iEEG Cleaning Pipeline

@UNICOG, NeuroSpin

The 4 main Steps for cleaning the iEEG data

1. Detection of hardware artifacts (saturated noise, unplugged electrodes etc) via median thresholding.

2. Detection of Spiking channels.

3. Detection of deviant channels from the Power-Spectrum.

4. Detection of pathological channels based on the detection of HFOs.

General pre-processing steps

1. Downsampling of the data (most iEEG recording systems record at an extremely high sampling rate ~2KHz. It is advised to downsample your data at 1KHz to ease processing and memory allocation).

2. Removal of line-noise and harmonics using a notch filter (unless you're interested in fast-Ripple detection research (>250Hz), a removal of the two first harmonics will be sufficient).

3. Linear-detrending of the data.

4. Prior to analyzing, cut the continuous data into pieces to ease resources allocation.

```
% Pre-processing : This is the pre-processing pipeline for the intracranial
% data. The goal of this scrip is to be as generic as possible, integrating
% data from various research centers such a: the Houston medical center,
% the Marseille research center etc.
% Written by : Christos-Nikolaos Zacharopoulos @UNICOG 2018
%            christonik@gmail.com
% based on a similar pipeline used at the Stanford University.
% This is a plug-and-play function, to change the research center under
% consideration, provide it as a pair-input in the command window.
% e.g : runPreprocessing('Hospital',{'Houston'})


function runPreprocessing(varargin)
% ---------- BRANCH 1 - SET THE PATHS ---------- %
% In this branch we define the paths for the user.
clc;  close all;
addpath(genpath('functions'));
% Get the data path based on the hostname of the computer in use. The
% variables are unaffected from the OS.
[~,hard_drive_path, elecs_path] = getCore;
% Check whether the parallel computing toolbox is installed - if yes get
% the default number of available cores.
if license('test', 'Distrib_Computing_Toolbox')
   % Get the number of default workers
   numCores = feature('numcores');
   % Open local cluster
   parpool(numCores);
end
% ---------- BRANCH 2 - SPECIFY THE RESEARCH CENTER ---------- %
% In this branch we manunally add the list of patients that corresponds to each
% individual project.
P = parsePairs(varargin);
checkField(P,'processing','quick');
% The other available option is 'slow'/'quick'.
% quick : minimaze file I/O  and visualization to save computing time.
checkField(P,'Hospital',{'Houston'});
```

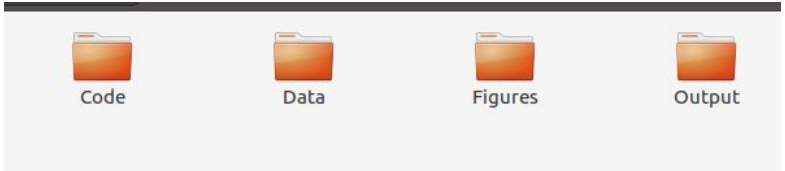
This is a function with default settings. Every input in this function comes in pairs (this is where the function parsePairs is used.) This is done to increase human readability when it comes to the inputs. For example, to change the processing option, type :

runPreprocessing(‘processing’,’slow’)

nPreprocessing.m | getCore.m | +

```
  [~, name] = system('hostname');
  % Get the Hostname (Computer ID)
  name = strip(name);
  % Christos - Windows PC
  if strcmp(name,'DESKTOP-4ALPTJB')
    hard_drive_path = fullfile('F:', filesep);
    script_path = fullfile('C:','Projects' );
    elecs_path = fullfile(hard_drive_path, 'NeuroSyntax2','Data','Houston');
    format compact
    format shortG


    % Fanis - Workstation


  end

elseif isunix == 1
    % Here, distinguish between multiple UNIX machines
    [~, name] = system('hostname');
    % Get the Hostname (Computer ID)
    name = strip(name);
    if strcmp(name, 'is154105')
        % Christos - CEA laptop
        script_path = fullfile(filesep,'home', 'czacharo','Projects');
        hard_drive_path = fullfile(filesep,'media','czacharo','Transcend');
        elecs_path = fullfile(hard_drive_path, 'NeuroSyntax2','Data','Houston');
        format compact
        format shortG
    elseif strcmp(name, 'is150940')
        % Christos workstation - NeuroSpin
        script_path = fullfile(filesep,'home', 'czacharo','Projects');
        hard_drive_path = fullfile(filesep, 'neurospin','unicog', 'protocols', 'intracranial');
        elecs_path = fullfile(hard_drive_path, 'NeuroSyntax2','Data','Houston');
        format compact
        format shortG
        % Fosca - Workstation



        % Fernanda - linux Laptop
    end
end

% Add the path to the load_settings_params function
addpath(fullfile(script_path,'Core'));
```

The “getCore” function is used to set the paths. This function is build to set the paths irrelevant of the OS used by the user. Moreover, the user can also specify the hostname of the PC in use, and the paths will be automatically set as long as the file-tree where the code is stored has the following structure:



Once the user specifies the hostname and set the “core” path that leads to the above tree (this can be a hard-drive or the server) the paths are set automatically.

