ECOG PROCESSING WITH SPM

A pre-processing pipeline for intracranial EEG

Abstract

This work presents a pre-processing pipeline for ElectroCorticoGraphy (ECoG) recordings, in Matlab. It uses SPM batches and routines, as well as in-house code and FASST tools.

Laboratory of Behavioral and Cognitive Neurosience, Stanford

Jessica Schrouff, Dominique Alessi

Foreword

This tutorial describes all the steps to perform pre-processing and univariate analysis of ECoG data in Matlab, using SPM (http://www.fil.ion.ucl.ac.uk/spm/), FASST (http://www.montefiore.ulg.ac.be/~phillips/FASST.html) and in-house routines. It provides code to perform two types of analyses:

1. Event-related analysis: similar as for EEG, this pipeline performs the classic steps, with additional freedom in terms of epoching, especially for response locked events.
2. Continuous analysis: Ideal for connectivity analyses or analysis of rest/sleep sessions. The steps involved are similar but require different baseline corrections.

The tutorial is divided in multiple parts, explaining the different routines and steps performed, how they can be combined in pre-processing pipelines and what parameters can/should be modified. Our lab has a special interest in High Frequency Broadband (HFB, 70-180Hz) power, so the final outputs are HFB signals. However, the steps can be combined in other ways to perform e.g. ERP in raw signal or phase-amplitude coupling analyses. All along, the SPM format is maintained.

Credits

This tutorial and all code presented was written at the Laboratory for Behavioral and Cognitive Neuroscience. It can be found on Github (https://github.com/JessicaSchrouff/ECoG_preprocessing_SPM) and is open-source. Please do mention our lab as well as the Github url in publications.

Installation

To use the routines, simply download the code from the Github. You then need to add the main folder to the Matlab path, as well as each sub-folder. Please do NOT 'add with subfolders' as this might create conflicts between FASST and SPM. This should be fixed in future versions. Section 0 details how to set up the Matlab environment.

To keep in mind

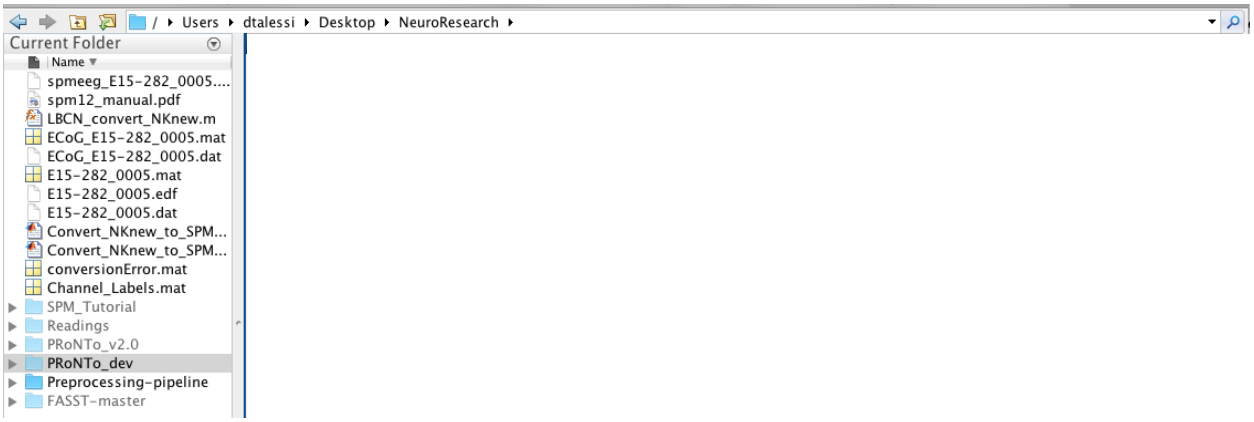
ECoG data is highly variable in terms of format and quality, probably more so than EEG or fMRI. The pipelines provided here are designed as starting points for your analyses and should probably not be used as such. All parameters can easily be changed in the batch (see Editing the batch) or directly in the function calls. It is also important to visually check your processed data regularly, as quality control. We present different options to do so.

Step 0: Setting up the environment

If you're familiar with SPM and Matlab, you can skip this section.

Configuring MATLAB

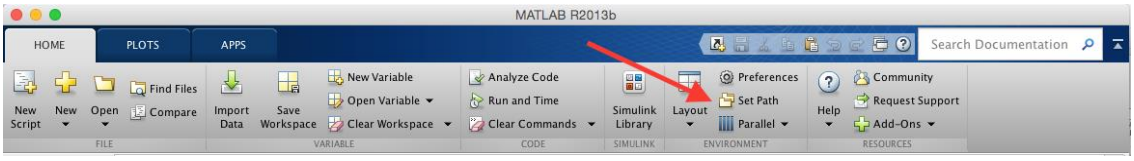
In order to use SPM, you will need to launch MATLAB and find the directory you wish to use. It is easiest to work in the folder containing your data. Once MATLAB is open, you can use the top bar and the left side panel to navigate folders/directories:



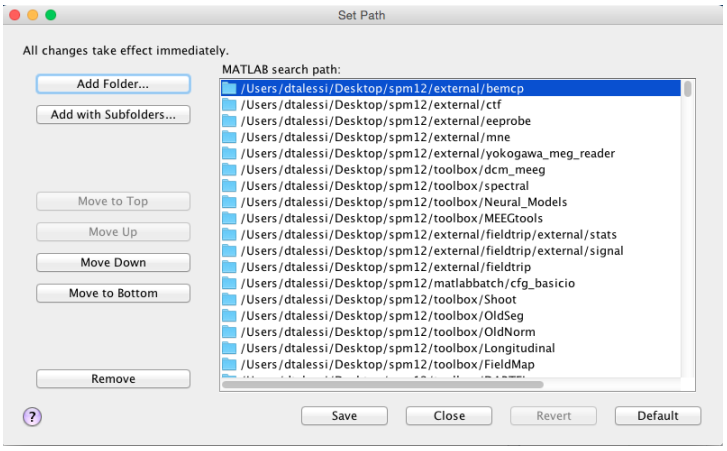
You need to download and install SPM12. If this is your first time using SPM and/or the Preprocessing pipeline, you will want to add the SPM and "Preprocessing-pipeline" folders to your MATLAB path. Specifically, your path should include the following folders:

- spm12
- Preprocessing-pipeline
- Preprocessing-pipeline/utils
- Preprocessing-pipeline/anonymize
- Preprocessing-pipeline/FASST_tools

To modify your path, click on the "Set Path" button in the MATLAB toolbar:



This will bring up a file selection window, where you can add new directories by clicking the "Add Folder…" button.



For our purposes, you should not use the "Add with Subfolders…" option; some of the FASST subfolders contain functions that would overwrite SPM or MATLAB functions should they be included in the path, and these FASST functions may not be compatible with our needs.

Calling functions and scripts in MATLAB

MATLAB functions

In MATLAB, .m files that begin with the word "function" are considered functions. The name of the function is the same as the name of the file, without the .m at the end. Functions may take parameters and/or return some result.

Ordering of parameters

If a function takes parameters, you must specify the parameters in order. For example, if you would like to call a function called **my_function**, which takes in as parameters a filename, an integer, and an array, in that order, you would type

```
>> my_function('myfile', 3, [1 2 3])
```

at the MATLAB command line to invoke **my_function** with parameters 'myfile', 3, and [1 2 3]. Note that strings, including filenames, should be enclosed in single quotes.

Optional parameters and defaults

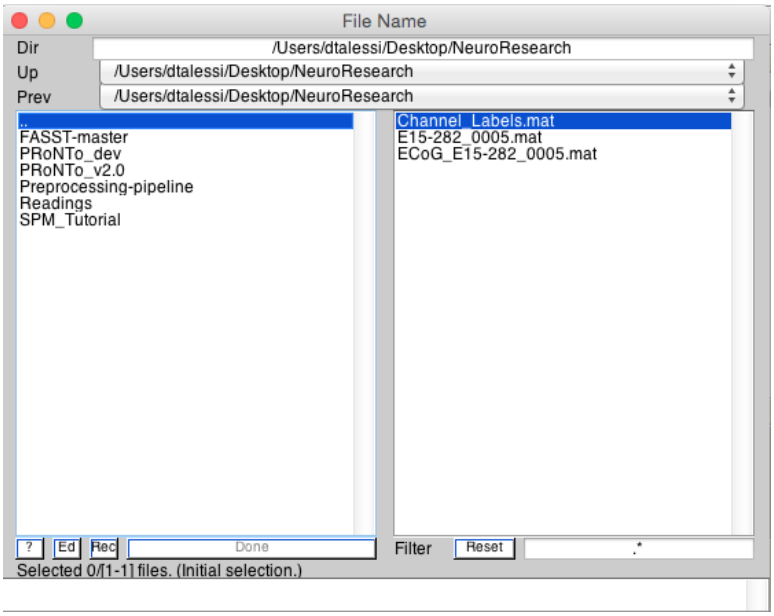
If a parameter is optional, you can simply not include it in the function call. For example, if the array parameter in **my_function** were optional, you could have the following:

```
>> my_function('myfile', 3)
```

If, on the other hand, both the integer and array parameters were optional, then in order to satisfy the ordering rules, you would still need to specify an integer (or an empty argument, **[]**) if you wanted to include an array argument. In other words, **my_function('myfile', [1 2 3])** would not be a legal function call because the second parameter is expected to be an integer, but **my_function('myfile', [], [1 2 3])** or simply **my_function('myfile')** should work.

Note that when a parameter is optional, a default value will be used in the case that the user supplies the empty argument, **[]**, or leaves the argument out entirely. You will need to consult the function's .m file to confirm what the defaults are. For the preprocessing pipeline, most of the defaults can be found in the file **get_defaults_Parvizi.m**.

For the preprocessing functions, if you do not supply a filename when you call a function, you will be prompted to specify a filename via the SPM file selection GUI:



Use the left-hand side to navigate directories (the ". ." will bring you to the directory one level up), and the right-hand side to specify the file or directory you wish to use. Click "Done" when you are finished.

If you want to manually select your files before entering a function, you can type **`fname = spm_select;`** to open the file selection GUI. After selecting your file(s), you can use **`fname`** as your filename input. This can be helpful if you want to call a function on the same file multiple times, or if you want to run a function on multiple files.

MATLAB scripts

Scripts are .m files that contain a series of MATLAB commands or functions. Running a script calls each of these commands/functions in sequence, as if you typed them into the MATLAB command line. Scripts do not take parameters, and they do not return values. If variables are created within the script to store the results of function calls or calculations, however, these variables will become part of the global space once the script is completed. In other words, after running the script, you will be able to use and access any variable that was created in the script (granted that you are running the same MATLAB session; if you quit and restart MATLAB, you will need to rerun the script). You can run a script by just typing the name of the script file, without the .m at the end.

For example, let's say we have a script file, `my_script.m`, that contains the following lines:

```
% Beginning of script
x = 2
y = 3
z = x + y
% End of script
```

Typing 'my_script' (without the single quotes) in the MATLAB command window will run those 3 lines, just as if you had typed them in on the command line. When the script has completed, you can use the

variables **x**, **y**, and **z** as you normally would a variable created at the command line, e.g. typing `m = z*x*y` at the command line would store the value 30 in the variable **m**.

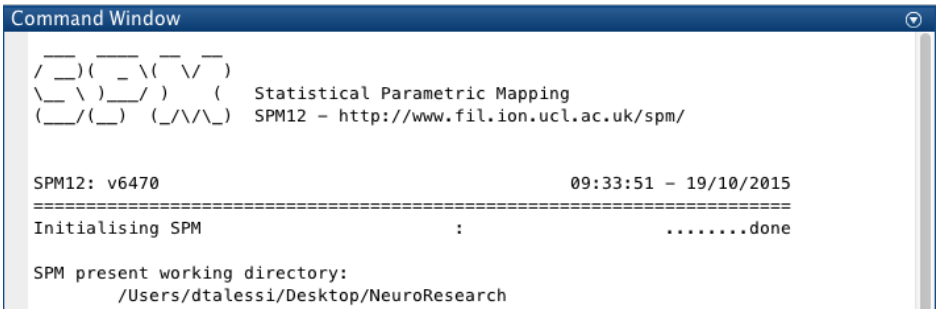
Useful MATLAB commands

'**`clear all; pack`**': clears the command space (i.e. variables) and acts as if MATLAB has been restarted without actually quitting and restarting the application.

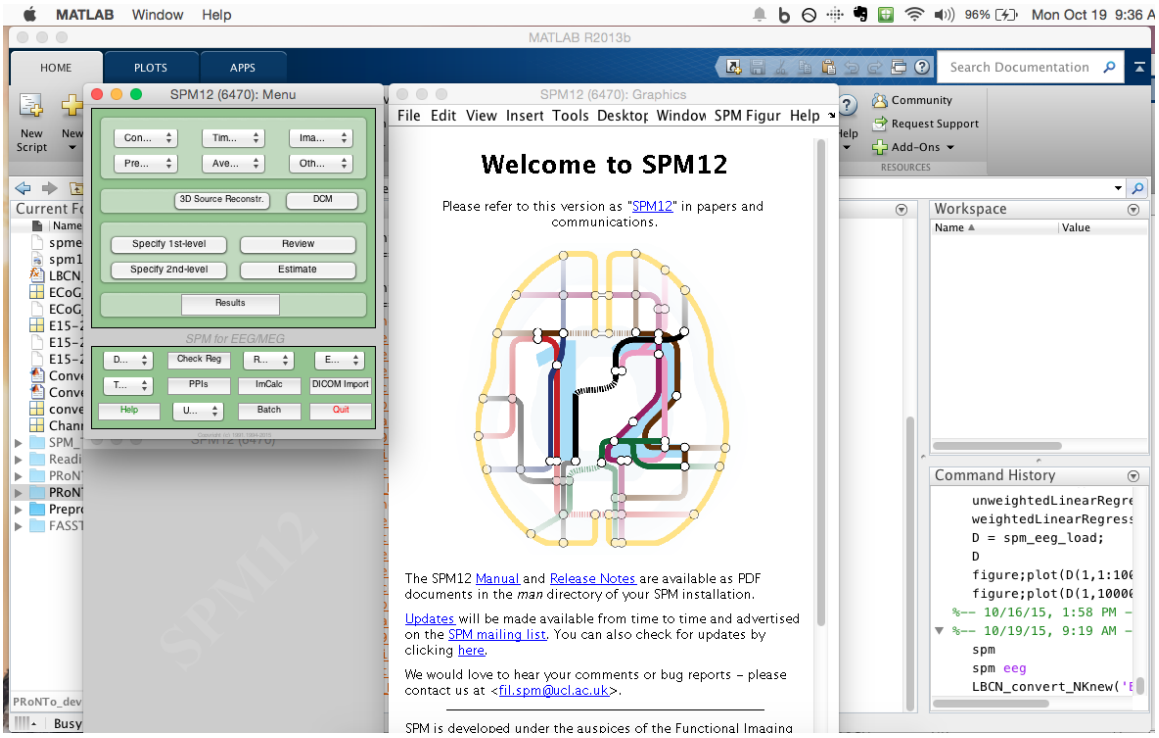
Launching SPM

While it is not necessary to start SPM in order to run the preprocessing functions, doing so allows you to leverage useful SPM features, including (but not limited to), visualization tools, the SPM batch editor for script editing, and a progress bar to track progress of preprocessing steps.

