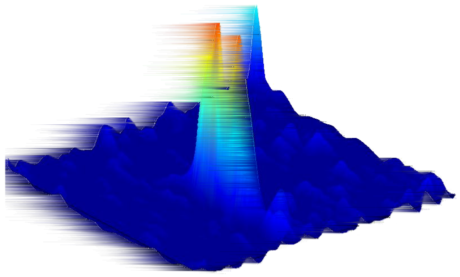


Fast_TF

Fast MEG/EEG Time-Frequency Analysis

-

VERSION: 5.3.0

CRICM - CENIR - Pitié-Salpêtrière - France

February 1, 2013

DO NOT USE THIS SOFTWARE FOR CLINICAL APPLICATION

Contents

1 Introduction

1.1 What's *Fast_TF* ?

Fast_TF is a standalone Linux software package dedicated to time-frequency analysis of MEG/EEG signals. It can compute power maps, z score maps, phase locking factor maps, coherence, and synchrony maps. This software results from a collaboration between Pitié-Salpêtrière MEG/EEG Center and COGIMAGE team from CRICM.
Principal contributors to this version:
Time frequency analysis core programming: C. Tallon-Baudery - D. Schwartz
Original wavelet algorithm: O. Bertrand (Lyon U821 INSERM)
I/O libraries: J-D. Lemaréchal and A. Ducorps **Matlab routine**: L. Yahia-Cherif, A. Barbot, M. Chaumon, D. Schwartz

1.1.1 Acknowledgments

We would like to thanks here all the people who contributed to this work as "pre-alpha" testers.

1.2 Copyright and Disclaimer

Fast_TF is distributed under GPL, so it's free software, in the sense intended by FSF (Please take a look at the complete license at the end of this document).

2 Installing *Fast_TF* program

2.1 Pre-Requirements

Fast_TF was developed under Linux using CentOs 5.8 distribution.

Supported systems: ***Fast_TF*** is available for Linux on Intel386 PC (both 32 bits and 64 bits are available) and MacOs X (compiled on 10.6 MacOS X system)

Needed softwares: ***Fast_TF*** is a standalone program and doesn't need any additional software. Visualization of time-frequency maps needs Muse, Matlab or a similar tool (necessary visualization functions are provided for matlab).

2.2 Downloading the Package

The package is available on the following web site:
http://cogimage.dsi.cnrs.fr/logiciels. Click on the download link.

2.3 Unpacking the Software

Execute: `tar -xzvf fast_tf.tgz`

2.4 Installing the software

Execute: `sh fast_tf/setup.sh`

Package is installed, you can now call ***Fast_TF*** : `fast_tf/bin/fast_tf --help`

3 General presentation

Fast_TF is command line program dedicated to time frequency analysis of MEG/EEG signals. Using wavelets, it computes time frequency maps in term of power, z score relative to a given baseline, phase, phase locking factor, synchrony between sensors across trials and synchrony between sensors on a window of time. New to version 4.5, ***Fast_TF*** also computes coherence between sensors. ***Fast_TF*** takes advantage of FFTW 3.1 library to offer very fast computation and multi-processing capabilities. This version can read CTF/VSM ds files, Elekta-Neuromag fiff files, LENA files and ASCII data. It generates LENA files readable in matlab or Muse. Matlab functions to read and to plot time-frequency maps are provided. This documentation begins with a quick start guide for impatient users followed by methodological considerations. A complete description of all options is available in the second part of the manual.

4 Quick start guide

In this section you will find examples of ***Fast_TF*** invokation and a short description of maps visualization in matlab.

4.1 evoked

Adding –evoked option to the command lines described below will allow computation of evoked/field potential corresponding to the averaging of all occurences of specified markers with respect to the analysis window.

4.2 Power

The following command line computes mean time frequency maps in term of power of MEG channels, across all occurences of marker AF in run01.ds and run02.ds.

```
fast_tf --power --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti --meg \
--marker AF --begin_analysis -0.7 --end_analysis 1.5 \
--input_files ‘‘test_tf/run01.ds test_tf/run02.ds’’
```

4.3 Mean power

The following command line computes mean power in the given time-frequency window. Values are saved fo each occurence of marker AF in run01.ds and run02.ds.

```
fast_tf --mean_power --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti --meg \
--marker AF --begin_analysis -0.7 --end_analysis 1.5 --begin_tfwin_time -0.2\
--end_tfwin_time 0.5 --begin_tfwin_freq 30 --end_tfwin_freq 70 \
--input_files ‘‘test_tf/run01.ds test_tf/run02.ds’’
```

4.4 Z score

The following command line computes z scores time frequency maps of EEG channels, across all occurences of marker AF in run01.ds and run02.ds.

```
fast_tf --z_score --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti --begin_baseline -0.5 \
--end_baseline -0.2 --marker AF --begin_analysis -0.7 --end_analysis 1.5 \
--input_files ‘‘test_tf/run01.ds test_tf/run02.ds’’ --eeg
```