That makes it easy to git-push and work between different people, OSs and PCs under the same OS.

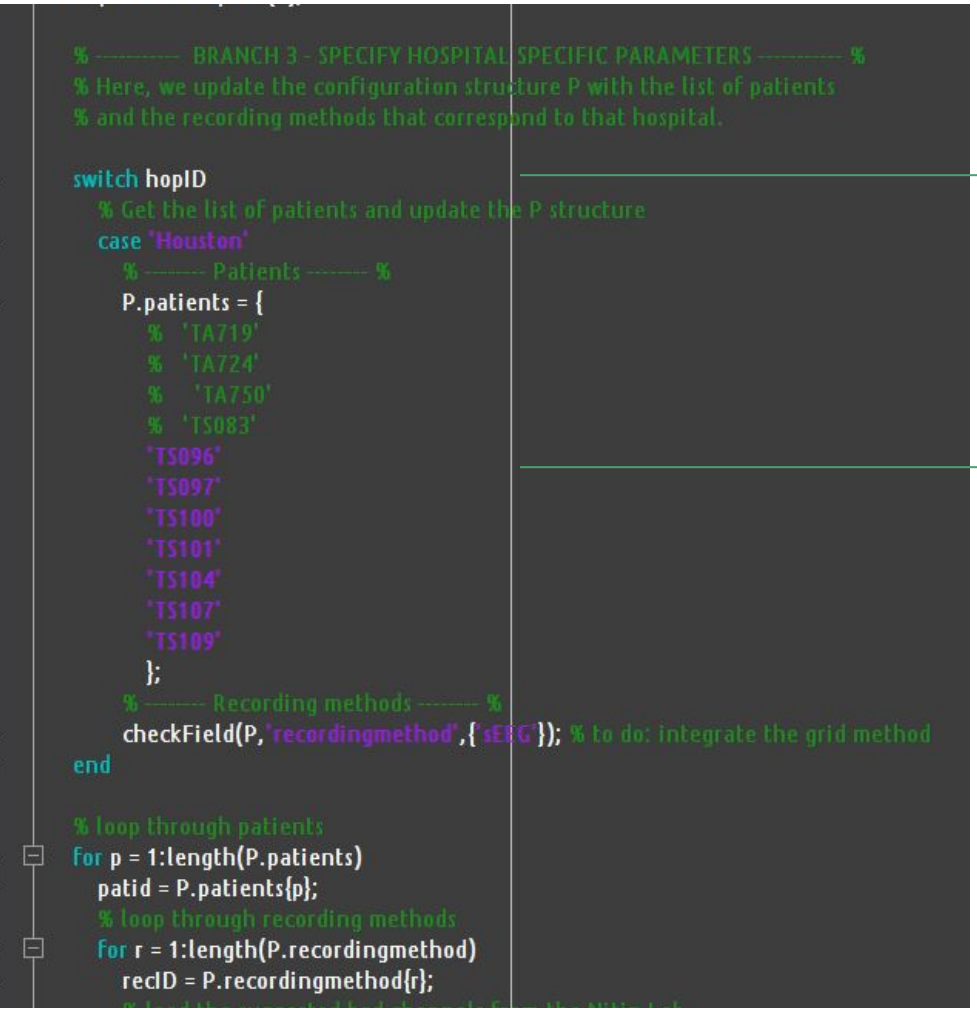
```
% Check whether the parallel computing toolbox is installed - if yes get
% the default number of available cores.
if license('test', 'Distrib_Computing_Toolbox')
    % Get the number of default workers
    numCores = feature('numcores');
    % Open local cluster
    parpool(numCores);
end
```

Automatically detect whether the parallel computing toolbox is installed. If so, soft-code the number of workers and open the local cluster.

```
% ---------- BRANCH 2 - SPECIFY THE RESEARCH CENTER ---------- %
% In this branch we manunally add the list of patients that corresponds to each
% individual project.
P = parsePairs(varargin);
checkField(P,'processing','quick');
% The other available option is 'slow'/'quick'.
% quick : minimaze file I/O  and visualization to save computing time.
checkField(P,'Hospital',{'Houston'});
```

From this point onward, we start to build the configuration structure. This will include the high-level parameters such as the Hospital where we analyzing the data from, the type of processing, etc.