To launch SPM, type **`spm eeg`** at the command line and press Enter. SPM will start initializing:



SPM will open up three new windows. The green window on the upper left is the SPM "Menu," which allows you to perform all preprocessing steps through the GUI. The gray window below will show the progress of the SPM preprocessing functions, and the graphics window on the right can be used for visualizations.



Useful SPM commands

'**`spm_update update`**' : update SPM to latest version (note that this command will *update* but not *upgrade* your version of SPM; i.e. if you are using SPM 8, this command will not upgrade you to SPM 12)

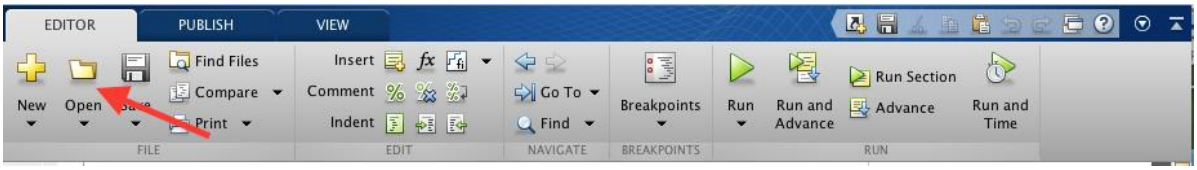
'**`methods meeg`**' : returns a list of available methods for EEG data

Editing the batch

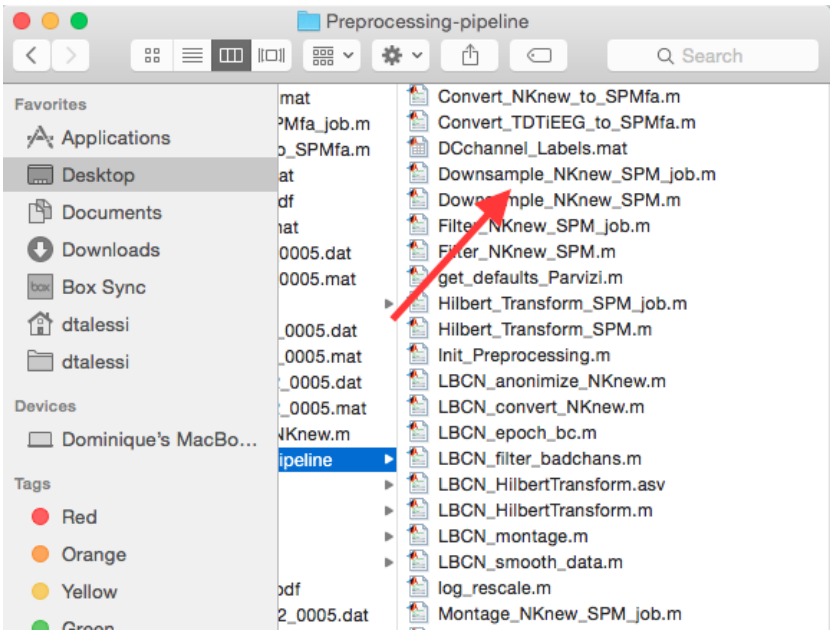
If you would like to make changes to the parameters used in a batch file, you can do so in one of two ways. To illustrate this process, we'll take the example of the downsampling batch, where we'd like to modify the new sampling rate (default: 1000Hz).

Editing the batch files directly

The batch file for downsampling is named 'Downsample_NKnew_SPM_job.m'. You can open the file directly in the MATLAB editor:



Conversely, you can open the file from a finder/explorer window (e.g. Finder for Mac), which will, by default, open the file in the MATLAB editor. You can right click on the file to open it with a different editor, e.g. TextEdit.



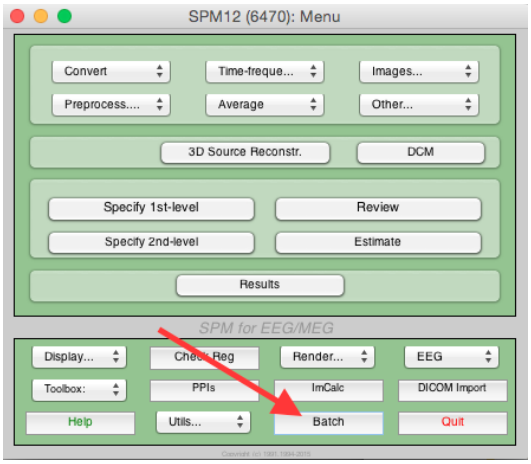
The batch files will have a similar form to the following:

```
%-----------------------------------------------------------------------
% Job saved on 23-Jul-2015 16:14:41 by cfg_util (rev $Rev: 6134 $)
% spm SPM - SPM12 (6225)
% cfg_basicio BasicIO - Unknown
%-----------------------------------------------------------------------
matlabbatch{1}.spm.meeg.preproc.downsample.D = '<UNDEFINED>';
matlabbatch{1}.spm.meeg.preproc.downsample.fsample_new = 1000;
matlabbatch{1}.spm.meeg.preproc.downsample.prefix = 'd';
```

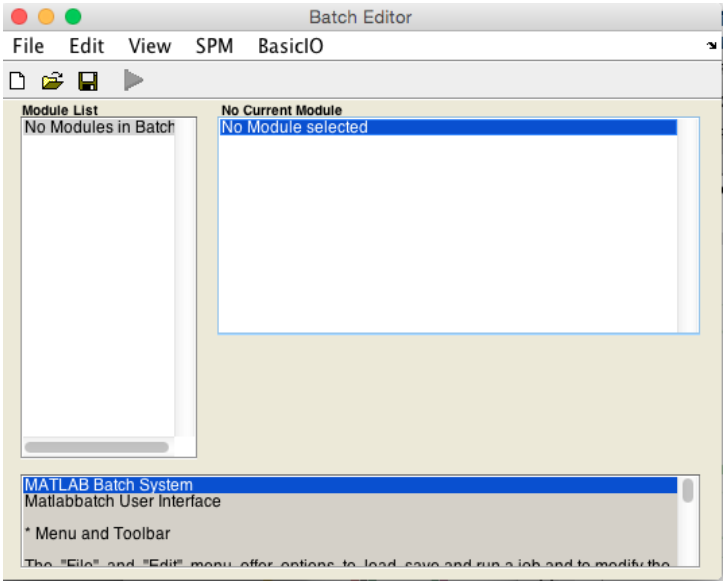
In this example, the two parameters are the new sampling rate, which is defined to be 1000, and the prefix, 'd', to add to the resulting file. (The "D" parameter is essentially a placeholder for the *.mat file that you will specify when you run the script.) You can adjust these parameters to suit your needs. See below for notes on saving your changes.

Modifying parameters through the batch editor.

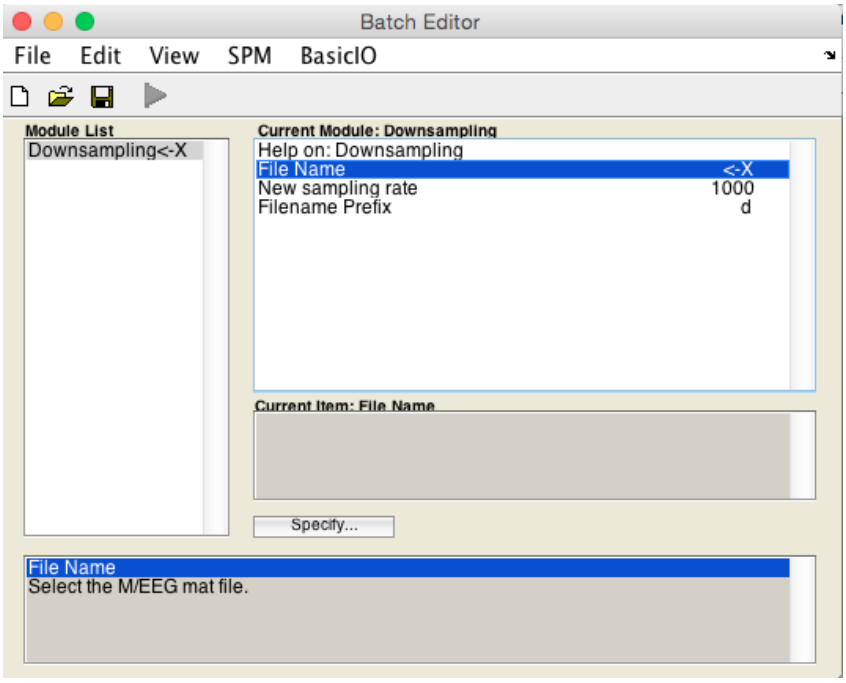
To open up the batch editor, click on "Batch" in the SPM Menu:



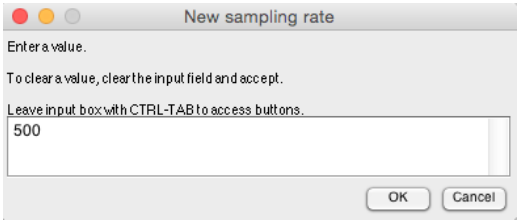
Clicking this button will open up the batch editor:



Click on “File”-> “Load Batch” and select the batch file you want to edit. If you are editing the downsampling file, for example, you should get the following window:



Simply click on the field of interest, e.g. “New sampling rate”, click “Specify,” and type in the appropriate value, e.g. 500.



Additionally, you can replicate or delete modules by clicking on the module in the “Module List” on the left side of the window, and clicking “Edit -> Replicate Module” or “Edit -> Delete Module”, respectively. Right-clicking on a module will provide the same options. You can find other modules under the “SPM” tab if you are interested in adding a new module. Before closing the window, you will need to save your edits, as described in the next section.

Saving your changes

Whether you make your changes to the batch file directly or through the batch editor, you will need to “Save” to keep your changes. If modifying the batch file directly, simply click the “Save” or “Save as…” button. If using the batch editor, you will need to click “File”->”Save batch and script” and enter the filename you wish to use, e.g. ‘Downsample_NKnew_SPM500’. It will save a .m routine that calls a ‘job’ routine (the actual batch data). The ‘_job’ suffix will be appended to your batch file name automatically.

As usual, saving your changes under the same filename will overwrite the previous contents of the file. If you would like to save a version of the original file, then you can save the modified file under a different

name. In this case, you will need to modify the main function to call the correct batch. For example, the LBCN_convert_NKnew function contains the following code to perform downsampling:

```
    % Step 3: Downsample ECoG data to 1000Hz
    %--------------------------------------------------------------------
    if downsample
        jobfile = {which('Downsample_NKnew_SPM_job.m')};
        namef = fullfile(path_block,D.fname);
        [out]=spm_jobman('run', jobfile, {namef});
        D = out{1}.D;
    end
end
```

If you modify the downsampling batch script and save it under a different name, you will need to update the **`jobfile = {which('Downsample_NKnew_SPM_job.m')};`** line to use your modified batch, e.g. **`jobfile = {which('Downsample_NKnew_SPM500_job.m')};`**.

Another option is to save your changes under the same filename, and then restore the default file using GitHub. To do this, open GitHub and 'discard changes' after the batch has been used. This will overwrite the version modified by the user with the version on the server, thus restoring the defaults. Please note that you will not be able to sync the GitHub folder if you have local changes that you have not committed. If you would like to keep those changes, saving locally in another folder (i.e. not the GitHub synced folder) might be a better option.

Step 1: Conversion

The first step is to convert the data files into a format SPM can work with. The method used to perform the conversion depends on the format of your raw data. In most cases, we advise you to use the SPM conversion routines (batch or spm_eeg_convert). The provided routines might work in some cases but are specific to the data format we had in the lab and to the way the data was exported from the proprietary recording software. We also advise to perform downsampling to reduce the computation load. This can also be done in SPM directly (batch or spm_eeg_downsample). Keep in mind that the downsampling should be three or four times higher than your highest frequency of interest (e.g. to be safe, 1000Hz for highest frequency of interest of 200Hz). Once you have performed those steps, you can jump to Step 2: Visual Inspection.

Conversion of EDF Files

Function: LBCN_convert_NKnew

Description: Converts (and optionally downsamples) EDF data to SPM format.

Parameters:

1. fname (optional): Name of the file to convert. If not specified, you will be prompted by the SPM file-selection GUI. Note that multiple files can be selected, in which case separate subdirectories (e.g. "Block1", "Block2", etc.) will be created to store the results.
2. path_save (optional): Name of location in which to store results. If not specified, the location of the selected file will be used.
3. downsample (optional): Flag to specify whether or not to downsample the data; 1 to downsample, 0 otherwise. The default value for downsampling is 1kHz. It is specified in the batch job and can be modified as described below.
4. exclude (optional): vectors of channel indices or cell array of channel names that are to be excluded from conversion. This allows to exclude channels that are not of interest from the data file (e.g. EKG or empty channels).