4.5 Mean Z score

The following command line computes mean z score in the given time-frequency window. Values are saved fo each occurence of marker AF in run01.ds and run02.ds.

```
fast_tf --mean_z_score --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti --meg \
--marker AF --begin_analysis -0.7 --end_analysis 1.5 --begin_tfwin_time -0.2\
--end_tfwin_time 0.5 --begin_tfwin_freq 30 --end_tfwin_freq 70 \
--input_files ‘‘test_tf/run01.ds --input_files test_tf/run02.ds’’
```

4.6 Phase locking factor

The following command line computes phase locking factor of both MEG and EEG channels accross all occurences of marker AF in run01.ds and run02.ds.

```
fast_tf --phase_lock --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti --begin_baseline -0.5 \
--end_baseline -0.2 --marker AF --begin_analysis -0.7 --end_analysis 1.5 \
--input_files ‘‘test_tf/run01.ds test_tf/run02.ds’’ --eeg --meg
```

4.7 Phase

The following command line computes phase of both MEG and EEG channels. Values are saved fo each occurence of marker AF in run01.ds and run02.ds.

```
fast_tf --phase --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti --begin_baseline -0.5 \
--end_baseline -0.2 --marker AF --begin_analysis -0.7 --end_analysis 1.5 \
--input_files ‘‘test_tf/run01.ds test_tf/run02.ds’’ --eeg --meg
```

4.8 Synchrony

Fast_TF can compute measures of synchrony between pairs of sensors :

4.8.1 Pairs file format

Pairs of sensors are described by an ASCII file with the following format:
First line contains sensor labels separated by space. The file should contain n lines with a sensor label and n numbers (0 or 1) separated by space where n is the number of sensors if the file to be analyzed. If you want to compute synchrony between sensor i and j you should put a 1 at line i column j and a zero otherwise. Here is a very simple example:

$$\begin{array}{cccccc} & y_1 & y_2 & y_3 & y_4 & y_5 \\ y_1 & 0 & 1 & 0 & 0 & 1 \\ y_2 & 0 & 0 & 1 & 0 & 0 \\ y_3 & 0 & 0 & 0 & 0 & 1 \\ y_4 & 0 & 0 & 0 & 0 & 0 \\ y_5 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Synchrony will be computed between the following pairs of sensors: (y_1,y_2) (y_1,y_5) (y_2,y_3) (y_3,y_5). The following command line averages synchrony for all occurences of marker AF in run01.ds and run02.ds.

```
fast_tf --sync_trial --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti --pairs test_tf/pairs.txt\
--marker AF --begin_analysis -0.7 --end_analysis 1.5 \
--input_files ‘‘test_tf/run01.ds test_tf/run02.ds’’
```

For computation involving a lot of pairs of sensors and long duration (more than 100 pairs and more than 1000 samples) you should use smp_fast_tf.py to speed up computation and save disk space during computation.

4.9 Mean Synchrony along time

The following command line averages synchrony between .2 second before and 0.5 second after each occurence of marker AF in run01.ds and run02.ds.

```
fast_tf --sync_time --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti --pairs test_tf/pairs.txt\
--marker AF --begin_analysis -0.7 --end_analysis 1.5 --time_synchrony_begin -0.2\
--time_synchrony_end 0.5 --input_files ‘‘test_tf/run01.ds test_tf/run02.ds’’
```

4.10 Coherence

The following command line computes coherence between the following pairs of sensors: (y_1,y_2) (y_1,y_5) (y_2,y_3) (y_3,y_5) for all occurences of marker AF in run01.ds and run02.ds.(see section 4.8 or 11.3 for description of the file describing the pairs of sensors)

```
fast_tf --coherence --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti --pairs test_tf/pairs.txt\
--marker AF --begin_analysis -0.7 --end_analysis 1.5 \
--input_files ‘‘test_tf/run01.ds test_tf/run02.ds’’
```

4.11 Multi-modes computation

You can computes everything at once by specifying all the computation modes you want:

```
fast_tf --evoked --power --phase_lock --sync_time --sync_trial \
--first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti --pairs test_tf/pairs.txt\
--marker AF --begin_analysis -0.7 --end_analysis 1.5 --time_synchrony_begin -0.2\
--time_synchrony_end 0.5 --input_files ‘‘test_tf/run01.ds test_tf/run02.ds’’
```

4.12 Visualization with matlab

This code was tested on version 2007a:

Add to your matlab path the following directory and sub directories :

```
[path_to_your_fast_tf_installation]/matlab
```

Then execute the following commands to display the time frequency map stored in test_power.lena for channnel MLC12 :

```
> [tf, v_t, v_f, channels]=read_lena('test_power.lena') ;
> h=plot_tf(v_t, v_f, tf, channels, '{MLC12}', '0') ;
```

4.13 Visualization with Muse

Muse is linux visualization tool developped by the LENA-UPR640 lab. It read natively the output of ***Fast_TF*** and can display the data in numerous ways such as time frequency maps, curves, 2D and 3D cartography. Currently it is available at the MEG/EEG Center in Paris and available upon request if you need it.