The option (‘processing’,’slow’) allows for visual inspection of the rejected channels at each step. Also, it saves the data from every step on the data path. It is recommended especially for the first time that you run the analysis on a new patient. The option ‘quick’ only informs the user on the number and location of the rejected channels, without showing them.



"hopID" stands for "Hospital ID"

Here, we specify the list of patients per hospital. We can either select a single patient or loop through all patients.

The processing for the Grid electrodes has not yet been implemented.

```
        % ---------- BRANCH 4 - EPOCHING OF CONTINUOUS DATA---------- %
        % This epoching is irrelevant of any condition. This is just
        % chuncking of continuous data to ease processing and avoid memory
        % errors.

        epochs = epochContinuousdata(raw_data,params);

        % ----------  BRANCH 5 - MAIN  CHANNEL REJECTION ANALYSIS---------- %
        % Here, we enter the main pipeline for the channel rejection
        % with specified inputs for the selected Hospital.
        cleandata = cell(size(epochs,1),1);

        % loop through the epochs
        for epoch = 1:size(epochs,1)
           cleandata{epoch} =  badChannelsRejection(P,settings,epochs(:,:,epoch),params, labels, gyri,hopID, epoch);
        end

    end
  end
end
```

```
Loading raw data - Patient TS097.
Recordings : 2.
Hospital : Houston

The data have been loaded.


The data from all recordings (if multiple),
have been loaded and concatenated into a single variable.

---- Epoching continuous data ----

Creating  epochs of 10 minutes.
```

```
Epoching completed, 4 epochs were created.
Removing line noise and harmonics from all channels : 35%    [...     ]
```

At each step of the process, the user get feedback on the command window.

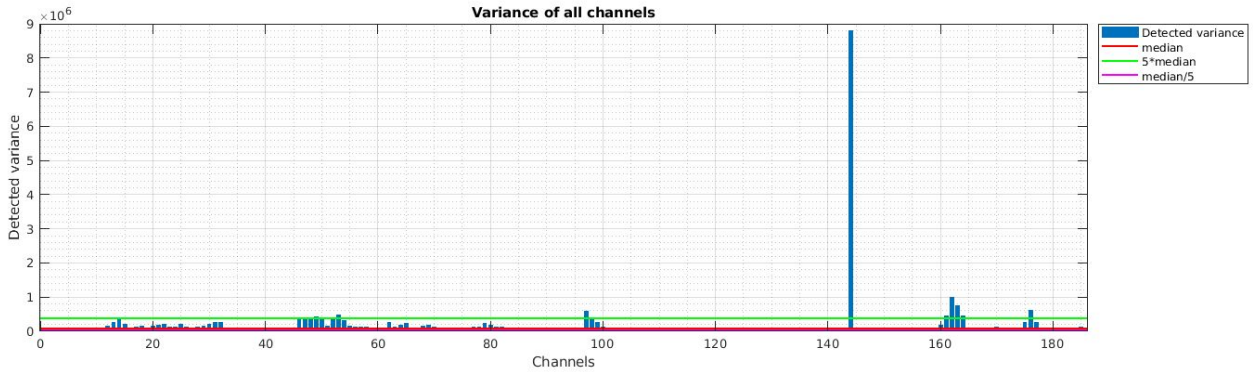
```
switch hopID
  case 'Houston'
    % ---------- STEP 0 ---------- %
    % Non-pathological cleaning steps

    % Create a channel log-file. This will be a logical array where 1
    % will denote a good channel and 0 will denote a bad channel.

    % Initialize variable to hold the filtered data
    channels = size(raw_data,1);
    % Get the duration of the recording
    duration = size(raw_data,2);

    filtered_data = zeros(channels-1,duration);
    % Initialize a logical array where we assume all channels to be
    % good
    allchannels = true(channels-1,1);


    timecount = linspace(1,100,size(filtered_data,1));
    close all;
    textprogressbar('Removing line noise and harmonics from all channels : ');
    % Filter the line noise and the harmonics
    for channel = 1:(channels-1)
      % Do not include the trigger channel
      textprogressbar(timecount(channel));
      % Downsample the data to 1kHz
      wave = downsample(raw_data(channel,:),1,0);
      [wave]= notch(wave, params.srate, 59, 61,1);
      [wave]= notch(wave, params.srate, 118,122,1); % Second harmonic of 60
      [wave]= notch(wave, params.srate, 178,182,1); % Third harmonic of 60

      filtered_data(channel,:) = wave;

    end


    % Remove the trigger channel labels
    labels(end) = [];
    gyri(end) = [];
```