Output: 4 files

1. Whole dataset in SPM format, which consists of a *.dat file, which stores the data, and a *.mat file, which stores the header in a data structure "D".
2. Brain activity data in SPM format. The names of these files (.mat and .dat) will be prepended with "ECoG_". These are extracted from channels with labels 'POL' in the edf header.
3. DC channels (e.g. from microphone and diod). The name of this file will be "DCChans_*.mat". These are extracted from channels with labels 'DC' in edf.
4. Downsampled brain activity in SPM format (if downsampling used). The name of this file will be prepended with "d" (so the converted *and* downsampled data will have the prefix "dECoG").

Those files will be saved in the path_save directory.

Running the function

You can run the conversion function simply by typing the function name, LBCN_convert_NKnew, at the command line with the appropriate arguments, e.g. **`LBCN_convert_NKnew([], [], 0, []).`**

You may see some warning messages, e.g.

```
Data type is missing or incorrect, assigning default.
```

You can ignore these messages.

The time it takes for the conversion to complete will depend on the size of your data file (i.e. the duration of the recording and the sampling rate). Once the conversion is completed, you should see the following in the MATLAB command window:

```
SPM M/EEG data object
Type: continuous
Transform: time
1 conditions
49 channels
1062465 samples/trial
1 trials
Sampling frequency: 1000 Hz
Loaded from file  /Users/dtalessi/Desktop/Prepro_Jessica/S15_90_SO/S15_90_SO_04_MMR2/dECoG_E15-579_0004.

Use the syntax D(channels, samples, trials) to access the data
Type "methods('meeg')" for the list of methods performing other operations with the object
Type "help meeg/method_name" to get help about methods
```

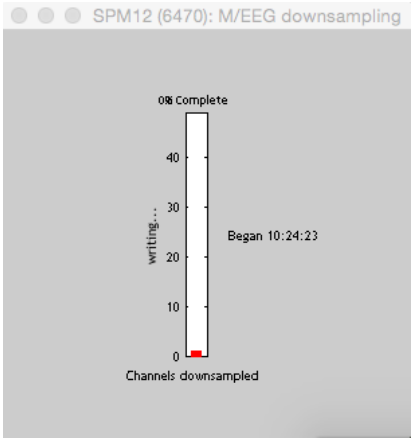
This gives a description of the SPM M/EEG data object that now stores your data, as well as the syntax for working with this object.

If the user chose to perform downsampling, the command window would show the following:

```
>> LBCN_convert_NKnew()
Data type is missing or incorrect, assigning default.
Data type is missing or incorrect, assigning default.


------------------------------------------------------------------------
Running job #1
------------------------------------------------------------------------
Running 'Downsampling'

SPM12: spm_eeg_downsample (v6016)                  11:29:22 - 11/11/2015
========================================================================
Done    'Downsampling'
Done
```

Generally, running an SPM function or a function/script that invokes an SPM function will print to the command window in a similar fashion. (The conversion leverages FASST, not SPM, functions, which is why you don't get a similar display for conversion.)

If you have SPM open, you can use the bottom-left SPM window to track the progress of the downsampling:



Conversion of TDT files

Function: Convert_TDTiEEG_to_SPMfa

Description: Converts TDT data (i.e. raw ECoG data in .mat format) to SPM format. This is lab specific but could be used if each channel is stored in a separate .mat file, under the variable name 'wav'. All input channels have to be recorded with the same sampling rate.

Parameters:

1. fsample: The sampling rate, which can be specified by a specific number or by one of the following 3 options:
    a. 'TDT' (default: 1525.88Hz). If no sampling rate is provided, this option will be used as the default.
    b. 'oldNK' (default: 1000Hz)
    c. 'newNK' (default: 1000Hz after export)
2. fchan (optional): The names of the .mat files corresponding to the brain signal on each electrode. If no names are specified, the SPM GUI will prompt you to choose the files.
3. downsample (optional): Flag to specify whether or not to downsample; 1 to downsample, 0 otherwise. If not specified, the flag will be set to 0 by default. The default parameters for downsampling TDT files are the same parameters as are used when converting NK data (i.e. the same batch is called).
4. path_save (optional): Name of location in which to store results. If not specified, results will be stored in the same directory as the electrode .mat files.
5. bch (optional): Indices of the bad channels. These channels will not be converted at all. It is similar to the 'exclude' input for edf conversion.

Output: 2 files

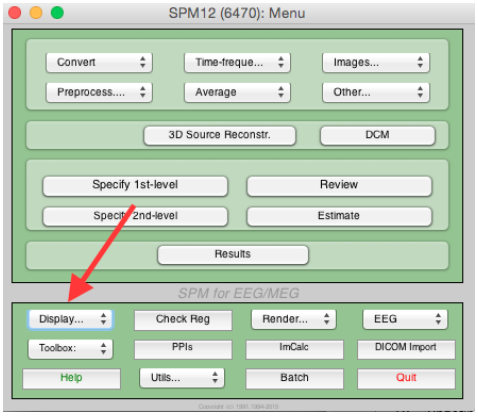
1. Brain activity in SPM format. Recall that SPM format includes a .dat file and a .mat file.
2. Downsampled brain activity in SPM format (if downsampling used).

Step 2: Visual Inspection

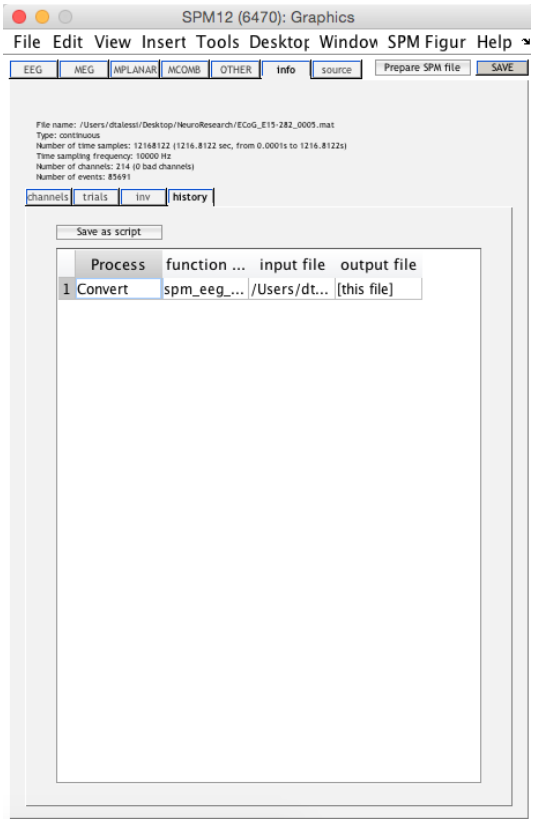
As you work your data through the pipeline, it is good practice to regularly look at the data and verify that it looks as you expect it to. Now that we are done with the conversion step, the data is ready to be visualized. This can be done in a variety of ways, based on SPM, Matlab command, FieldTrip or FASST. We recommend FASST or FieldTrip for continuous files and SPM for epoched data.

SPM GUI

Click on "Display…"-> "M/EEG" in the SPM Menu:

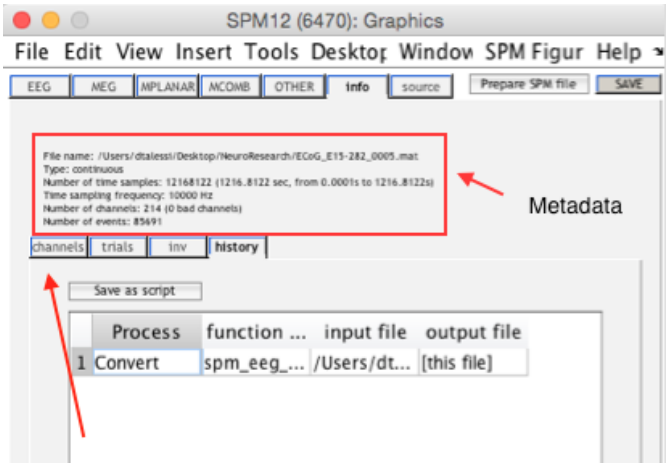


When asked to specify a file, choose "ECoG_file_name.mat" (or "dECoG_file_name.mat" if the data was downsampled), where file_name is the name of your original data file. The SPM Graphics window should update to the following:

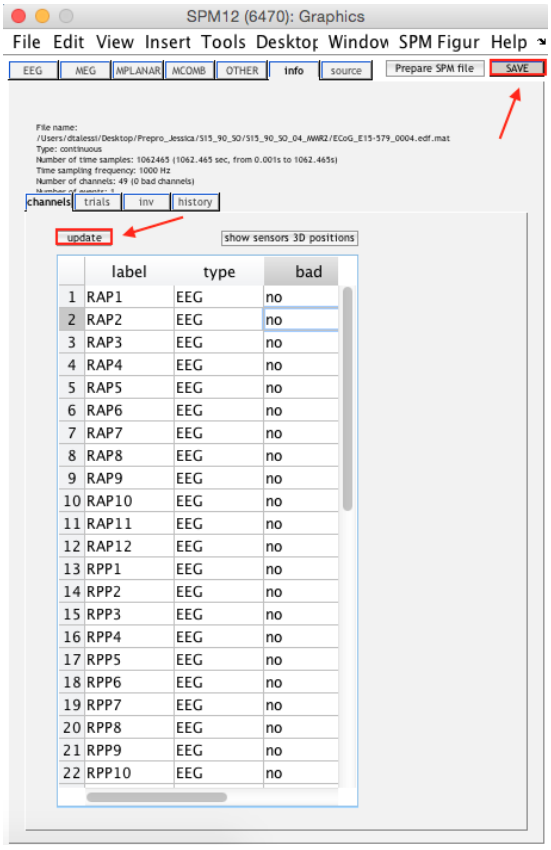


(Note that if downsampling was performed, there will be an additional “Downsample” step, below the “Convert” step, in the window above.)

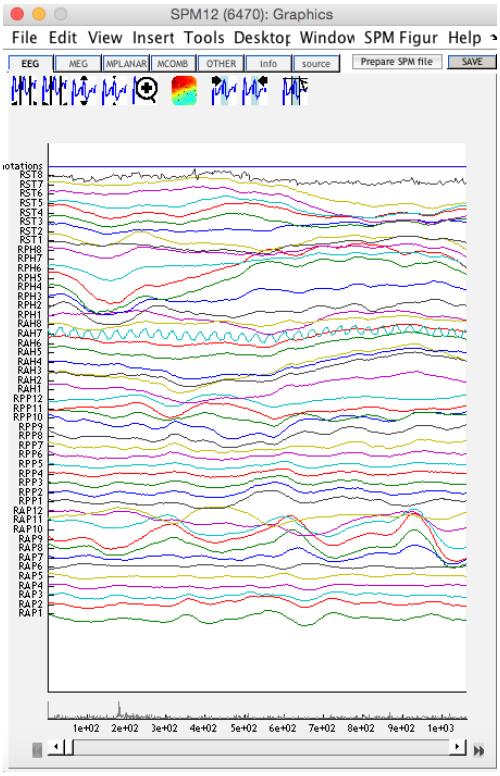
As you can see, the window displays metadata, which you can quickly verify for correctness. To explore the data at a more granular level, you can click on the “channels” tab, which will pull up a table containing basic information about all your channels.



If you wish to make any modifications to the data displayed in this table, you can do so directly; simply click on the cell in the table that you want to modify, make the change, click the “update” button above the table, and then the save button on the top right corner of the window to save your changes.

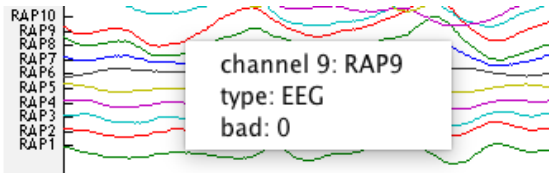


To visualize the data in this editor, click on the “EEG” tab towards the top left of the window. Channels will be displayed vertically.

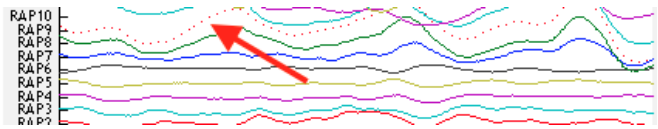


You can use the 9 buttons along the top to: 1) increase the width of the plotted time window, 2) decrease the width of the plotted time window, 3) increase contrast (intensity rescaling), 4) decrease contrast (intensity rescaling), 5) zoom (click on the time series data to zoom in, double click to zoom out), 6) visualize scalp data via scalp interpolation, 7) go to the closest selected event, in the forward direction, 8) go to the closes selected event, in the backward direction, 9) add an event to the current selection.

Right clicking on a particular time series will open a menu in which you can mark a channel as good or bad (by clicking on “bad: 0”). Make sure to click the “Save” button at the top right corner of the window to save any changes.



The time series for bad channels will be shown as dotted lines.