5 Methodological aspects

5.1 Step by step time frequency analysis

5.1.1 Short non mathematical description of time-frequency analysis

Let's imagine you have a MEG sensor with one unique channel and you just recorded your first motor evoked field. Now you want to determine the frequency composition along the time of your signal. A simple

FFT (Fast Fourrier Transform) will give you a global view of the frequencies components with no readily available temporal information. Using wavelets, we will be able to retrieve this temporal information, at least to a certain degree. It means that you will be able to localize in time the variations of power for each frequency you want to study. See figure 1 for a graphical description of the method we used.

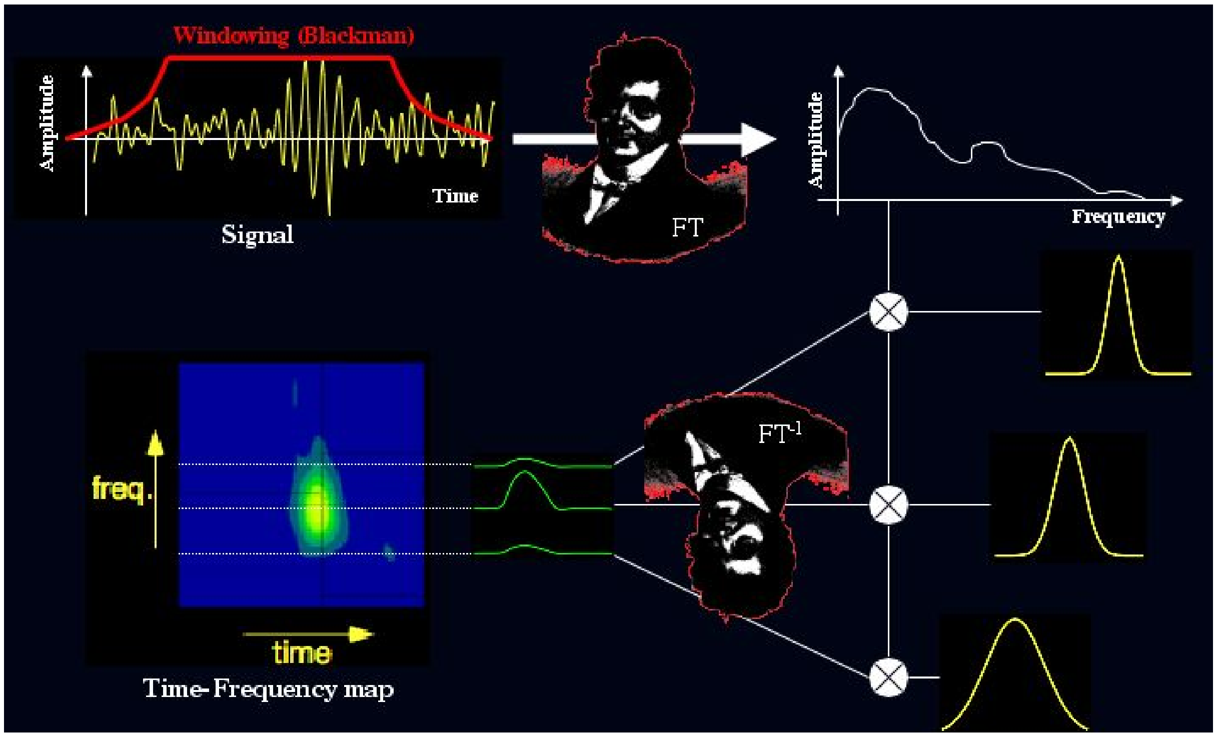


Figure 1: Framework of the method

Fast_TF starts with an FFT of your MEG signal. Then comes the famous mathematical object "wavelet". For each frequency you want to study (for example 30Hz) we construct a wavelet (see equation 1). This wavelet is then used to extract the contribution of the frequency 30Hz to your original signal from the result of the FFT. This operation is called convolution (of the wavelet and the FFT result). The result of this operation is the contribution of 30Hz to your signal in the frequency domain. Hence we just have to do an inverse FFT of this result to come back to the temporal domain and obtain the contribution of 30Hz to your signal along the time. Repeat that for all your wanted frequencies and you will obtain a time-frequency map.

