncertainty and frequency uncertainty must be at least $\frac{1}{4\pi}$.

Types of Wavelets

Time-varying frequency components can be identified by filtering a signal with short distibrutions called ***wavelets.*** Wavelets are brief oscillations. When a filter is applied, these wavelets expand or contract to fit the frequencies of interest.



In neurophysiological signal analysis, the Mexican Hat and Morlet wavelets are commonly used. The Morlet wavelet contains a greater number of cycles, which improves the detection of signal oscillations relative to transient events. It also means that we lost time information in order to increase frequency resolution.

One common error in time-frequency analysis is the misattribution of signal events as oscillations. Neural signals appear as *spikes*. Sharp transitions in the signal are built not of a single frequency component, but a vast range of frequencies. Any filter that is applied to the signal over a spike will give the appearance of a power increase at that moment, and the strength of the power inrease is related to the size of the spike. Thus, noise can lead to mistinterpretation of time-frequency results.

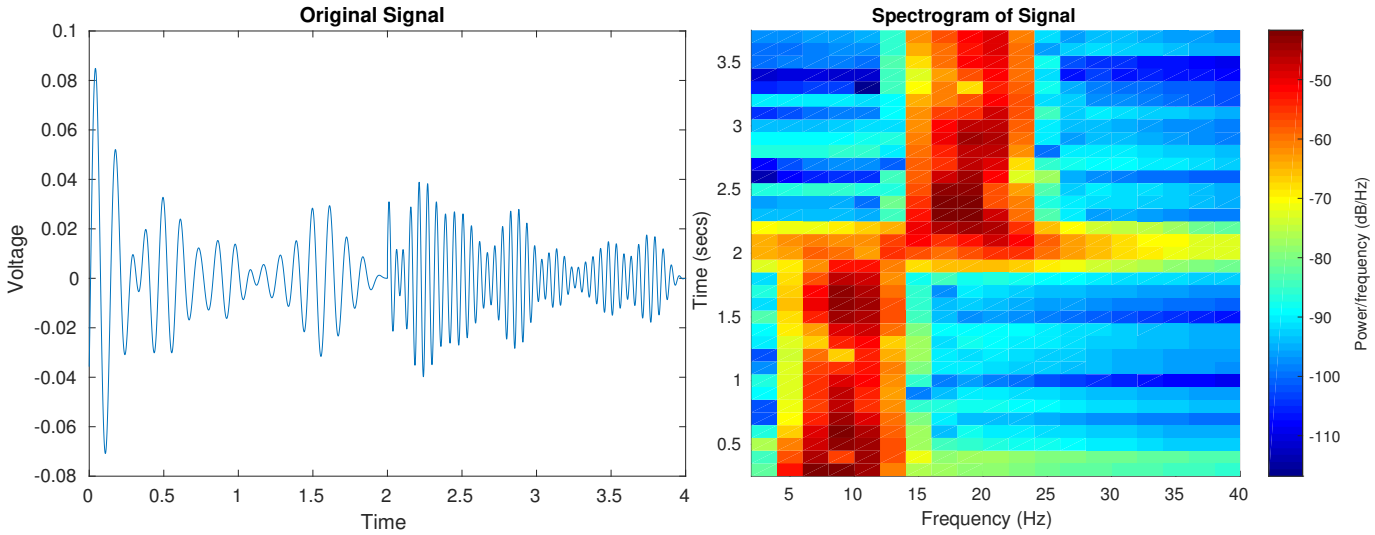
Here is code to pre-filter a signal to remove noise:

```
function fLFP = filt_LFP(sig1,lower_limit,upper_limit,sF)
% fLFP = filt_LFP(sig1,lower_limit,upper_limit)
%
% filt_LFP uses a butterworth filter to bandpass filter the signal between
% lower and upper limit
%
% INPUTS:
% sig1 =         signal to be filtered
% lower_limit = lower bound of bandpass
% upper_limit = upper bound of bandpass
% sF = sampling Frequency (default: 2000 Hz)

% Set Default sF = 2000 Hz
if nargin < 4
    sF = 2000;
end
if isempty(sF)
    sF = 2000;
end

Nyquist_freq = sF/2;
lowcut = lower_limit/Nyquist_freq;
highcut = upper_limit/Nyquist_freq;
filter_order = 3; % may need to be changed based on bandpass limits
passband = [lowcut highcut];
[Bc Ac] = butter(filter_order, passband);
fLFP = filtfilt(Bc,Ac,sig1);
```