MATLAB Command Line

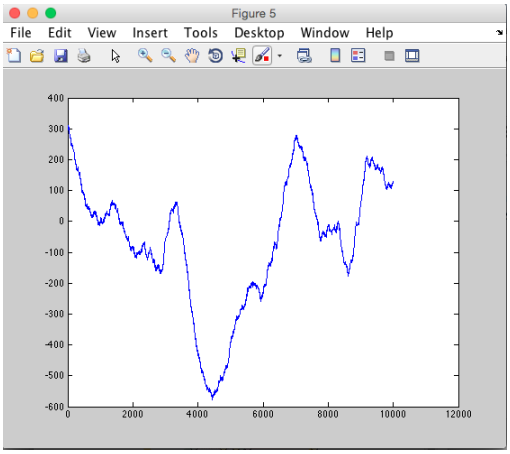
You can also inspect your data via the command line. To do this, type **`D = spm_eeg_load`** in the MATLAB command window, and pick the ECoG file when prompted to specify a filename. Metadata about the ECoG file will be printed in the window, and you will be instructed on how to access information from the data structure D:

```
>> D = spm_eeg_load
SPM M/EEG data object
Type: continuous
Transform: time
1 conditions
214 channels
12168122 samples/trial
1 trials
Sampling frequency: 10000 Hz
Loaded from file  /Users/dtalessi/Desktop/NeuroResearch/ECoG_E15-282_0005.mat

Use the syntax D(channels, samples, trials) to access the data
Type "methods('meeg')" for the list of methods performing other operations with the
Type "help meeg/method_name" to get help about methods
```

To see a visual representation of the data, you can use the following syntax:

`>> figure; plot(D(10, 100000:110000))`

This will plot data from channel 10, samples 100000 through 110000. Your plot might look something like the following (though the specific signal will, of course, depend on your data):



FieldTrip

Function: LBCN_databrowser_FT

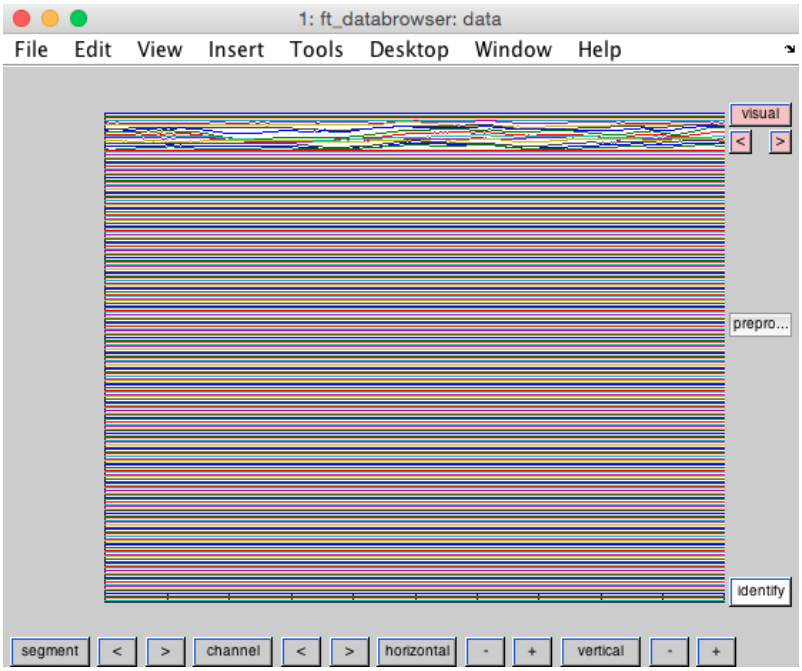
Description: Opens the FieldTrip data browser for data visualization.

Parameters:

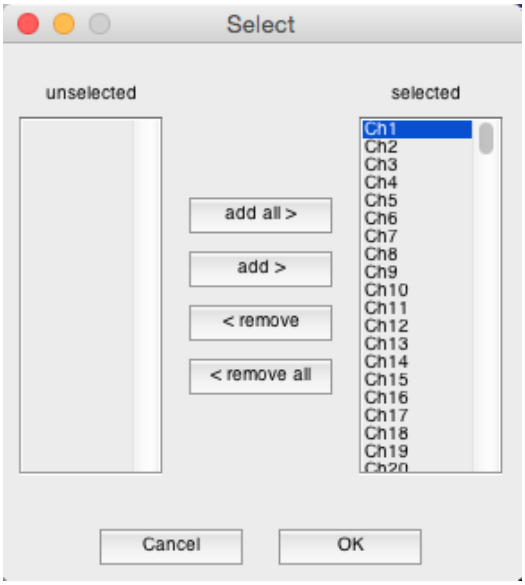
1. filename (optional): Name of file to display, in SPM format (.mat) or EDF format (.edf). If no filename is provided, you will be asked to select a file via the SPM GUI.

Output: Interactive FieldTrip data browser display.

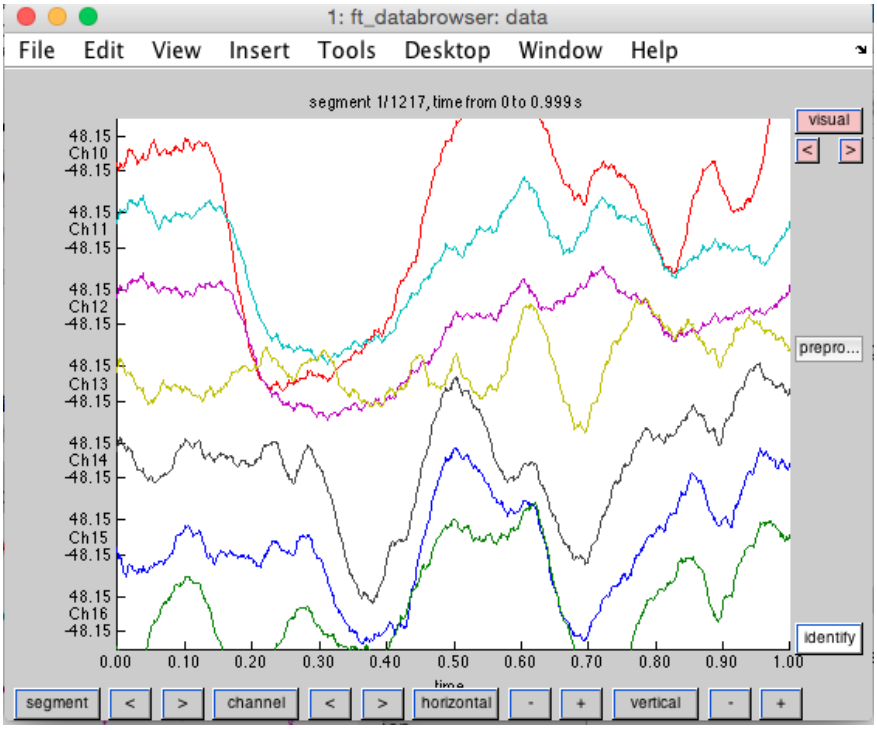
When you first open the display, it may look something like the following:



This initial display shows all the channels, making it difficult to glean any information. To make the data easier to work with, you can click on the “channel” button, which will pull up the following window:



Here you can select which channels you would like to display. By choosing channels 10-16, for example, I get the following display:



You can modify the horizontal and vertical scales by a) clicking the "horizontal" and "vertical" buttons and specifying the scale numerically, or b) by clicking the "+" and "-" buttons.

You can also use the keyboard to manipulate the visualization, as explained in the MATLAB command window:

```
You can use the following keyboard buttons in the databrowser
1-9                 : select artifact type 1-9
shift 1-9           : select previous artifact of type 1-9
                      (does not work with numpad keys)
control 1-9         : select next artifact of type 1-9
alt 1-9             : select next artifact of type 1-9
arrow-left          : previous trial
arrow-right         : next trial
shift arrow-up      : increase vertical scaling
shift arrow-down    : decrease vertical scaling
shift arrow-left    : increase horizontal scaling
shift arrow-down    : decrease horizontal scaling
s                   : toggles between cfg.selectmode options
q                   : quit
```

Aside from the basic viewing options, FieldTrip offers additional tools and features, which can be explored using the tabs at the top of the display.

FASST

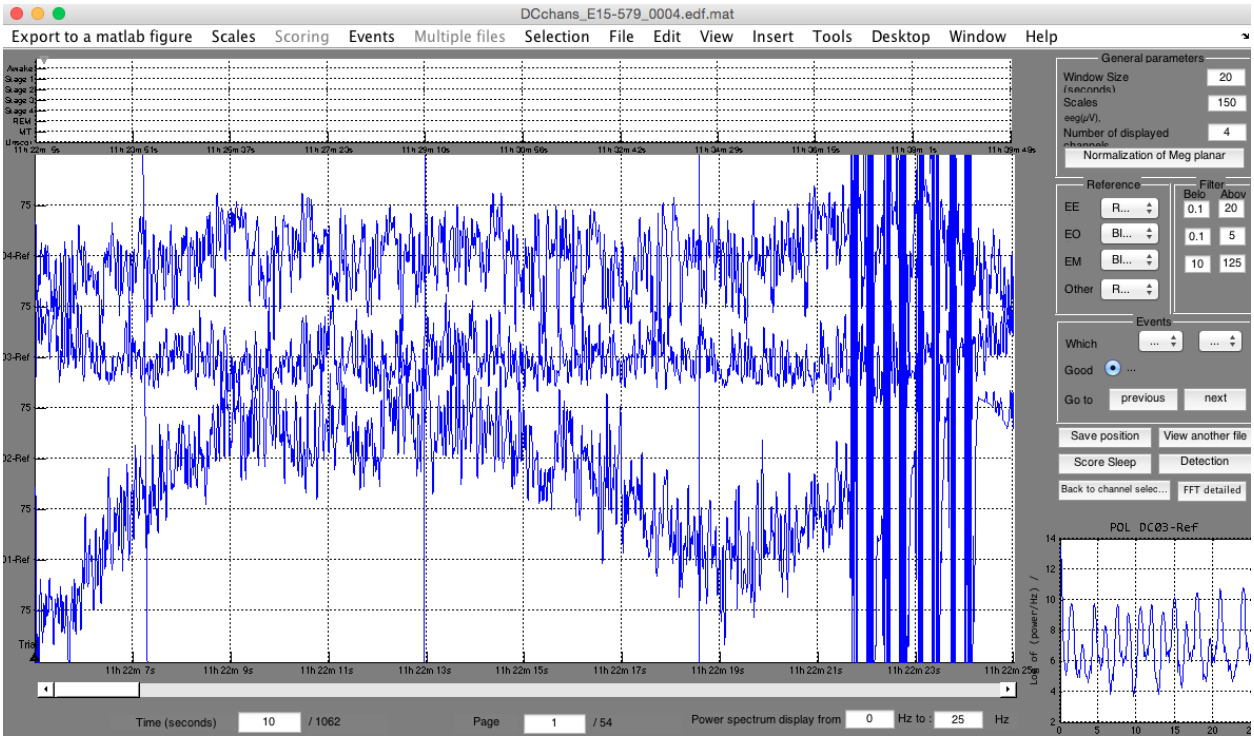
Function: LBCN_databrowser_FASST
Description: Opens the FASST data browser for data visualization.
Parameters:

1. filename (optional): Name of the file to display, in SPM (.mat) or EDF (.edf) format. If no file is provided, you will be prompted to select one via the SPM GUI.

2. indchan (optional): Indices of specific channels to display. If left empty, all channels will be displayed. Note that you can change the number of channels to display in the interface itself.

Output: Interactive FASST display of the dataset.

The interface for FASST looks like the following:

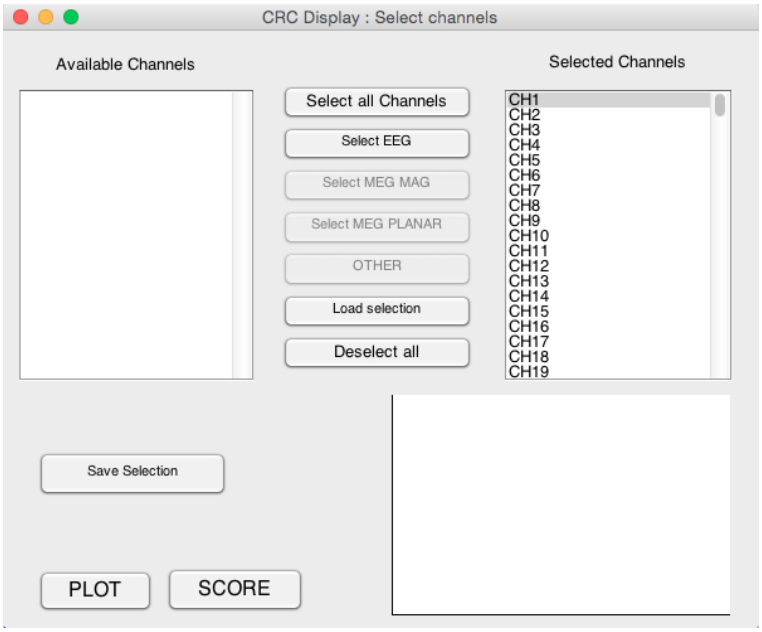


The bottom right hand corner displays a spectrogram of the area your mouse is hovering over, and the window along the top can be used to score sleep. The parameters in the top right corner control the resolution of the display (i.e. the window size / time scale and the number of channels to display). The text fields along the bottom of the screen can be used to, for example, display only high frequency signals (via changing the "Power spectrum display" parameters). There are also options on the right hand-side of the display to specify which events you want to examine.

You can view different channels by moving the slider on the right-hand side of the graph, and you can view different time windows by moving the slider along the bottom of the panel. The signals for "bad" channels will be displayed with a dotted line.

You can also right-click on a channel name, along the left border of the display, to see its spectrogram in a separate window, allowing you to save that particular spectrogram.

Lastly, you can specify which channels you want to display by clicking on “Back to channel selection” on the right hand side, above the spectrogram grid. The GUI for selecting channels is as follows:



As with FieldTrip, there are additional tools and features that can be explored using the tabs at the top of the display.

Step 3: Filtering and bad channel detection

In an effort to increase signal-to-noise ratio, it is important to detect bad channels and to remove them or otherwise note them. Our preprocessing pipeline supports both manual and automatic detection of bad channels.

Manual detection

The visualization tools described above can be used to manually identify bad channels. Unless you use a tool that automatically updates the underlying M/EEG object, you will need to record in a vector (e.g. [1, 3, 53, 33]) the indices corresponding to the channels you have identified as bad. Alternatively, you can create a cell array with the bad channel labels (e.g. {'TP8', 'TP10'}). This vector can then be input into the LBCN_filter_badchans function described below.