Here is a short description of the algorithm:

- Load the signal
- Load the parameters (wavelet parameter m, window of analysis, frequencies list ...)
- Build the wavelets according to m and frequencies list

- Do a FFT of your signal
- For each frequency convolve the results of the FFT and the wavelet associated to the frequency
- Compute an inverse FFT of the results of the convolution

From the basic time-frequency maps obtained we can compute z-scores, phase-locking factor and several measures of synchronies between pairs of sensors.

5.1.2 Simple examples

This section shows time-frequency analysis of very simple signals to illustrate typical maps you will obtain with ***Fast_TF*** . Shown on figure 2 are 7 sensors $y1$, $y2$, $y3$, $y4$, $y5$, $y6b$ and $y7b$ with various temporal behaviors. Normally distributed random noise was added to the last two ones.

Figures 3, 4 and 5 show time-frequency maps for each sensor in term of power (see equation 10) for three different values (5, 10 and 20) of the wavelets parameter m.

We can see that when m increases frequency resolution increase and temporal resolution decreases as illustrated on figure 6.

5.2 Algorithm design and theoretical background

Let s be the spatio temporal matrix containing the MEEG signals on N_s sensors (rows) on N_t time steps (columns). Let SR be the sampling rate of signal. Let f be a vector of N_f frequencies for which we want to compute the energy.

The wavelets used in the following computation are defined as follows in the frequency domain:

$$W_i(k) = \alpha e^{(-(k*\delta f - f_i)^2/\sigma_{f_i})} \tag{1}$$

where $i = 1 \ldots N_f,\ k = min_{f_i} \ldots max_{f_i}$, $\delta f = SR/N_t$, $\sigma_{f_i} = f_i/m$ and $\alpha = \sqrt{\frac{10^3}{N_t.\sigma_{f_i}.\sqrt{\pi}}}$. min_{f_i} and max_{f_i} are defined as follows:

$$min_{f_i} = \frac{(f_i - 4\sigma_{f_i}) * N_t}{SR} \tag{2}$$

$$max_{f_i} = \frac{(f_i + 4\sigma_{f_i}) * N_t}{SR} \tag{3}$$

As a first step we compute the Discrete Fourier Transform of the signal F_c for sensor c:

$$F_c = \sum_{k=0}^{N_t-1} s_c e^{2\pi ikn/N} \tag{4}$$

where $i = 1 \ldots N_t$

F_c contains the real part F_c^r and the imaginary part F_c^i of the results of the FFT.