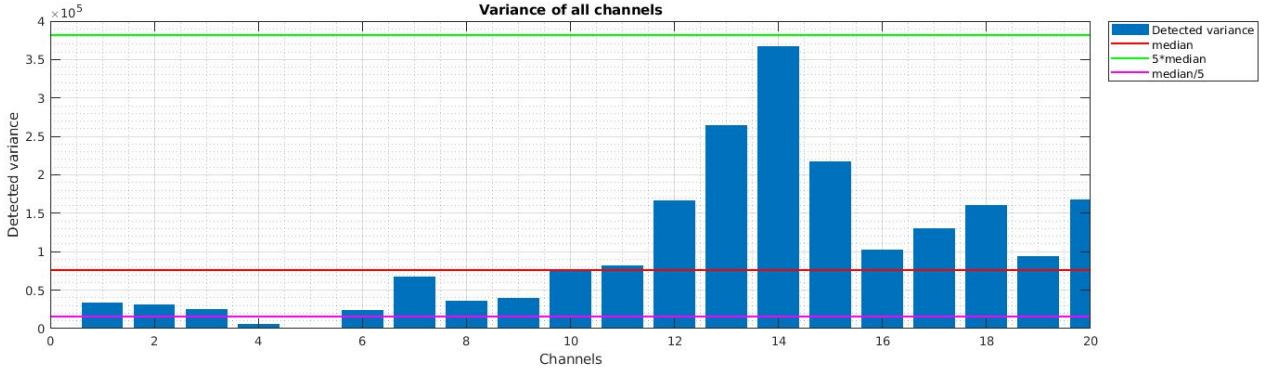
We initialize a channel logical array (in Houston, the trigger is on the last channel, so we exclude it here.)

1 indicates a good channel
0 indicates a bad channel

Filter with the notch filter and downsample here. At this point you can also change the bandwidth of the filter. In the future this will be automated based on where the data come from (Europe or otherwise)

Step 1 : Median thresholding (Rejection based on raw power)





We get the variance of all channels. This returns al. row matrix of dimensions (1 x channels) – we have a single variance value per channel (σ^2/channel).

We then threshold and exclude those channels that exceed an upper and a lower threshold.

Upper threshold = 5*median(var(all_channels))

Lower threshold = median(var(all_channels))/5

```
In total 16 have been removed based on the variance of the all channels.
 The channels have the following labels :

    "LIN4"
    "LIN5"
    "AER3"
    "TP2"
    "TP3"
    "TP4"
    "TP5"
    "TP6"
    "TP8"
    "TP9"
    "TOP1"
    "AH4"
    "AH5"
    "AH6"
    "AH7"
    "PH5"

and are located in the following regions :
    'S_occipito-temporal_lateral'
    'S_occipito-temporal_lateral'
    'G_occipit-temp_med-Parahippocampa…'
    'Pole_temporal'
    'Pole_temporal'
    'Pole_temporal'
    'G_temp_sup-Planum_polare'
    'G_temp_sup-Planum_polare'
    'G_temp_sup-Planum_polare'
    'G_temp_sup-Lateral_aspect'
    'G_temp_sup-Lateral_aspect'
    'Medial_wall'
    'Medial_wall'
    'Medial_wall'
    'Medial_wall'
    'Medial_wall'

Detecting spiking channels.
100%    [.........]
Detection completed.
In total 6 have been removed due to spiking activity.
 The channels have the following labels :
    "AER1"
    "AER2"
    "MBT6"
    "AH3"
    "PH4"
    "PH6"
and are located in the following regions :
    'G_occipit-temp_med-Parahippocampa…'
    'G_occipit-temp_med-Parahippocampa…'
    'G_temporal_middle'
    'G_occipit-temp_med-Parahippocampa…'
    'Medial_wall'
    'Medial_wall'
So far,  22 channels have been rejected out of the total 186.

16 of them have been rejected based on raw power
6 due to detected spiking activity.
```