Automatic detection

Function: LBCN_filter_badchans

Description: Filters data for line noise + two harmonics and detects bad channels automatically (with an option to manually specify bad channels as well). Important note: when you load multiple files, it can detect and mark 'bad' channels that are common to all input files. This is useful when running the same experiment in multiple blocks.

Parameters:

1. file(s) (optional): Name(s) of file(s) to perform filtering and bad channel detection on. If not specified, you will be prompted by GUI to provide filename(s). Note: The files should come from the same patient and have the same channels for bad channel detection to be performed jointly.
2. bch (optional): Vector with indices of bad channels, e.g. `[2 5 10]` , or cell array with labels of bad channels, e.g. `{'TP8', 'TP10', 'AT1'}`. If not specified, bad channels will be marked solely by the automated detection process.
3. filter (optional): Flag to filter the data or not; 1 for yes, 0 for no. If not specified, the data will be filtered by default. Three band-stop Notch filters are applied, at 57 to 63Hz, 117 to 123 Hz and 167 to 173Hz. These bands can be modified in the corresponding batch (Filter_NKnew_SPM_job). The Filter_NKnew_SPM_job_China performs similar filtering for 50Hz line noise.
4. conserv (optional): (Relevant if number of files specified in "files" parameter is 2+). Number to indicate degree of conservancy for rejecting bad channels. Specify 0 to reject the union of all bad channels from all files. Specify a number x between 1/nFiles and 1 to exclude bad channels that are common to x*100% of all files. For example, specify the value "0.50" to reject bad channels that are marked bad in at least 50% of all files. If not specified, the default value of 0 will be used.

Output: 3 files

A separate file, prepended with 'f', will be created for each round of filtering. Since there are three rounds of filtering, the final data file, filtered and with bad channels marked, will have the prefix 'fff'. Only this last file will contain information about the bad channel rejection. The command line will also display which channels were detected as bad and/or spiky for each file.

Filtering

The LBCN_filter_badchans function will call the Filter_NKnew_SPM_job script to perform 3 rounds of filtering, as follows:

1. Type: Butterworth; Band: stopband; Cutoff(s): [57 63]; Direction: zero phase; Filter order: 3; Filename prefix; f
2. Type: Butterworth; Band: stopband; Cutoff(s): [117 123]; Direction: zero phase; Filter order: 3; Filename prefix; f
3. Type: Butterworth; Band: stopband; Cutoff(s): [177 183]; Direction: zero phase; Filter order: 3; Filename prefix; f

If you would like to make modifications to the above parameters, you can edit the Filter_NKnew_SPM_job file directly, or open the job file in the batch editor and make your edits there, as explained above.

Methods

The function will use 2 methods to perform automatic bad channel detection:

Variance

First, the variance is computed for the signal of each individual channel. The median of these variances is then computed to identify a "baseline" variance (excluding the channels manually marked as bad). Any channel whose variance is greater than 5*median or less than 5/median will be rejected as bad channels, where the number 5 is a default value for the "varmult" variable as specified in "get_parvizi_defaults.m" (using a different threshold requires just a simple modification to this file). While there is no specific flat-channel detection procedure, the variance method should account for such channels.

Spikes

Here, the differences between successive time points are calculated for each channel. The number of spikes, where a spike is identified as a difference larger than 100 μV, is counted for each channel, and the median of the number of spikes across all channels is computed as well (again, excluding the channels manually marked as bad). Any channel whose spiky-channel count is greater than 3*median will be labeled as a bad channel (referred to as 'spiky' in output window), where the number 3 is a default value for the "stdmult" variable as specified in "get_parvizi_defaults.m."

A summary of the steps performed and their results will be displayed in the MATLAB command window:

```
>> LBCN_filter_badchans


------------------------------------------------------------------------
Running job #1
------------------------------------------------------------------------
Running 'Filter'

SPM12: spm_eeg_filter (v5876)                      12:34:25 - 09/11/2015
========================================================================
Done    'Filter'
Running 'Filter'

SPM12: spm_eeg_filter (v5876)                      12:35:21 - 09/11/2015
========================================================================
Done    'Filter'
Running 'Filter'

SPM12: spm_eeg_filter (v5876)                      12:35:55 - 09/11/2015
========================================================================
Done    'Filter'
Done

Bad channels for file 1:36  37
Spiky channels for file 1:1   2  33  37  38  43  44  45  48
Bad channels for all files: 1   2  33  36  37  38  43  44  45  48

ans =

    'RAP1'    'RAP2'    'RPH1'    'RPH4'    'RPH5'    'RPH6'    'RST3'    'RST4'    'RST5'    'RST8'


ans =

    [49x1062465x1 meeg]
```

Multiple Runs

You may need to run the function multiple times, with different parameters. If this is the case, the bad channels in the selected files will be deleted before re-running the bad channel detection, and they will not be appended. This means that if the user modifies the detection parameters after running the code once, he/she only needs to provide the filtered fffECoG file to obtain the "newly" detected bad channels. On the other hand, you can always enter channels that should be detected as bad manually through the 'bch' input.

Additionally, you should set the "filter" flag to 0 for any subsequent runs since, generally, the filtering process needs to be run only once. In this case, you should select the ''fffdECoG_filename.mat' as input.

Additional Tools

Spectrogram Plot

Function: LBCN_plot_power_spectrum

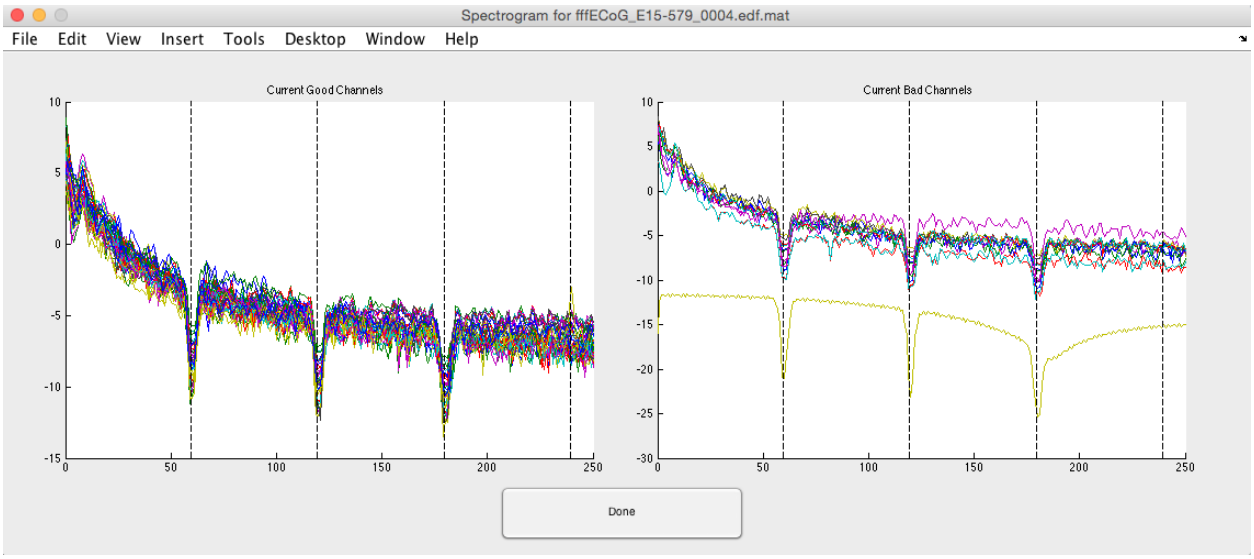
Description: Plots the power spectrum for good channels.

Parameters:

1. fname (optional): Name of the file to plot. If none is specified, you will be prompted to select a file via the SPM GUI.
2. timewin (optional): The time window to use for plotting the spectrogram, in the format [start stop] in seconds. If none is specified, the time window will be the length of the recording. It is *highly recommended*, however, that you specify a time window. Otherwise, it will take a very long time for the spectrogram to display.
3. power (optional): Flag to specify whether to plot the power (1) or the psd (0). If not specified, the parameter will be set to the default value of 0.
4. indchan (optional): Indices of the channels to plot.
5. savespect (optional): Flag to specify whether or not to save the spectrogram; 1 for yes, 0 for no. The default option is 0. It is helpful to save the spectrogram if you plan to use it again; this way, the calculations need to be performed only once.
6. spectdata (optional): Filename of previously saved spectrogram data, which should contain variables 'data_pxx' and 'f'.

Output: Interactive plot of the power spectrum using pwelch.

The spectrogram display looks like the following:



The panel on the left displays the good channels, while the panel on the right displays the bad channels. On the left panel, you can click on the spectrum for a particular channel to label it as bad. The plot will immediately update to remove the spectrum for that channel from the left panel and add it to the right panel. Each time a channel is marked as good or bad, the MEEG structure is updated (i.e. this information is saved directly). Additionally, the MATLAB command window will record your selections:

```
>> LBCN_plot_power_spectrum([], [25 30])
Labelling RAH7 as bad.
Labelling RAH3 as bad.
Labelling RAH2 as bad.
Labelling RAH1 as bad.
```

Lastly, there are a variety of tools, listed under the "Tools" tab, that can be used to manipulate the display (e.g. to zoom in). Click "Done" when you are finished.

Kurtosis vs. Skewness

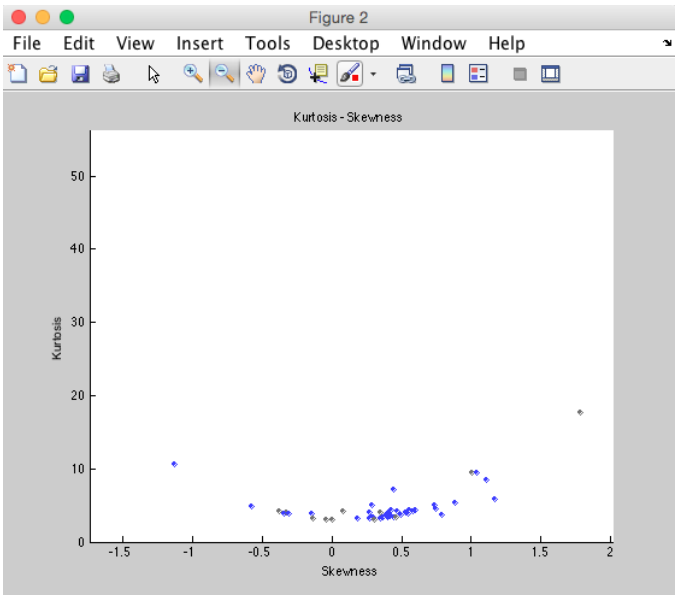
Function: LBCN_plot_Kurtosis_Skewness
Description: Computes and plots the Kurtosis and Skewness of each channel.
Parameters:

1. fname (optional): Name of the file, in SPM format, to look at. If no filename is given, you will be asked to provide one via the SPM GUI.
2. threshold (optional): Flag that indicates whether or not to perform thresholding; "1" to apply the threshold, "0" to only plot the data. If no flag is specified, the default of 0 will be used.

Output: Plot of Kurtosis vs. Skewness for each channel. This function *does not* update the M/EEG object. If you want to update it, do something like this:

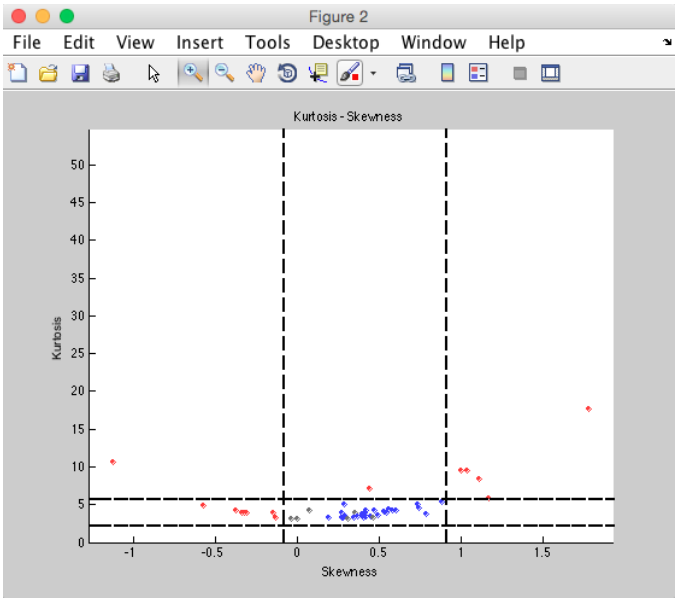
```
D = badchannels(D,good_channel_index,0);
D = badchannels(D,bad_channel_index,1);
save('/yourfile.mat'],'D');
```

By calling this function, a new window will pop up to display the plot:



"Good" channels are colored blue, and "bad" channels (as determined by previous analyses) are colored grey. You can click on a data point to identify which channel that particular data point corresponds to.

If the user chooses to use a threshold, then channels that fail to meet the threshold will be colored red. The lines defining the threshold are also shown. The channels displayed in red will however not be marked automatically as bad, as we have not been able yet to identify parameters that are generalizable.



Step 5: Montage / re-reference

ECoG is usually set up such that the signal recorded from one channel is relative to that recorded at the next. Re-referencing provides a way to change the reference of the data after the initial recording. While the most common technique is to re-reference using the average across all channels, the LBCN_montage function offers a "custom" option so that users can apply any re-referencing technique (given that the transformation can be represented as a matrix).

Function: LBCN_montage
Description: Creates a montage matrix, which will be used to re-reference the data.
Parameters:

1. file(s) (optional): Name(s) of file(s) to be re-referenced. If no filename is provided, you will be prompted to select the file(s) via the SPM GUI.
2. option (optional): Type of re-referencing to perform. There are 3 options (of which the second is the default should no option be specified):
    a. average: re-references using the average of all channels
    b. av_good: re-references using the average of only the good channels
    c. custom: re-references using a custom montage
3. montmat (optional): (Relevant if "custom" option is chosen.) Name of the .mat file containing the custom montage to be used for re-referencing. If the "custom" option is chosen but no montage file is provided, a GUI will prompt you to specify the file. The file should contain a structure with the 'old' channel labels, the 'new' channel labels, and the transformation matrix. Note that when changing the channel names and/or numbers, all channel information will be deleted (including bad channel detection). Please see the SPM manual for how to specify this matrix.