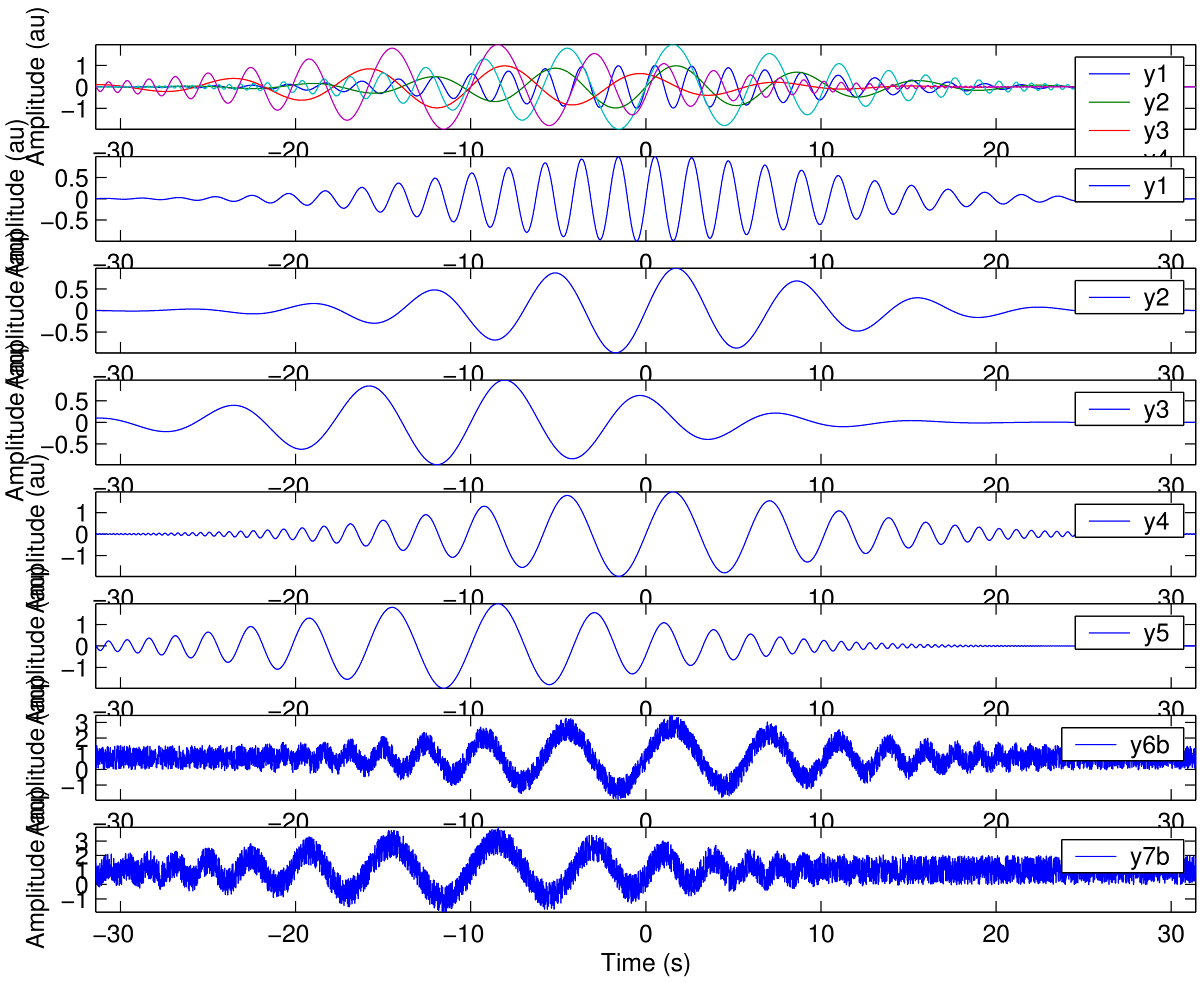


Figure 2: Tests signals

Each wavelets is then convoluted with F_c:

$$Fwav_c^i = F_c \otimes W_i \tag{5}$$

where $i = 1 \ldots N_f$. $Fwav_c^i$ is the contribution of frequency f_i to s in the frequency domain. Hence, to obtain the contribution of f_i to s in the time domain (i.e. the time-frequency map for f_i) we compute the inverse Fourrier Transform of $Fwav_c^i$:

$$fmap_c = \frac{1}{N_t} \sum_{k=0}^{N_t-1} Fwav_c e^{-2\pi ikn/N} \tag{6}$$

where $i = 1 \ldots N_t$. $fmap$ represents the real part $fmap^r$ and the imaginary part $fmap^i$ of the energy for each frequency in f along the time.