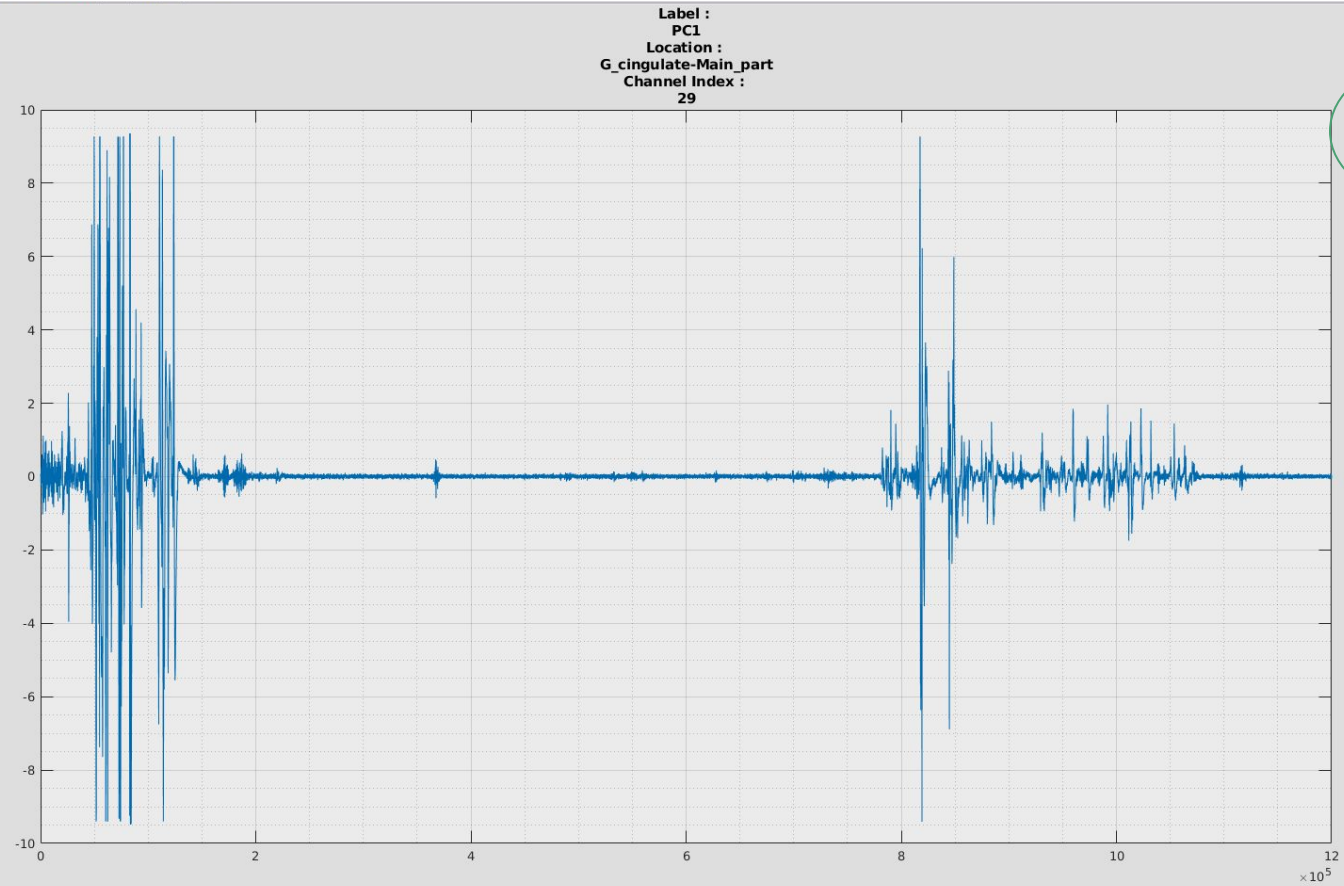
The user receives feedback at each stage of the processing. Here, we see that 16 channels were rejected on the first stage.