Output: Re-referenced MEEG objects saved in SPM format. All files produced will be prepended with prefix 'M'.

As before, there is a Montage_NKnew_SPM_job.m file whose default parameters can be modified directly or via the batch window. The current defaults are 655360 for block size, and "No" for keeping other channels. The latter option refers to the case in which the montage is applied to only some channels (i.e. only some channels are re-referenced). If you use the "No" default option, the resulting file will include only the channels to which the montage was applied.

You may want to perform another visual inspection at the end of the re-referencing stage to ensure that noise hasn't spread to other channels; if the re-referencing and bad channel detection were successful, your signals should look better now than before!

Step 6: Epoch and baseline correction

Up until this point, the data we've been working with has been continuous. By epoching the data (i.e. splitting up the data into individual trials), it becomes easier to start identifying patterns; epochs are often averaged together to cancel out noise and isolate the main signal. It can also be helpful to baseline correct, i.e. to subtract out the average pre-stimulus signal, such that signal of interest is not conflated with average noise. Trials are most commonly defined by the time of onset (i.e. stimulus presentation), and SPM provides methods to epoch in this way exclusively. Nevertheless, the LBCN_epoch_bc offers flexibility such that you can define trials by, say, the response time of the patient for each trial.

To use this function, an 'event file' is needed. If you do not need a special epoching (e.g. epoch around response but baseline correct according to onset), it might be simpler to use the regular SPM function. Otherwise, you need to create the following structure, saved as the variable 'events' in a .mat file.

```
events =

  struct with fields:

    categories: [1×6 struct]
```

In the present case, I have 6 categories. For each, I need to specify subfields:

```
events.categories(4)

ans =

  struct with fields:

     name: 'autobio'
     categNum: 4
     numEvents: 40
     start: [1×40 double]
     duration: [1×40 double]
     stimNum: [1×40 double]
```

These fields are necessary to insert the events into the data structure and epoch the file. All times should be specified in seconds since the beginning of the file.
In the present case, we have added a few other fields, as we would like to epoch around the time of onset, but baseline correct based on a time window before the trial onset.

```
     wlist: {40×1 cell}
     RT: [1×40 double]
     RTons: [1×40 double]
     sbj_resp: [1×40 double]
     correct: []
     subcat: []
```

These fields can have any names and will only be used during the epoching step. They will not be imported into the SPM MEEG object.

Function: LBCN_epoch_bc

Description: Epochs the data. Can be performed before or after time-frequency decomp.[1]
Parameters:

1. fname (optional): Name of the file to epoch. If no file is specified, you will be prompted to choose one via the SPM GUI.
2. evt (optional): Name of the file containing event information. If no file is specified, you will be prompted to do so via the SPM GUI. (Note that the LBCN_epoch_bc function expects the event file to be in the format used by and specific to the Parvizi lab, see above.)
3. indc (optional): Indices of the categories to epoch (e.g. there may be some rest or control events that you don't need to examine, so you would exclude the indices of these categories from indc). If none are specified, then all categories will be epoched at.
4. fieldons (optional): Name of the field, in the events.mat file, that defines the beginning of the epoch. If none is specified, then the default option of "start" (i.e. onset) will be used.
5. twepoch (optional): Time window, in ms, for the epoch, e.g. [0 1000]. If none is specified, then the mean of the RTs (response times) will be used.
6. bc (optional): Flag to specify whether or not to apply baseline correction; "1" to apply the correction, "0" otherwise. If unspecified, bc will be set to the default of 0.
7. fieldbc (optional): (Relevant if bc flag is set to "1".) Which field in the events.mat file to use for the baseline correction. If none is specified, the default field "start" (i.e. onset) will be used.
8. twbc (optional): (Relevant if bc flag is set to "1".) Time window, in ms, for the baseline correction. The default option is [-200 0].
9. prefix (optional): the file name will be prepended with the specified prefix. By default, 'e' will be added at the beginning of the file name. If multiple epochings are performed (e.g. based on stimulus onset, and then on RT), this parameter should be used not to overwrite other files.

Output: An epoched (and baseline corrected, if applicable) SPM object, D, saved in SPM format. Files will be prepended with the prefix (default: letter "e"). Information about D will be printed in the MATLAB command window:

```
>>  LBCN_epoch_bc([],[],[],'start',[-300 1700],1,'start',[-300 0])
Data type is missing or incorrect, assigning default.
Epoching trial (out of 140):   140Done: Epoching
SPM M/EEG data object
Type: single
Transform: time
5 conditions
49 channels
2001 samples/trial
140 trials
Sampling frequency: 1000 Hz
Loaded from file  /Users/dtalessi/Desktop/Prepro_Jessica/S15_90_SO
```

Example: epoch based on Response Time (i.e. start+RT for each trial, computed in a new field called 'RTonset') and baseline correct based on onset of trial:

```
LBCN_epoch('file_name.mat','event_file.mat',[],'RTonset',[-1000
200],1,'start',[-300 0],'RT_')
```

IMPORTANT NOTE: If you perform artefact detection on the continuous file in 'mark' mode before epoching, the routine will identify trials that include artefacted windows and mark the corresponding trials as 'bad' if it comprises artefacted windows in at least 30% of the channels. The threshold at 30% is quite arbitrary and can be modified in line 214 (`part>=0.3`) of LBCN_epoch_bc.

---

[1] Performing time-frequency decomposition after epoching will run more quickly, but you will need to specify wider windows for epoching (and baseline correction, if applicable) to avoid edge effects during the time-frequency decomposition stage. On the other hand, performing TF before epoching adds to the temporal correlation between time points and potentially trials, which is not recommended for further multivariate analysis.

To visualize your data at any point after epoching, you can use the LBCN_plot_averaged_signal_epoch function, described next.

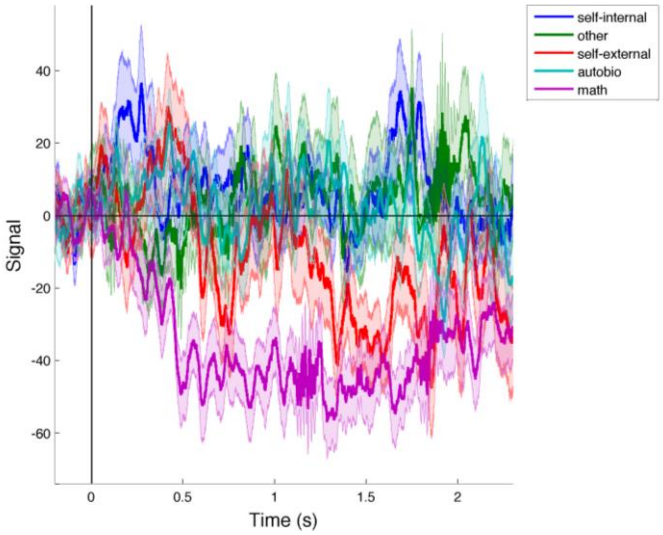
Plot Average Signal

Function: LBCN_plot_averaged_signal_epochs.m (in "utils" folder)
Description: Plots (and saves) the mean and standard error of the signal for each condition, on each channel.
Parameters:

1. fname (optional): Name of file to display. If none is specified, you will be prompted to select a file via the SPM GUI.
2. indchan (optional): Index or name of the channel to plot. If none is specified, all channels will be used one at a time.
3. conds (optional): List of indices of conditions, e.g. **`[1 5 50]`**, or cell array of names of conditions, e.g. **`{'Cond1', 'Cond5', 'Cond50'}`**, to display. If none is specified, all conditions in the file will be displayed. Using cell arrays, it is possible to pool multiple categories together, e.g. plotting only 2 lines **`{'Cond1', {'Cond5', 'Cond50'}},`** one for 'cond1' and one for the average of 'cond5' and 'cond50'.
4. timewin (optional): The time window to display, in milliseconds. By default, the function will use all time points.
5. save (optional): Flag to specify whether or not to save the plot(s); 1 for yes (default option), 0 for no. If saved, the plot(s) will be stored in a 'figs' subdirectory of the fname path.
6. suffix (optional): Suffix to append to saved plots. If none is specified, no suffix will be used.
7. colors (optional): Colors to display each plotted category with. This should be a [n_categories, 3] matrix. If not specified, the color map 'line' is used, with pink as its first color. If the number of rows in the colors matrix is smaller than the number of categories, the color map is completed using 'lines' for the remaining categories.
8. labels (optional): As it is possible to pool multiple categories together, it is also possible to specify 'labels', in the form of a cell array, for each plotted category.

Output: Plots of the signal average and standard error for each condition, one plot per channel. If 'save', a 'figs' subfolder is created, where all plots are saved under the name 'channel_label_suffix'. Plots are printed in 300dpi, png format.

The following shows the plot created for a single channel using the default options. See below to compare results before and after smoothing.



A similar function can be used to display the files epoched from onset and from RT (or any different field of epoch) next to one another.

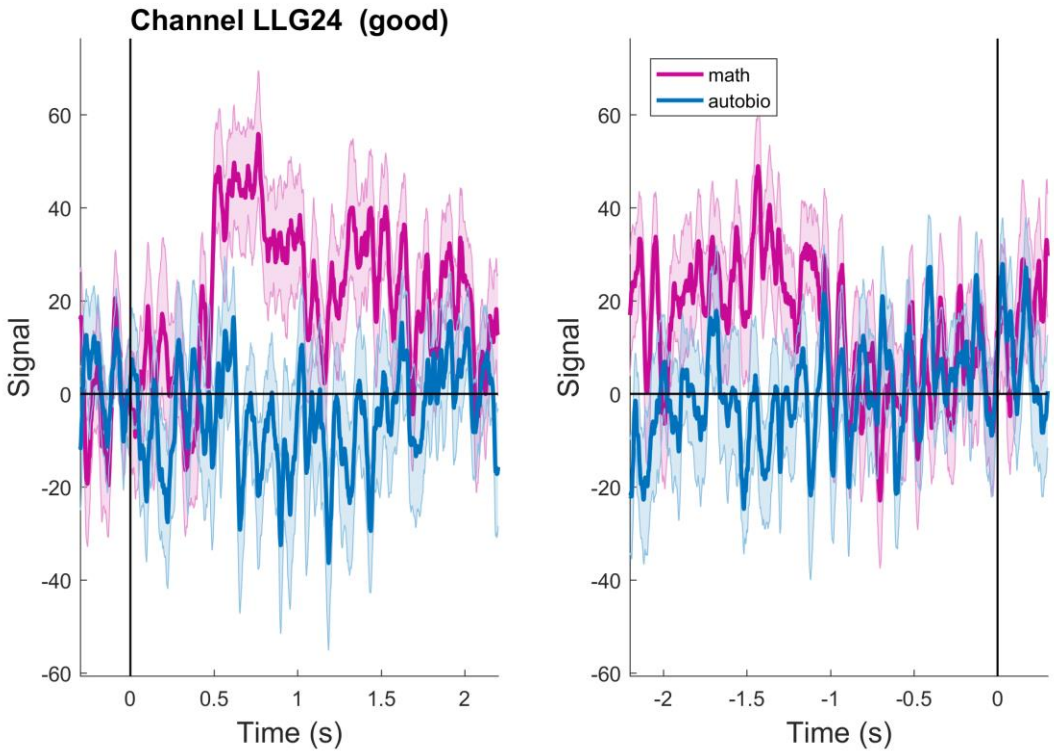
Plot Average Signal – onset and RT

Function: LBCN_plot_averaged_signal_epochsRT.m (in "utils" folder)
Description: Plots (and saves) the mean and standard error of the signal for each condition, on each channel, from 2 different files next to one another.
Parameters:

1. fname (optional): Name of first file to display. If none is specified, you will be prompted to select a file via the SPM GUI.
2. fnameRT (optional): Name of second file to display. If none is specified, you will be prompted to select a file via the SPM GUI.
3. timewin (optional): The time window to display the first file in, in milliseconds. By default, the function will use all time points.
4. timewinRT (optional): The time window to display the second file in, in milliseconds. By default, the function will use all time points.
5. indchan (optional): Index or name of the channel to plot. If none is specified, all channels will be used one at a time.
6. conds (optional): List of indices of conditions, e.g. `[1 5 50]`, or cell array of names of conditions, e.g. `{'Cond1', 'Cond5', 'Cond50'}`, to display. If none is specified, all conditions in the file will be displayed. Using cell arrays, it is possible to pool multiple categories together, e.g. plotting only 2 lines `{'Cond1', {'Cond5', 'Cond50'}},` one for 'cond1' and one for the average of 'cond5' and 'cond50'.
7. save (optional): Flag to specify whether or not to save the plot(s); 1 for yes (default option), 0 for no. If saved, the plot(s) will be stored in a 'figs' subdirectory of the fname path.
8. suffix (optional): Suffix to append to saved plots. If none is specified, no suffix will be used.
9. colors (optional): Colors to display each plotted category with. This should be a [n_categories, 3] matrix. If not specified, the color map 'line' is used, with pink as its first color. If the number of rows in the colors matrix is smaller than the number of categories, the color map is completed using 'lines' for the remaining categories.
10. labels (optional): As it is possible to pool multiple categories together, it is also possible to specify 'labels', in the form of a cell array, for each plotted category.

Output: Plots of the signal average and standard error for each condition, one plot per channel, for the first file on the left of the figure, and for the second file on the right of the figure. If 'save', a 'figs' subfolder is created, where all plots are saved under the name 'channel_label_suffix'. Plots are printed in 300dpi, png format.