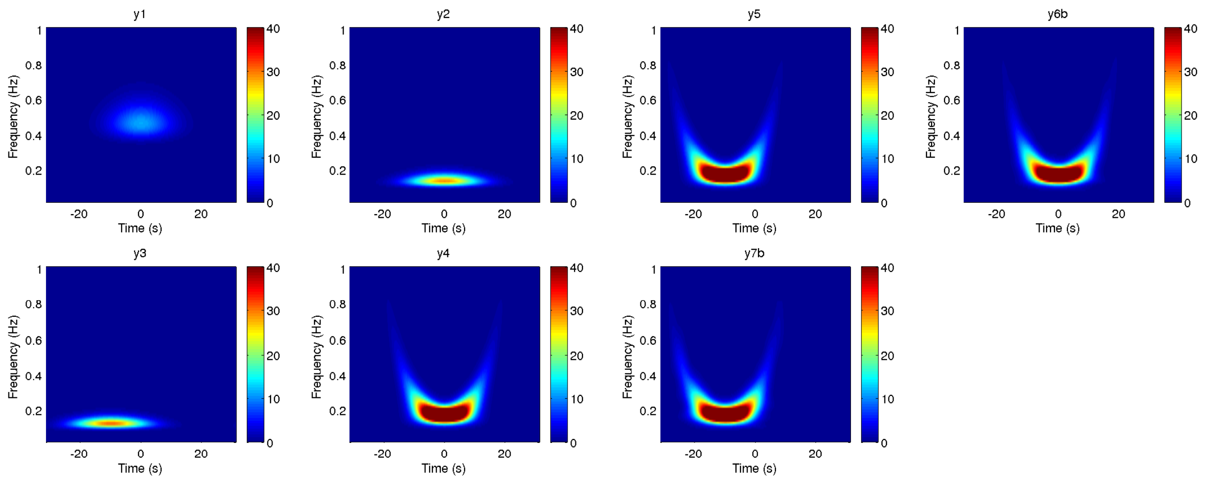


Figure 3: Time-frequency maps (Power) $m = 5$

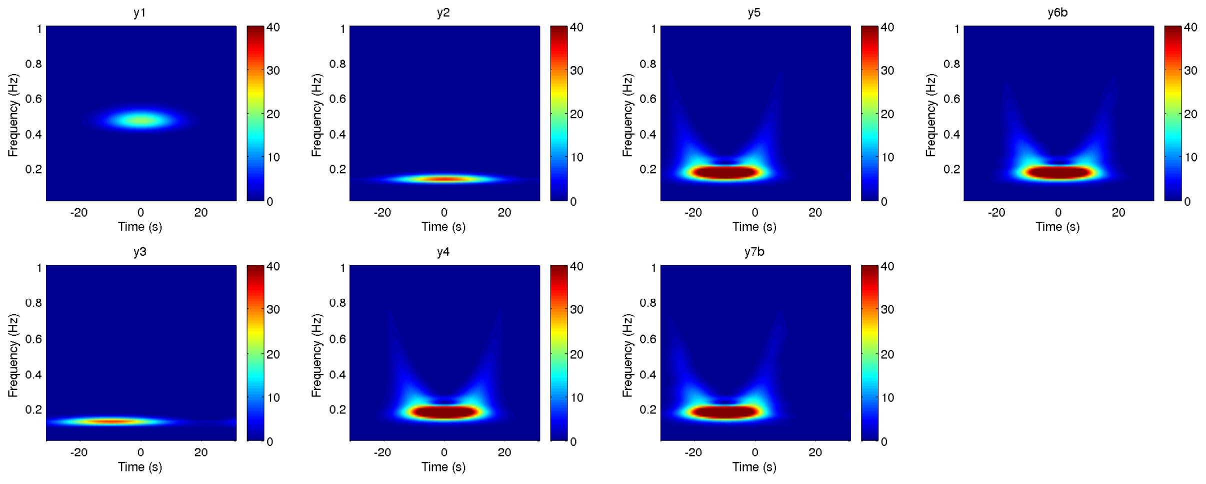


Figure 4: Time-frequency maps (Power) $m = 10$

To end this part let us define the power for frequency f_i for sensor c at time t :

$$F_p = ((fmap_j^r)^2 + (fmap_j^i)^2) \tag{7}$$

5.3 Frequential and temporal resolution of the time-frequency map

Temporal and frequential resolutions at a given frequency are link to m parameter:

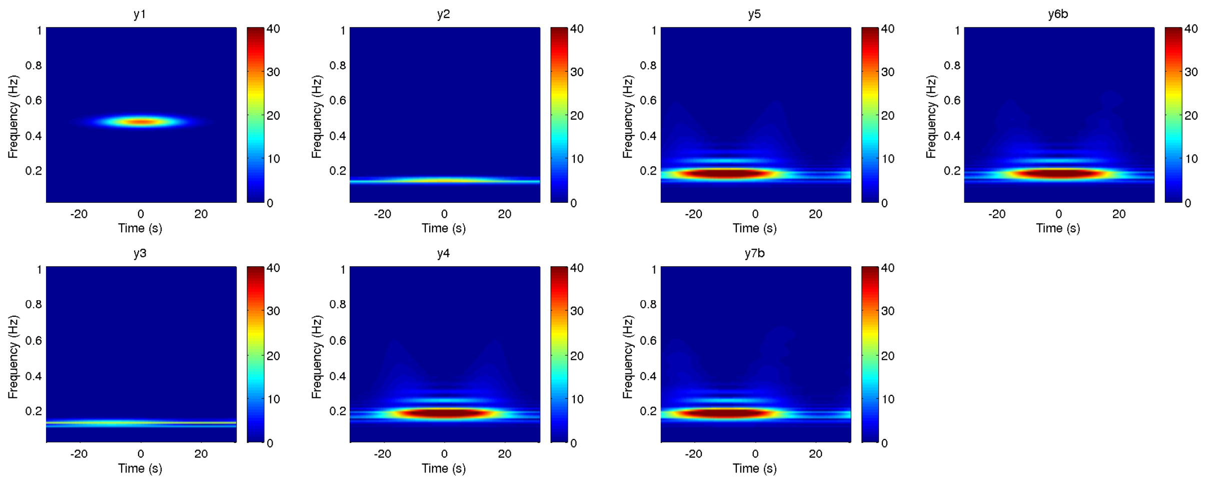
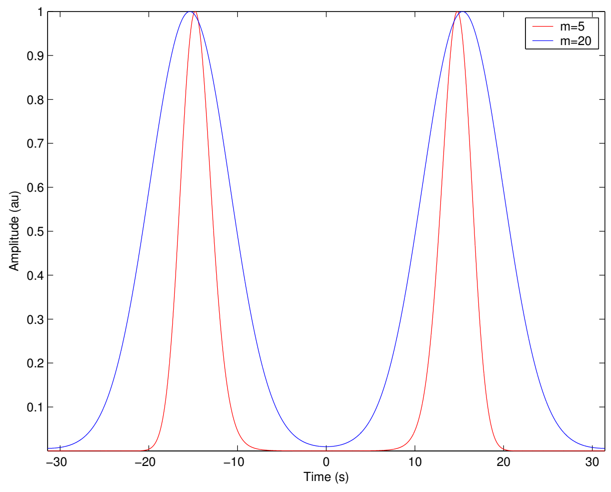