Wavelet Demo

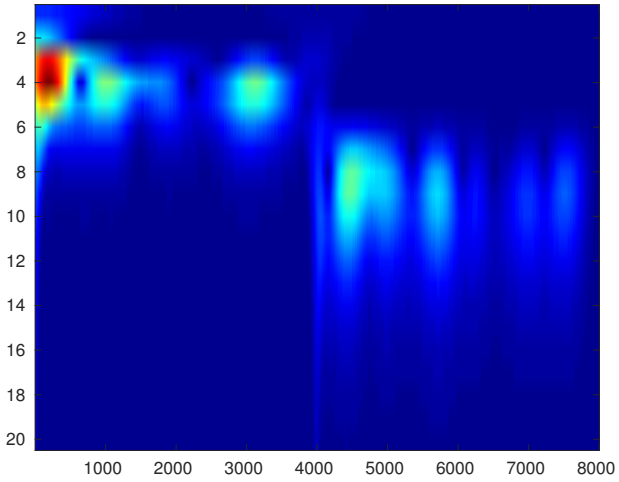
To begin this demo, let's simulate a signal with an 8 to 10 Hz segment followed by a 16 to 20 Hz segment and then apply a sliding window Fourier analysis:

```
% Wavelet Analysis

% Generate a signal with a 8-10 Hz segment
rng(11); % seeds random number generator
noiselfp = rand(1,4001);
LFP_6to10 = filt_LFP(noiselfp,6,10,2000);

% Generate a signal with a 16-20 Hz segment
rng(12);
noiselfp = rand(1,4001);
LFP_16to20 = filt_LFP(noiselfp,16,20,2000);

% Concatenate signals
LFP = [LFP_6to10 LFP_16to20];
figure; plot(linspace(0,4,8002),LFP);
title('Original Signal');
xlabel('Time'); ylabel('Voltage');

% Apply sliding window Fourier analysis
window = 1000;
% the sampling rate is 2000 data points per second
% so the window size represents 1/2 second or 500 ms of data
noverlap = 800;
% amount of overlap from one window to the next. An overlap of 800 samples
% will mean that the window steps over the data in 100 ms increments
% (1000-800 = 200 samples = 100 ms)

F = 2:2:40; % freq to compute spectral data
Fs = 2000;
figure;
spectrogram(LFP,window,noverlap,F,Fs);
view(2)
colormap jet;
title('Spectrogram of Signal');

% Morlet wavelet
coefsi = cwt(LFP,centfrq('cmor1-1')*Fs./F,'cmor1-1');
f2 = figure;
wm_image = imagesc(abs(coefsi));

% Mexican hat wavelet
coefsi_hat = cwt(LFP,centfrq('mexh')*Fs./[2:40],'mexh');
imagesc(abs(coefsi_hat));
```

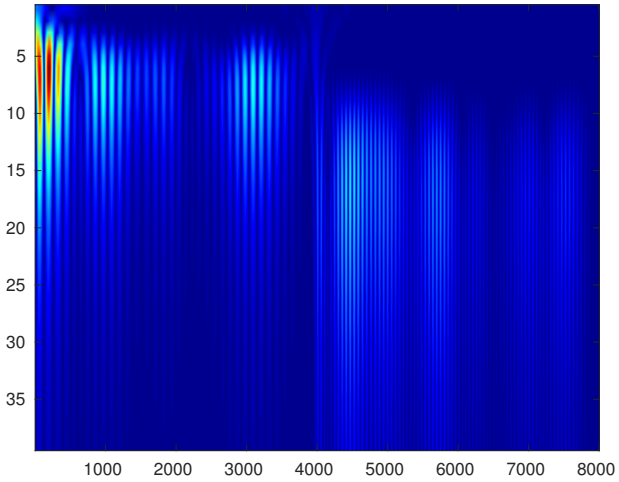


There are two approaches to applying wavelet transformations in MATLAB: discrete and continuous. These differ according to how the set of frequencies (wavelet width) are chosen.