The labels of the discarded channels

The locations of the discarded channels

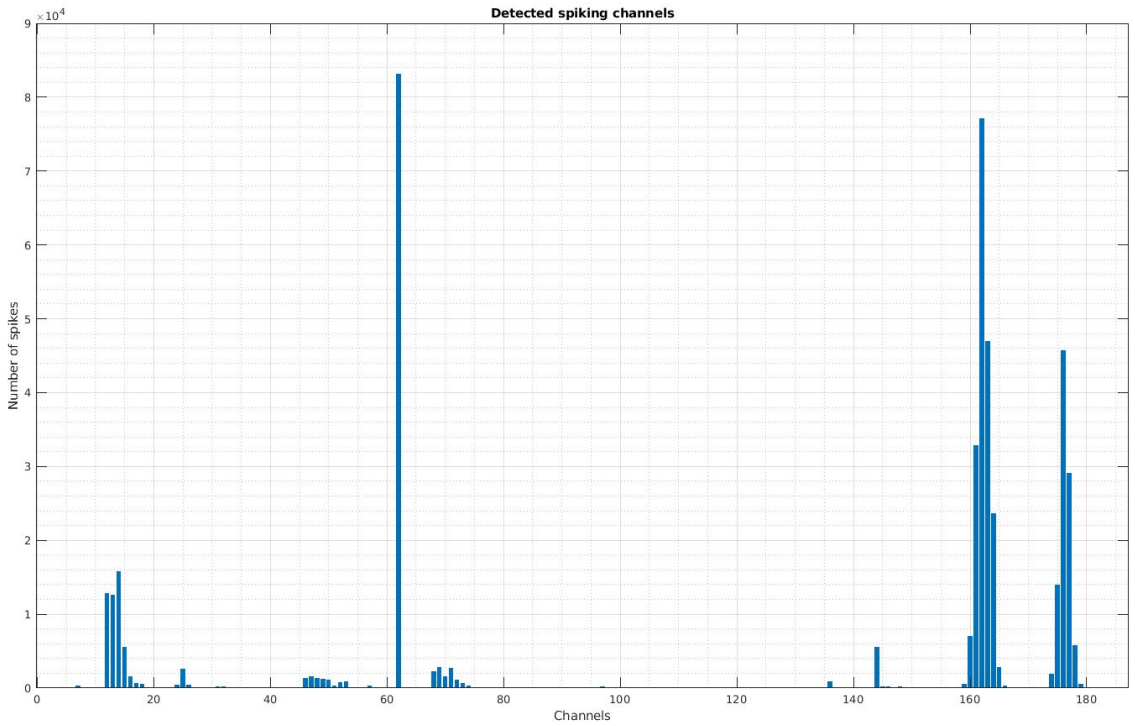
The same holds for each step. In step n.2, 6 channels were rejected

Example of a discarded channel from stage 1.



Current channel and total number of discarded channels in this stage.