Figure 5: Time-frequency maps (Power) $m = 20$



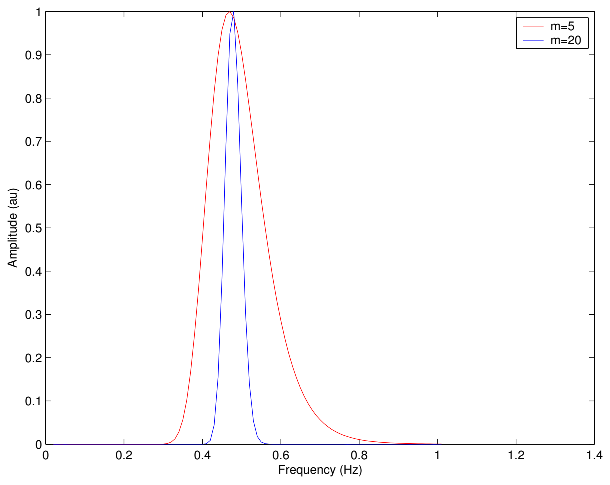


Figure 6: left: sensor y_1 at 0s for $m = 5$ and $m = 20$ - Right: sensor y_4 at 0.5Hz for $m = 5$ and $m = 20$

Frequential resolution at frequency f is given by:

$$\boxed{\sigma_f = \frac{f}{m}} \tag{8}$$

Temporal resolution at frequency f is given by:

$$\boxed{\sigma_t = \frac{m}{2*\pi*\sigma_f}} \tag{9}$$

6 General options

6.1 Verbose mode

By default ***Fast_TF*** is mute but if you turn verbose mode on it becomes rather very communicative.

–verbose: Turn on verbose mode

6.2 Rewrite mode

By default ***Fast_TF*** does not overwrite a file, you can allow it to do that.

–rewrite: Allow overwriting of existing output file.

6.3 Marker name

Tells to ***Fast_TF*** which marker to select (all time specified will be relative to this marker)

–marker: Name of marker (between quotes if several markers are specified)

6.4 MEG channels selection

Tells to ***Fast_TF*** to select for analysis all MEG channel found in the input files (default)

–meg: Select meg channels

6.5 EEG channels selection

Tells to ***Fast_TF*** to select for analysis all EEG channel found in the input files

–eeg: Select eeg channels

6.6 MEG & EEG channels selection

Tells to ***Fast_TF*** to select for analysis all MEG & EEG channel found in the input files

–meeg: Select meg and eeg channels

6.7 Channel selection

Tells to ***Fast_TF*** to select for analysis the given list of channels

–channels: Select the given channel (between quotes if several channels are specified)

Here is an example of the command line which tells to ***Fast_TF*** to select MRC21 EEG and MLT. It means that MRC21 and all channels with a label containing either EEG or MLT will be selected. If one channel doesn't exist it is ignored.

```
fast_tf --power --first_frequency 0.02 --last_frequency 3 --frequency_step 0.01
--blackman_win 1.0 --wavelet_m 20 --begin_baseline -31 --end_baseline -25 \
--begin_analysis -31 --end_analysis 31 --verbose --rewrite --output_file \
tf_curves20 --input_files test_tf/run01.ds --channels ‘‘MRC21 EEG MLT’’
```

6.8 Strict channel name matching

Insure that channel name specified with –channels strictly matches a channel name is the analyzed file. This option is usefull if a set of channel names contains similar pattern and you want to analyze only one of them (bipolar channels for example).

–strict_channel_name: Srict match enforced between wanted channels and channels list

Here is an example of the command line which tells to ***Fast_TF*** to select MRC21, MRC22 and MLT12 with strict channel name matching. It means that only channels named MRC21, MLT12 and MRC22 will be analyzed. A channel named MRC21_MLT12 will be ignored.

```
fast_tf --power --first_frequency 0.02 --last_frequency 3 --frequency_step 0.01
--blackman_win 1.0 --wavelet_m 20 --begin_baseline -31 --end_baseline -25 \
--begin_analysis -31 --end_analysis 31 --verbose --rewrite --output_file --strict_channel_name\
tf_curves20 --input_files test_tf/run01.ds --channels ‘‘MRC21 MRC22 MLT12’’
```

7 Window of analysis - Time parameters

Fast_TF perfoms its analysis with respect to occurences of one or several markers (triggers). All the timing are defined with respect to these markers. The figure 7 gives a graphical view of all the parameters.

7.1 Window of analysis

The options –begin_window_of_analysis and –end_window_of_analysis define the time window with respect to the specified markers where the time frequency computation will be done.