In this exercise, we compute the complex Morlet wavelet. We can obtain meaningful values of amplitude and phase in this way.



We repeat this process for the Mexican hat wavelet. The built-in MATLAB function does not use complex conjugates, so we will only detect upward deflections in the LFP signal at each frequency band.

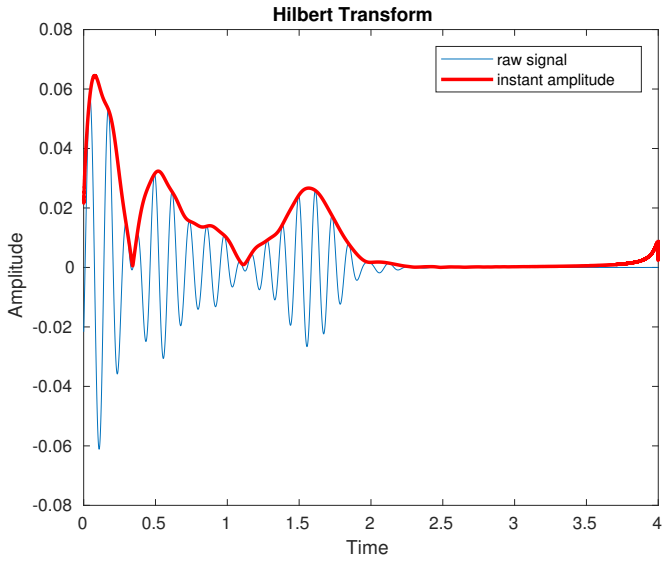


In this image, we see stripes where we previously saw smooth transitions across time bins. The warmer-colored streaks also extend farther across the frequency spectrum, while the resolution appears to be stronger in the temporal domain. This illustrates the *uncertainty principle* described above.

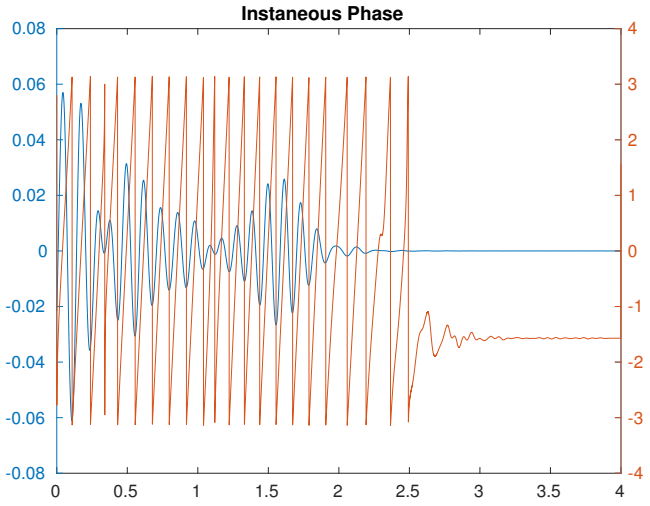
Hilbert Transform

The Hilbert transform is used to compute the instantaneous phase and amplitude of a signal. It is best to use pre-filtered data. Here is the code to do so:

```
% Compute instantaneous phase and amplitude of a signal using Hilbert
% transform

LFP_6to10post = filt_LFP(LFP,6,10,2000);
h_LFP_6to10post = hilbert(LFP_6to10post);
amp_LFP_6to10post = abs(h_LFP_6to10post);

figure;
hold on;
plot(linspace(0,4,8002),LFP_6to10post);
plot(linspace(0,4,8002),amp_LFP_6to10post,'r','LineWidth',2);
legend('raw signal','instant amplitude');
xlabel('Time'); ylabel('Amplitude');
title('Instaneous Amplitude');
hold off;

figure;
phs = atan2(imag(h_LFP_6to10post),real(h_LFP_6to10post));
plotyy(linspace(0,4,8002), LFP_6to10post, linspace(0,4,8002), phs);
title('Instaneous Phase');
```



The red line indicates our instanteous amplitude, which *envelopes* the filtered signal. We could alternatively use the complex conjugate to identify the instantaneous phase of the oscillation in the filtered signal:



Seizure Simulation

Epilepsy is a chronic neurological disorder characterized by recurrent seizures. In children, many seizures display rhythmin ***spike-wave (SW) discharges*** in the EEG. In a study published by *PLoS One* (*Taylor and Wang, 2014*), a computational model of seizures was built on MATLAB. In this demo, we will explore their simulation:

```
function dudt=AmariTCimpBS(t,u)
%
    %connectivity parameters
    w1 =1.8;%PY  -> PY
    w2 = 4;    %PY -> I
    w3 = 1.5; % I -| PY
    % w4= 0;
    w5 = 10.5; %  TC -> RTN
    w6 = 0.6; % RTN -| TC
    w7 = 3; % PY -> TC
    w8 = 3; %   PY -> RTN
    w9 = 1; %  TC -> PY
    w10= 0.2;% RTN -| RTN

   h_p = -.35; %
   h_i = -3.4; %
   h_t = -2; %    -2 for bistable for ode45; -2.2 for excitable for ode45.
   h_r = -5; %
   s=2.8;
    a=1;
   % Time scale parameters
   tau1=1*26;    %
   tau2=1.25*26;    %
   tau3=.1*a*26;    %
   tau4=.1*a*26;    %



   sig_py = (1./(1+250000.^-(u(1))));
   sig_in = (1./(1+250000.^-(u(2))));
   sig_tc = (1./(1+250000.^-(u(3))));
   sig_re = (1./(1+250000.^-(u(4))));

   dudt=zeros(4,1);
```

```
    dudt(1) = (+h_p -u(1) +w1*sig_py -w3*sig_in + w9*sig_tc )*tau1;
    dudt(2) = (+h_i -u(2) +w2*sig_py                          )*tau2;
    dudt(3) = (+h_t -u(3)  +w7.*sig_py - w6*(s*u(4)+.5)       )*tau3;
    dudt(4)= (+h_r -u(4) +w8.*sig_py + w5*(s*u(3)+.5) -w10*(s*u(4)+.5))*tau4;



end
```

The model describes the temporal evolution of the state of four variables corresponding to the activity of populations of:

- cortical pyramidal neurons (PY)
- cortical inhibitory interneurons (IN)
- thalamo-cortical neurons (TC)
- inhibitory (thalamic) reticular neurons (RE)

There is a background state of normal activity and a rhythmic state of pathalogical activity (SW complex). We can then modify their code to detect peaks by thresholding:

```
%initial condition near the fixed point
fp_approx = [0.1724    0.1787   -0.0818    0.2775];


[t1,u]=ode45(@AmariTCimpBS,[0 10],fp_approx);%background state
[t2,v]=ode45(@AmariTCimpBS,[10 15],u(end,:)-[.3 .3 0 0]);%seizure state
[t3,w]=ode45(@AmariTCimpBS,[15 30],v(end,:)-[.3 .3 0 0]);%background state
%%


PY=[u(:,1); v(2:end,1); w(2:end,1)];
IN=[u(:,2); v(2:end,2); w(2:end,2)];

TC=[u(:,3); v(2:end,3); w(2:end,3)];
RE=[u(:,4); v(2:end,4); w(2:end,4)];

t=[t1;t2(2:end);t3(2:end)];

figure(1)
plot(t,mean([PY,IN],2),'k')
hold on
m=mean(fp_approx(1:2));
% plot([10 10],[m m-.3],'r','LineWidth',5)
% plot([15 15],[m m-.3],'b','LineWidth',5)
hold off
xlabel('Time (sec)','FontSize',20)
ylabel('Simulated EEG','FontSize',20)
set(gca,'FontSize',15)
%legend('Simulated EEG','Stimulus pulse to induce SWD','Stimulus pulse to terminate
    SWD')

% Use built-in peak detection
time = t;
volt = mean([PY,IN],2);
pks = findpeaks(volt,0.3); % threshold for peaks > 0.3
pks = pks.loc; % convert struct to double
hold on;
scatter(t(pks),volt(pks),'r');
```

```
legend('raw signal','spikes');
title('Simulated Seizure');
hold off;
```