The following figure shows the plot created for a single channel using the call:
`LBCN_plot_averaged_signal_epochsRT(fname,fnameRT,[],[],[],{'math','autobio'},1,'RT').`



Step 7: Artefact Rejection and Time-Frequency Decomposition

These steps of the pipeline are performed using solely SPM methods. While we provide a batch script that you can use, you will most likely want to create your own version or at least modify the parameters. An important detail is the edge effect, i.e. edges of the signal do not have the same profile as the 'inside' of the temporal window after performing the time-frequency decomposition. The duration of this effect on each side of your window depends on the frequencies studied, the method used to perform the TF (e.g. number of Morlet wavelets) and the sampling rate of the recording (after downsampling if performed). These can be computed as specified in (http://www.sciencedirect.com/science/article/pii/S1053811912009895).

Function: batch_ArtefactRejection_TFrescale_AvgFreq.m (in "utils" folder)
Description: Calls the specified batch script (a parameter), which will perform multiple pre-processing steps, namely artefact rejection, time frequency decomposition, and averaging within frequency bands.
Parameters:

1. fname (optional): Name of file to perform processes on. If none is specified, you will be prompted to select a file via the SPM GUI. You can select multiple files, they will be processed one after the other.

2. script (optional): Name of the batch script to call. If none is specified, the batch_ArtefactRejection_TFrescale_AvgFreq_job.m script will be used. This is where you can input your own batch to perform similar steps.
3. timewin (optional): Time window for rescale of TF (default: [-Inf 0]).

Output: d, a cell array with final M/EEG objects. Final files will contain prefixes as indicated in the steps below.

The script runs the following SPM processes:

1. Artefact detection: Here we are looking at jumps greater than 100 microV. While this jump size is the same that was used for bad channel detection, the latter rejects channels on the basis of the *frequency* of these jumps, while artefact detection simply looks for the *existence* of such a jump. Thus, artefact detection has a less stringent criteria for rejecting channels; if a jump is found in event x, then event x will be discarded across all channels. You may want to increase the threshold for bad channel detection to make sure that no additional channels (i.e. no channels aside from the ones detected as bad in previous steps) are marked as bad.[2] The names of the bad channels (as determined from this step or previous ones) will be shown in the MATLAB command window. The files created by this step will be prepended with the letter "a".
2. Artefact Removal: This step is optional. It simply removes the bad trials from the file. You can remove it if you would like to still see the trials labeled as 'bad'. If run, this step will create files prepended with the letter "r".
3. Time Frequency Decomposition: In almost all cases, you will want to construct your own method for time frequency decomposition. For example, you can choose between a number of algorithms, namely continuous Morlet wavelet transform, Hilbert transform, and multitaper spectral estimation, and the best algorithm to use will depend on your interests and/or the question you are trying to answer. The resulting file, prepended with "tf", will contain the instantaneous power. If you choose to include phase estimation (specify "yes" for the "Save Phase" option), then a separate file will be created, prepended with "tph", to store the phase estimates. You can choose the channels and frequencies for which you would like to compute power (and optionally phase) estimates. The batch script we provide uses Morlet wavelets (5 cycles) for frequencies between 1 and 170Hz (with larger jumps between larger frequencies, i.e. [1 2 3 … 12 13 16 19 22 … 37 40 45 50 … 65 70 80 … 160 170], to account for the fact that higher bands have better resolution), and it does not save the phase. If a user is interested in the phase, he/she might consider looking into the "phase amplitude coupling" module provided by SPM.
4. Rescaling: Generally, you will want to rescale the power data in some way; the raw data itself can be hard to draw insights from. There are several options to choose from: 'LogR' calculates the log of power in the baseline (user specified, can be a different file) for each frequency bin and then scales the data at each frequency bin; 'Diff' simply subtracts the baseline; 'Rel' transforms power into % of baseline units; 'Log' and 'Sqrt' compute the log and square root, respectively, without baseline correction. 'Zscore' creates a z-scored file, based on the mean and standard deviation in the baseline of each trial, or across all baselines (option 'pooled' set to 1). You will need to specify either the baseline and/or a file to calculate the baseline if the selected method requires so. The latter option may be helpful if, for example, your events are temporally close together; in such a case, it would be difficult to use the event data to find the baseline, and so you can instead use a file containing rest/control data to calculate the baseline. The files output by the

[2] If you performed bad channel detection on multiple files jointly, then depending on the 'conserv' input used, it is possible that bad channels exist in individual files. For example, if you used a 'conserv' value of 1 (and thus rejected only the channels marked as bad across all files), then there may still be bad channels that are not common to all files but that exist in some file(s) nonetheless. To ensure those channels are not marked as bad in Artefact detection (and hence that bad channel sets are consistent across files), the threshold should be as close to 1 as possible.

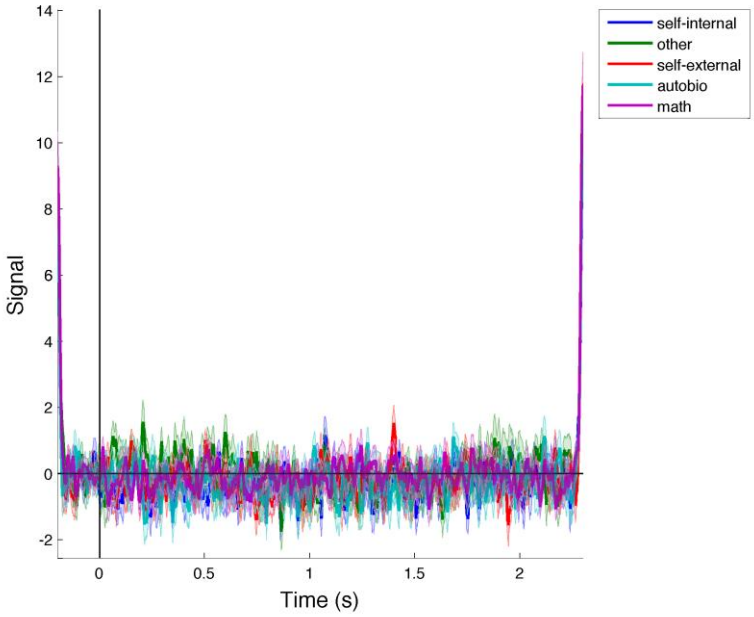
rescaling step contain the prefix “r”. The batch script uses the LogR option. If you choose to baseline correct in this step, be careful to specify a baseline smaller than the actual baseline to avoid edge effects (e.g., epoch in [-300 1200]ms after onset, TF on the time window, rescale on [-250 0]ms if looking at high frequencies, or [-200 0]ms if including lower frequencies or sampling rate in smaller). We recommend using the ‘pooled’ option for Z-score if your inter-stimulus interval is short (see http://onlinelibrary.wiley.com/doi/10.1111/ejn.13179/full).

5. Averaging: This last step will compute averages within a band of frequency (e.g. 70 to 170Hz, user specified). While, by default, SPM uses the prefix ‘P’ for the output filenames, the user can specify a prefix that uses the name of the frequency band, e.g. ‘HighGammaFilename’.

When running the function, your MATLAB command window should show which step is currently running and which steps have completed. The following is what your MATLAB command window should look like once all the steps have finished:

```
Running job #1
------------------------------------------------------------------------
Running 'Artefact detection'

SPM12: spm_eeg_artefact (v6035)                    18:00:16 - 15/11/2015
========================================================================

        SPM12: spm_eeg_artefact_jump (v6060)       18:00:17 - 15/11/2015
        ------------------------------------------------------------
0 rejected trials:
10 bad channels: RAP1 RAP2 RPH1 RPH4 RPH5 RPH6 RST3 RST4 RST5 RST8
Done    'Artefact detection'
Running 'Remove bad trials'

SPM12: spm_eeg_remove_bad_trials (v5079)           18:00:19 - 15/11/2015
========================================================================
Done    'Remove bad trials'
Running 'Time-frequency analysis'

SPM12: spm_eeg_tf (v6146)                          18:00:28 - 15/11/2015
========================================================================
Done    'Time-frequency analysis'
Running 'Time-frequency rescale'

SPM12: spm_eeg_tf_rescale (v5626)                  18:08:09 - 15/11/2015
========================================================================
Done    'Time-frequency rescale'
Running 'Average over frequency'

SPM12: spm_eeg_avgfreq (v5190)                     18:11:54 - 15/11/2015
========================================================================
Done    'Average over frequency'
Done
```

We can again use the LBCN_plot_averaged_signal_epochs function to plot our signal:



You can see the edge effects resulting from the TF analysis. You will want to keep these in mind when performing subsequent steps, e.g. when choosing parameters for rescaling and smoothing, as described in the next section.

Step 8: Smoothing the data (optional)

While smoothing at this stage is optional, you might want to smooth your data for visualization and/or for detecting response onset. Smoothing will take into account the temporal variability between time points. Note, however, that it decreases the temporal resolution of the signal and thus should be used with caution.

Function: LBCN_smooth_data
Description: Smooths data using a Gaussian window.
Parameters:

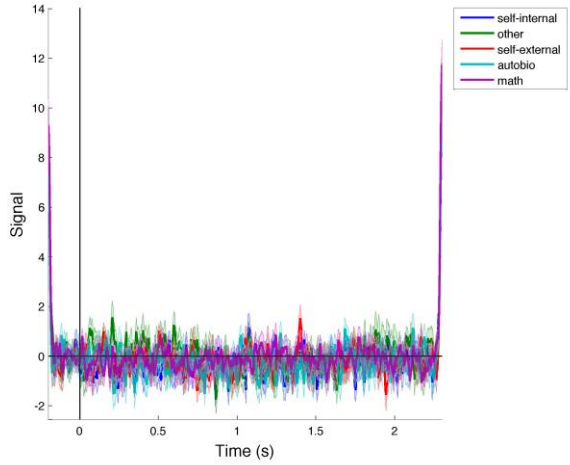
1. file(s): Name of .mat file(s) to smooth. If none is specified, you will be prompted to select a file via the SPM GUI.
2. win_length: Length of the window to smooth over, in milliseconds. This parameter represents the total length of the window, and not the FWHM of the Gaussian. The default used is 50ms, as is specified in **`get_defaults_Parvizi.m`**.
3. time_win: Time window of the signal to consider, in ms. Specify this parameter to avoid the edges from the TF decomposition. By default, the whole time window is used.

Output: Cell array containing the smoothed MEEG objects created. The resulting files contain the prefix "S". Note that the output file will contain the smoothed data for only the selected time window.

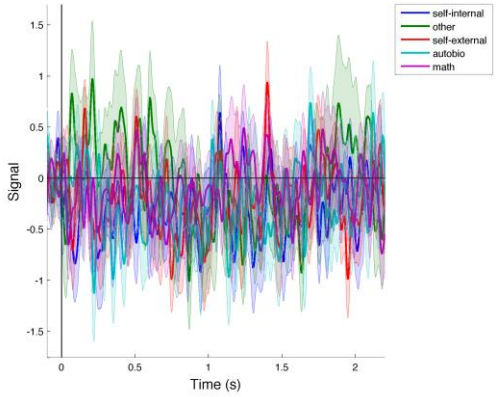
The smoothing routine extracts the full time window, convolves it with the Gaussian window, and then cuts the selected time window.

Important Note: As the edges are smoothed, the length of the considered smoothing window will greatly impact the obtained signal! The window used for epoching should hence be large enough to take this into account.

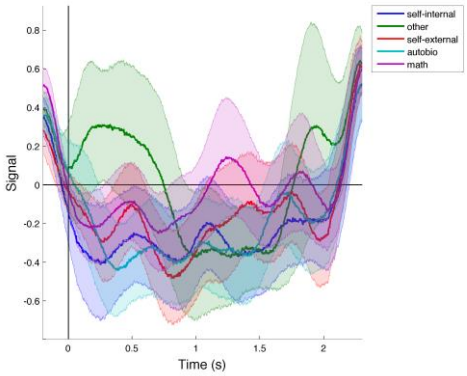
The following shows a single channel smoothed using the default options (i.e. window length of 50ms and full time window) and displayed with the LBCN_plot_averaged_signal_epochs function (described above):

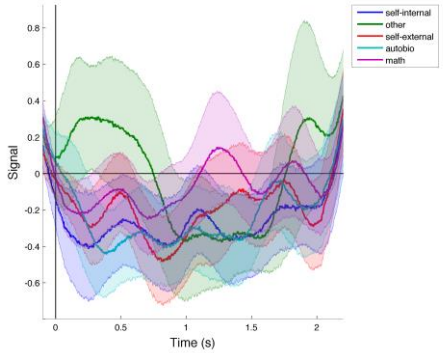


As you can see, the edge effects left over from TF decomposition are still apparent. You will need to specify an appropriate time window to remove these. For example, smoothing our channel with a 50ms window length but specifying a time window of [-100 2200] gives the following (recall the full time window is [-200 2300]):

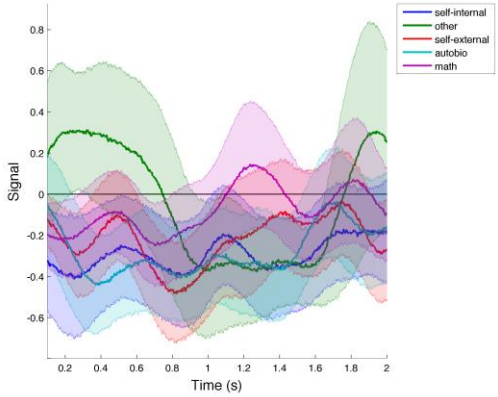


Using time windows of [-200 2300] and [-100 2200] but instead smoothing over a 500ms window will produce the following, respectively:





You can see that the edge effects have a more long-range effect when using a wider time window for smoothing. Thus, you will need to cut the time window even more to remove the edge effects. Still smoothing with a 500ms window but using a time window of [100 2000] gives us:



Running the pipeline for event-related HFB power

While the pipeline can be run by individually calling each function above, we provide an example function that performs all the steps. This should be seen as a gross first pass before digging more into e.g. bad channel detection.