Step 2 : Detection of Spiking channels

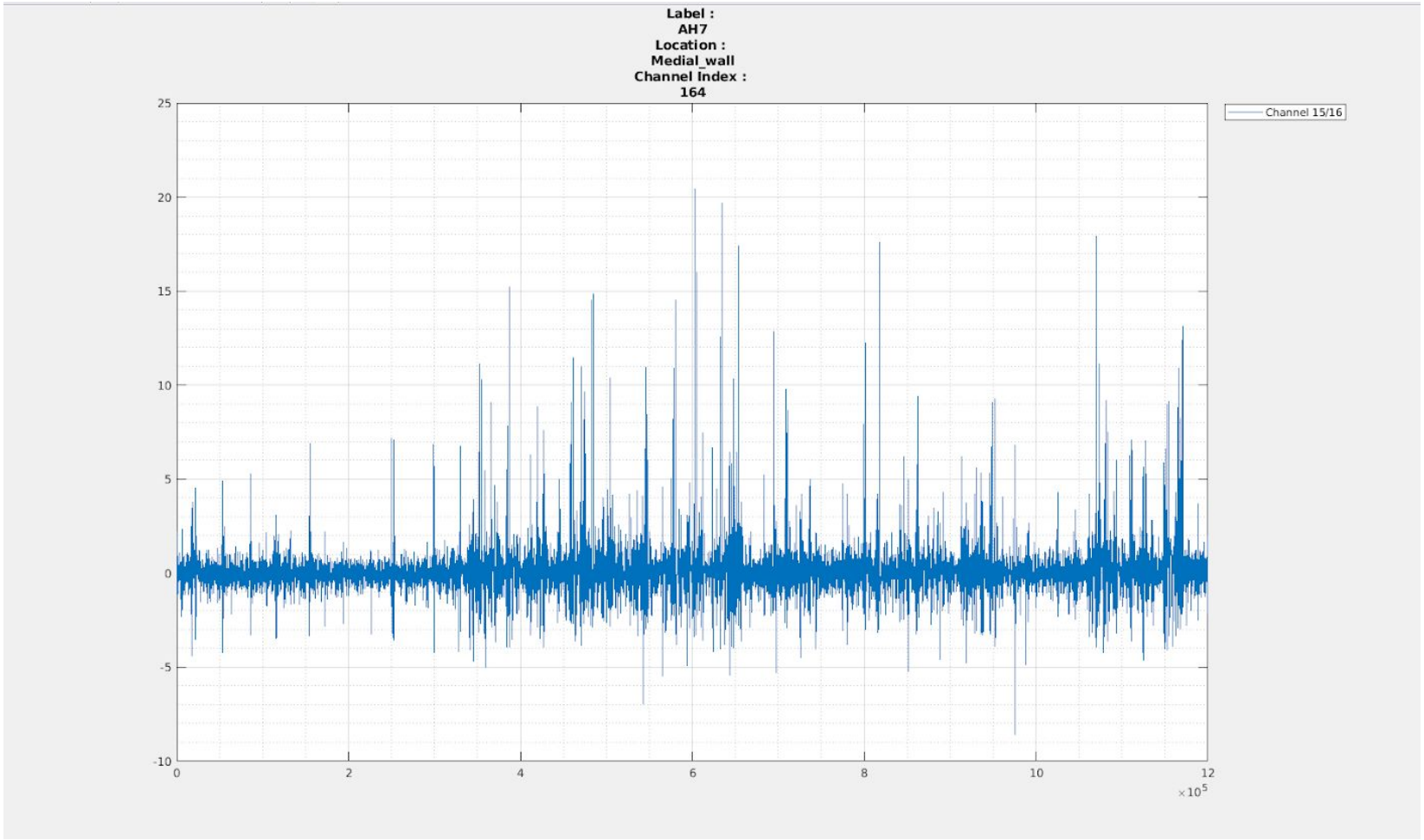


The goal here is to detect rapid changes in the signal (“jumps”).

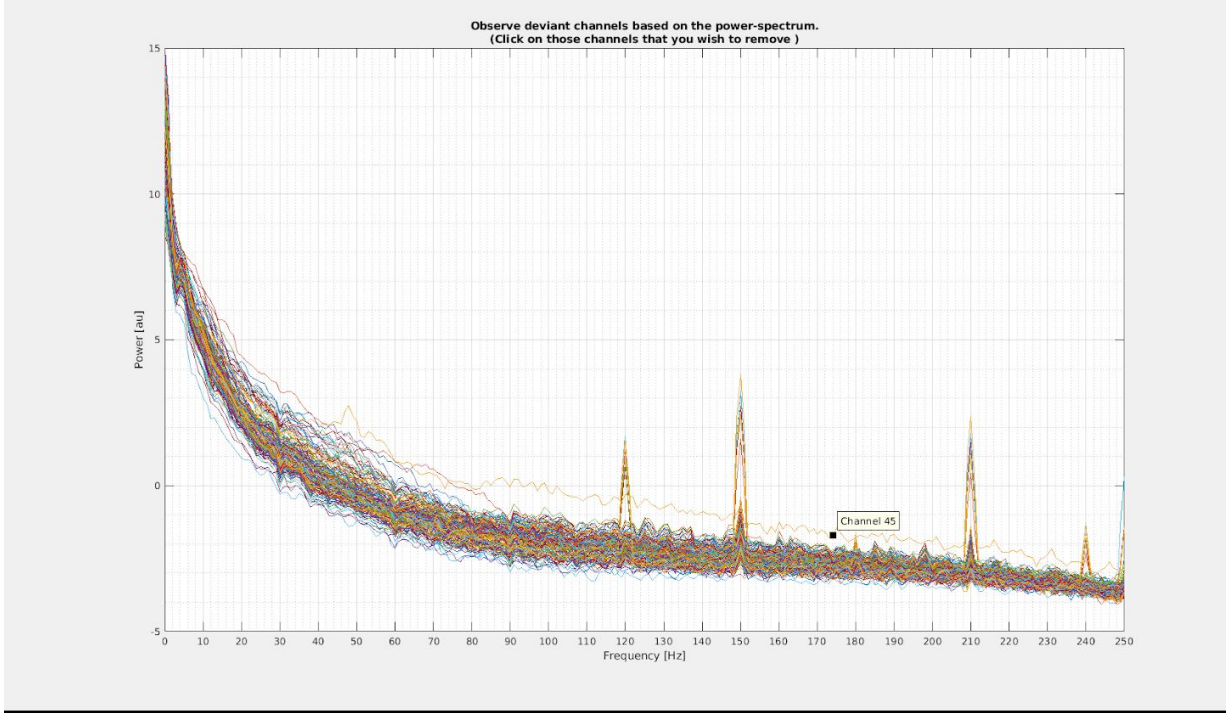
Initially, we set a threshold in mV (80 mV).

We loop through each individual channel and get the difference of two successive points. If this difference exceeds the provided threshold, we call that a spike and
we register the spiking event on the channel.

Example of a discarded channel from stage 2.