Function: script_Event_Preprocessing_example
Description: Performs full pre-processing for task recordings.
Parameters:

1. filename (optional): in SPM format. Multiple files can be loaded together. They should however be multiple runs of the same experiment as they will be merged after TF rescaling and smoothing.
2. sodata (optional): Name of the behavioral .mat file. If none is specified, you will be prompted to choose a file via the SPM GUI. This file will be used to construct the events file. The events.categories file should be provided at this stage.
3. bch (optional): Indices, e.g. **`[2 32 5]`**, or names, e.g. **`{'TP8', 'TP10', 'OF1'}`**, of pathological channels to exclude. If none is specified, no channels will be excluded and the bad channels will be detected automatically only.

Output: Fully pre-processed data, in SPM format. The final files will contain a series of prefixes reflecting each of the pre-processing steps performed. See pre-processing step descriptions above for details.

The function first performs bad channel detection, calling the LBCN_filter_badchans function with the bch argument used in the original function call, after which it re-references the data with the LBCN_montage function, using the defaults. Next, the function epochs and baseline-corrects the data with the LBCN_epoch_bc function. It uses the defaults time windows and fields, that should definitely be modified (see code). After, the function uses the batch_ArtefactRejection_TFrescale_AvgFreq function to call the batch performing the time frequency decomposition steps. The data is then smoothed with the LBCN_smooth_data function. All sessions are then merged intone file. Finally, the LBCN_plot_averaged_signal_epochs function is used to plot the results for the selected time window, for all channels and conditions. The plots are saved with the prefix 'Onset_'. As before, you can make edits to the function or the functions/scripts it calls, as needed.

Continuous file pre-processing

When no event file is available, as is the case for rest periods, baseline correction and TF rescaling have to be performed using different strategies. In this section, we highlight the differences with the previously presented pipeline and provide potential answers for processing continuous files.

This pipeline includes the same steps for Notch filtering, bad channel detection and re-referencing of the signal than previously presented.

To discard trends in the raw signal, we need to baseline correct. As we don't really have a baseline, the following function can be used. This function also performs baseline correction for TF files. To discard artefacted time windows from being considered in the baseline, a first pass of artefact detection with the 'mark' option can be run (see an example in `batch_artefact_detect_continuous.m`).

Baseline correction on whole file

Function: LBCN_baseline_Timeseries
Description: Performs baseline correction in raw or TF signal.
Parameters:

1. fname (optional): in SPM format. Name of file to baseline correct
2. prefix (optional): Prefix for output file (by default, 'b' will be prepended).
3. method (optional): Method for baseline correction. The choices are 'average', 'logR' or 'Zscore'. The 'average' just subtracts the average over the specified time window, channel by channel, for each frequency bin (if TF). 'logR'and 'Zscore' are similar to the SPM methods of TF rescaling, but use the specified time window (a single window) instead of trial by trial baseline.
4. time_win (optional): Time window for baseline correction, since beginning of the file, in ms. Default is using the whole file [-Inf Inf].
5. fnamebase (optional): Name of additional file to use as baseline (e.g. use another rest period to correct the present file).

Output: Baseline corrected data, in SPM format, prepended with 'prefix'. Please note that this step can take quite some time, depending on the number of time points, channels and frequency bins.

IMPORTANT NOTE: For each channel, the artefacted windows will be discarded from the baseline. This means that the baseline will include different time points for each channel. Channels that have only artefacted windows will be baseline corrected using the whole time series. A warning will be displayed and we advise to mark that channel as 'bad'.

Hilbert continuous decomposition

The time frequency decomposition itself can be performed in SPM, without rescaling. We have however written a code that seems to be faster for Hilbert transform.

Function: LBCN_Time_frequency_Hilbert
Description: Performs Hilbert transform (based on FieldTrip).
Parameters:

1. fname (optional): in SPM format. Name of file to perform the TF on.

2. freq (optional): matrix of size [2, n_frequencies], specifying on the first row the first bin of the frequency band to extract, and on the second row the last frequency bin to extract. Default is [70; 170].
3. prefix (optional): Prefix for output file (by default, 'tf_' will be prepended).

Output: Hilbert amplitude in considered frequency bands, in SPM format, prepended with 'prefix'. Please note that this step can take quite some time, depending on the number of time points, channels and frequency bins. IMPORTANT: the frequency bins in the SPM MEEG object correspond to a column of the 'freq' input. If using the default band of 70;170, the corresponding frequency bin in the object is 1. In this other example:

```
freqs = [1 4 8 13 30 40 70; 3 7 12 29 39 69 170];
```

HFB corresponds to D(channel, 7, time points).

This routine can be used to perform a frequency decomposition using resolution bins. In this case, to extract HFB in bins of 10, you would specify:

```
freqs = [70 80 90 100 110 120 130 140 150 160; …
         79 89 99 109 119 129 139 149 159 170];
```

The SPM function 'average frequency' can then be used to compute the average of the signal across those frequency resolution bins.

The following function provides an example of pre-processing pipeline for continuous files. Please note that we have added an extra step to mark large peaks in Z-score (discarding peaks larger than 8 on each channel). Such peaks reflect artefact (due to increases in variability in the raw signal over a window of time) and should be discarded from further analyses (e.g. correlation between 2 channels).

Pipeline example for continuous files

Function: script_Continuous_Preprocessing_example

Description: Performs full pre-processing for continuous files.

Parameters:

1. filename (optional): in SPM format. Multiple files can be loaded together. They should however be multiple runs of the same experiment as they will be merged after TF rescaling and smoothing.
2. sodata (optional): Name of the behavioral .mat file. If none is specified, you will be prompted to choose a file via the SPM GUI. This file will be used to construct the events file. The events.categories file should be provided at this stage.
3. bch (optional): Indices, e.g. **`[2 32 5]`**, or names, e.g. **`{'TP8', 'TP10', 'OF1'}`**, of pathological channels to exclude. If none is specified, no channels will be excluded and the bad channels will be detected automatically only.

Output: Fully pre-processed data, in SPM format. The final files will contain a series of prefixes reflecting each of the pre-processing steps performed. See pre-processing step descriptions above for details.

The steps performed include: Notch filtering and detection of bad and spiky channels, re-referencing to average of all good channels, marking artefacts in raw signal (jumps>100muV and z-score diff>6, exclusion window of 1 second), baseline correcting the raw signal ('average' method), perform Hilbert transform for 7 frequency bands, baseline correct the TF using the 'Zscore' (baseline = whole time

series), extract the HFB signal. Another step can be added to identify peaks in Zscore. The final step smoothes the data with a 50ms window.

Univariate analyses for event-related

In this section, we present 3 routines that perform univariate analyses by looking at each channel separately and averaging the signal within a specific time window. These routines assume the data is not in TF format, i.e. it is D(channels, time points, trials). You can use the `batch_extract_HFB(fname,[],[freq1 freq2],'HFB');` batch to extract a specific band of frequency, as computed by our LBCN_Time_Frequency_Hilbert or use the spm 'average frequencies' option.

Note: the names of the conditions should match the epoched category names exactly.

Compare condition to baseline

Function: LBCN_Univariate_Condition_Baseline
Description: Performs univariate analysis, comparing the trials of one condition to their baseline, in a paired permutation test, FDR corrected. The signal is averaged in the time window specified by user, yielding one p-value per channel.
Parameters:

1. fname (optional): in SPM format. File to analyze, in raw or TF space, but with 3 dimensions only (otherwise the first frequency bin will be considered only).
2. cond: Condition to look at, in a string or cell array. Conditions entered in a cell array will be pooled together for this test.
3. twcond: Time window, in ms, to average the trial signal in. e.g. [100 1000].
4. twbas: Time window, in ms, to average the baseline in. e.g. [-200 0]
5. ichan: Indices, e.g. **`[2 32 5]`**, or names, e.g. **`{'TP8', 'TP10', 'OF1'}`**, of channels to perform the test on. If none is specified, no channels will be excluded. The string '**`good`**' can also be specified to perform the test on all channels, excluding channels marked as bad.
6. nperm: number of permutations to run (default: 10000).
7. suffix: string with suffix to be appended to the .mat result file
8. tail: flag set to 1 for testing condition>baseline, 0 for condition ≠ baseline. (default: 0)

Output: 'res' structure, saved in .mat on the path of the file name. The structure comprises:

```
res =

  struct with fields:

    sign_chans: {'iEEG_14'  'iEEG_15'  'iEEG_19'  'iEEG_20'}
    pchan_Cond: [26×1 double]
     pchan_WSR: [26×1 double]
         hCond: [26×1 logical]
    crit_pCond: 9.9990e-05
         ichan: [26×1 double]
```

The sign_chans field provides the labels of channels showing significant differences between condition and baseline, after FDR correction at 0.05. pchan_Cond provides the p-value for each channel computed from permutations (used for significance assessment). pchan_WSR provides the p-value computed from Wilcoxon Signed Rank test, for information/comparison. hcond is a logical vector, with 0 suggesting that the null hypothesis (after FDR correction) could not be rejected for that channel, 1 for significant result. crit_pCond is the threshold for pchan_Cond to be considered significant (FDR corrected p of 0.05) and ichan simply represents the indices of the channels considered in the test, for reference.

Example: assessing visual responses to all types of visual stimuli presented:

```
[res] =
LBCN_Univariate_Condition_Baseline(fname2,{'Animals','Faces','Buildings'},[15
0 800],[-200 0],label,10000,'Learn',1);
```

Compare two conditions

Function: LBCN_Univariate_Condition_Compare

Description: Performs univariate analysis, comparing the trials of one condition to the trials in another condition, in a permutation test, FDR corrected. The signal is averaged in the time window specified by user, yielding one p-value per channel.

Parameters:

1. fname (optional): in SPM format. File to analyze, in raw or TF space, but with 3 dimensions only (otherwise the first frequency bin will be considered only).
2. cond: Condition to look at, in a string or cell array. Conditions entered in a cell array will be pooled together for this test.
3. twcond: Time window, in ms, to average the trial signal in. e.g. [100 1000].
4. cond2: Condition to compare it to, in a string or cell array. Conditions entered in a cell array will be pooled together for this test.
5. twcond2: Time window, in ms, to average the trial signal of condition 2 in. e.g. [100 1000].
6. ichan: Indices, e.g. **[2 32 5]**, or names, e.g. **{'TP8', 'TP10', 'OF1'}**, of channels to perform the test on. If none is specified, no channels will be excluded. The string '**good**' can also be specified to perform the test on all channels, excluding channels marked as bad.
7. nperm: number of permutations to run (default: 10000).
8. suffix: string with suffix to be appended to the .mat result file
9. tail: flag set to 1 for testing condition 1> condition 2, 0 for condition 1 ≠ condition 2. (default: 0)

Output: 'res' structure, saved in .mat on the path of the file name. The structure has the same fields as when comparing a condition to its baseline.

IMPORTANT: if multiple conditions are entered in a cell array, they will be pooled together.

Example: assessing face selectivity in the [150 to 800]ms after stimulus onset.

```
[res] = LBCN_Univariate_Condition_Compare(D,{'face'},[150
800],{'animal','building'},[150 800],label, 10000,'Face_sel_',1);

res =

  struct with fields:

    sign_chans: {'iEEG_14'  'iEEG_15'  'iEEG_20'  'iEEG_21'  'iEEG_24'}
    pchan_Cond: [26×1 double]
     pchan_WSR: [26×1 double]
         hCond: [26×1 logical]
   crit_pCond: 5.9994e-04
         ichan: [26×1 double]
        values: [26×1 double]
```

Compare two conditions in binary tests

In this case, each category will be compared pairwise to all other categories and the list of channels showing significant effects in ALL binary comparisons will be returned. This function allows to test for

selectivity in a stringent manner, ensuring that e.g. condition A is significantly larger than conditions B, C and D.

Function: LBCN_Univariate_Condition_Compare_Bin
Description: Performs univariate analysis, comparing the trials of one condition to the trials in another condition, in a permutation test, FDR corrected. The signal is averaged in the time window specified by user, yielding one p-value per channel.
Parameters:

1. fname (optional): in SPM format. File to analyze, in raw or TF space, but with 3 dimensions only (otherwise the first frequency bin will be considered only).
2. cond: Condition to look at, in a string or cell array. Conditions entered in a cell array will be compared one by one to the conditions specified in cond2.
3. twcond: Time window, in ms, to average the trial signal in. e.g. [100 1000].
4. cond2: Condition to compare it to, in a string or cell array. Conditions entered in a cell array will be compared one by one to the conditions specified in cond.
5. twcond2: Time window, in ms, to average the trial signal of condition 2 in. e.g. [100 1000].
6. ichan: Indices, e.g. **[2 32 5]**, or names, e.g. **{'TP8', 'TP10', 'OF1'}**, of channels to perform the test on. If none is specified, no channels will be excluded. The string '**good**' can also be specified to perform the test on all channels, excluding channels marked as bad.
7. nperm: number of permutations to run (default: 10000).
8. suffix: string with suffix to be appended to the .mat result file
9. tail: flag set to 1 for testing condition 1> condition 2, 0 for condition 1 ≠ condition 2. (default: 0)

Output: 'common_list' structure, saved in .mat on the path of the file name. It comprises the list of channels that show a significant effect in all binary comparisons. The results of each binary comparison is saved in the 'res' file. That structure has the same fields as when comparing a condition to its baseline or a condition to another.

IMPORTANT: if multiple conditions are entered in a cell array, they will NOT be pooled together, but all binary comparisons will be performed.

Example: assessing face selectivity in the [150 to 800]ms after stimulus onset. This yields different results from the example above!

```
[res] = LBCN_Univariate_Condition_Compare_Bin(D,{'face'},[150
800],{'animal','building'},[150 800],label, 10000,'Face_sel_',1);

res =

  struct with fields:

    sign_chans: {'iEEG_20'  'iEEG_21'}
    pchan_Cond: [26×1 double]
     pchan_WSR: [26×1 double]
         hCond: [26×1 logical]
    crit_pCond: 5.9994e-04
         ichan: [26×1 double]
        values: [26×1 double]
```