Step 3 : Detection of deviant channels from the Power - Spectrum

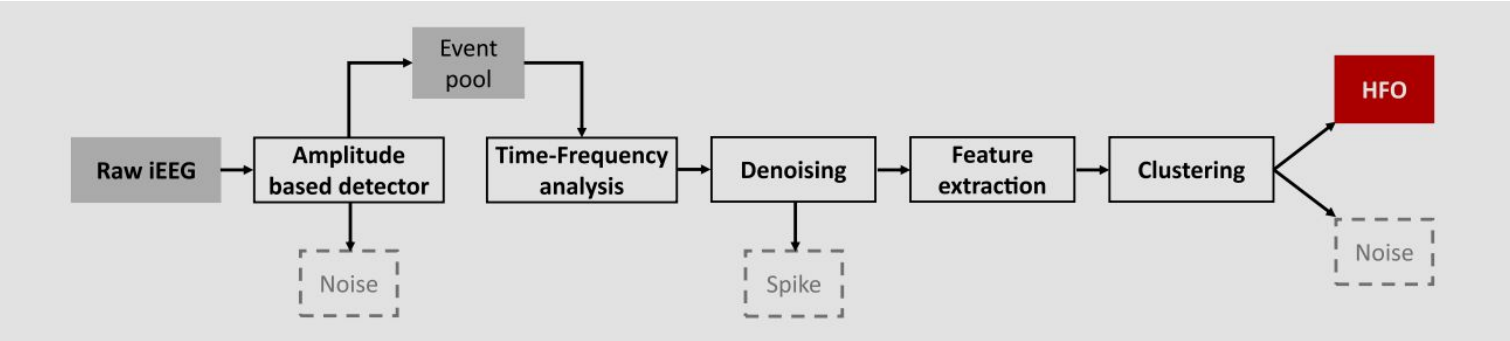


A callback function allows the user to see the index of the deviant channel (in that case, 45).

The user can then update the rejected channels on the command window

```
Calculating the Welch's Power Spectral Density
100%    [..........]
Estimation completed
Do you want to add channels for rejection? Y/N :
```

Step 4 : Rejection of channels based on the presence of HFOs



doi:10.1093/brain/awx374 BRAIN 2018: 141; 713–730 | 713

BRAIN
A JOURNAL OF NEUROLOGY

Stereotyped high-frequency oscillations discriminate seizure onset zones and critical functional cortex in focal epilepsy

Su Liu,[1] Candan Gurses,[2] Zhiyi Sha,[3] Michael M. Quach,[4] Altay Sencer,[5] Nerses Bebek,[2] Daniel J. Curry,[6] Sujit Prabhu,[7] Sudhakar Tummala,[7] Thomas R. Henry[3] and Nuri F. Ince[1]

PAPER

Exploring the time–frequency content of high frequency oscillations for automated identification of seizure onset zone in epilepsy

To cite this article: Su Liu *et al* 2016 *J. Neural Eng.* **13** 026026

Concluding notes

- The code is not fully vectorized, yet for the largest part, it is.
- To make this code as generic as possible, I've identified several hot-spots where a memory error can occur. At those points, I tried to provide alternative solutions (slower). This code has been tested in Windows (i7,16gb ddr3), Linux (i7,16gb ddr3), Linux (i7,32gb ddr3) and two matlab versions (2017b, 2018a)

```
catch ('Out of memory. Type HELP MEMORY for your options');
  disp(['The available RAM limit has been reached. ' newline ...
    'Trying to calculate the variance for each channel manually.'])
  % Pre-allocate the variance variable
  dataVariance = zeros(1,size(filtered_data,1));
  for channel = 1: size(filtered_data,1)
    dataVariance(1,channel) = var(filtered_data(channel,:)');
  end
  disp(['The calculation has been completed'])
  switch P.processing
    case 'slow'
      figureDim = [0 0 1 1];
```

- At the end of the pipeline, the user can eyeball the discarded channels, where different colors indicated rejection in different stages.
    - Stage 1 : blue
    - Stage 2 : black
    - Stage 3 : magenda
    - Stage 4 : red
- After applying CAR (Common Average Re-referencing), the user can eyeball the non-rejected channels
- Up to this point, the user has to mentally-note the channels that he/she wants to keep or reject after the visual inspection and manually add them at the editor. I will soon implement an interactive way of doing that from the command window.
- Experiment with the thresholds and find a “Hospital-specific” list of thresholds for each stage.
- So far, this procedure has been implemented at the channel level. I will also implement it at the epoch level.

Example of a non-discarded channel.