7.2 Blackman window

The option –blackman_window define the time window used to bring the signals to zeros at the beginning AND the end the window of analysis. It means that these two segments of signals will not contain any meaningfull data of the time-frequency maps. This is necessary to avoid fft artefacts.

7.3 Baseline window

The options –begin_baseline and –end_baseline define the time window (with respect to the specified markers) where the baseline values (mean and variance) will be computed. Other options for the baseline are described in paragraph 11.1.

8 Input / Output files specification

8.1 Files inputs

By default ***Fast_TF*** reads CTF “ds” files or LENA “minf” specified by option -input_files.

–input_files: Path and names of input files (extension must be specified, either “.fiff”, “.ds” or “.lena”, if several files are specified, put them between quotes”

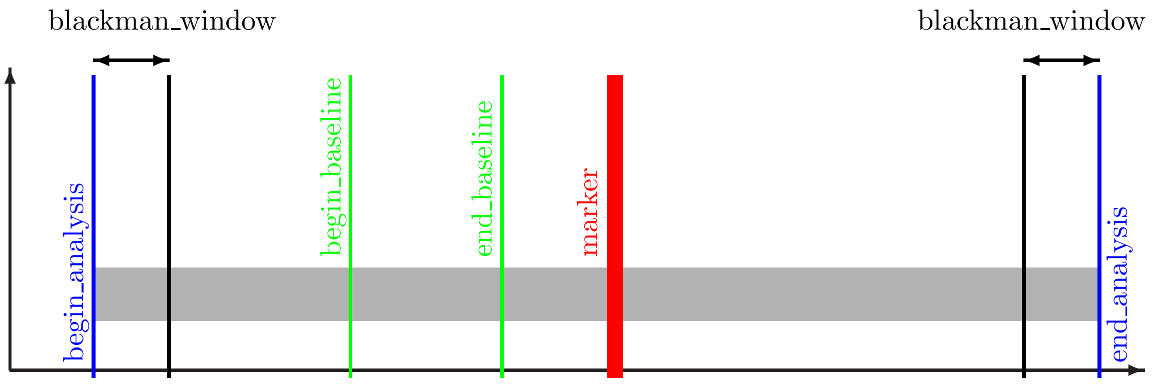


Figure 7: Graphical example of the timing definition in ***Fast_TF***

8.2 Using standard input

Fast_TF can also read what is called "standard input" in Unix dialect.

–stdin: Turn on stdin mode, i.e. read data from stdin

Here is an example of the corresponding command line which tells to ***Fast_TF*** to compute power while reading stdin input (here curves.txt file).

```
fast_tf --power --first_frequency 0.02 --last_frequency 3 --frequency_step 0.01
--blackman_win 1.0 --wavelet_m 20 --begin_baseline -31 --end_baseline -25 \
--begin_analysis -31 --end_analysis 31 --verbose --rewrite --output_file \
tf_curves20 --stdin < curves.txt
```

ASCII data on the standard input should have the following format:

```
ascii
Time [Numbers of times] [Time steps separated by space]
Trials [Number of trials]
Channels [Number of channels] [Names of Channels separated by space]
[data: trial -> sensor -> time, 1 line = 1 sensor, values separated by space]
```

Exemple:

```
ascii
Time 10 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Trials 3
Channels 2 EEG1 EEG2
8.1 9.7 5.3 3.7 2.4 1.4 2.7 8.8 9.9 10
7.7 6.7 5.8 8.4 9.4 9.8 5.3 2.5 1.9 4.5
9.5 3.8 2.8 1.1 7.4 3.4 8.4 4.4 8.1 2.8
8.7 7.9 6.4 3.5 0.4 7.8 5.3 4.8 6.4 6.3
8.5 5.5 2.9 3.4 9.1 5.4 8.8 7.5 6.8 7.4
2.7 3.9 0.7 8.1 6.8 7.3 8.7 4.1 3.4 1.5
```

8.3 Output file

Fast_TF generates "LENA" files (see above for matlab compatibility) with a postfix specifying the king of values stored in the file.

–output_file: Path and names of output file

Postfix values:

- Evoked signals: "evoked"
- Power: "power"
- Mean power: "mean_pow"
- Z score: "z_score"
- Mean Z score: "mean_zscore"
- Phase: "phase"
- Phase locking: "phase_lock"
- Synchrony on trial: "sync_trial" and "sync_trial_phase"
- Synchrony on time: "sync_time" and "sync_time_phase"
- Phase : "phase"
- Coherence on trial: "coherence"
- Coherence on time: "coherence_time"
- Mean power: "mean_power"
- Mean z_score : "mean_zscore"

The output of associated statistical modes have the same name plus "_stat".

9 Computation modes

9.1 Power

9.1.1 Purpose - Method

Compute basic time-frequency maps in term of power, by averaging, for each sensor time-frequency map f_p across N occurences of the markers (See figure 8):

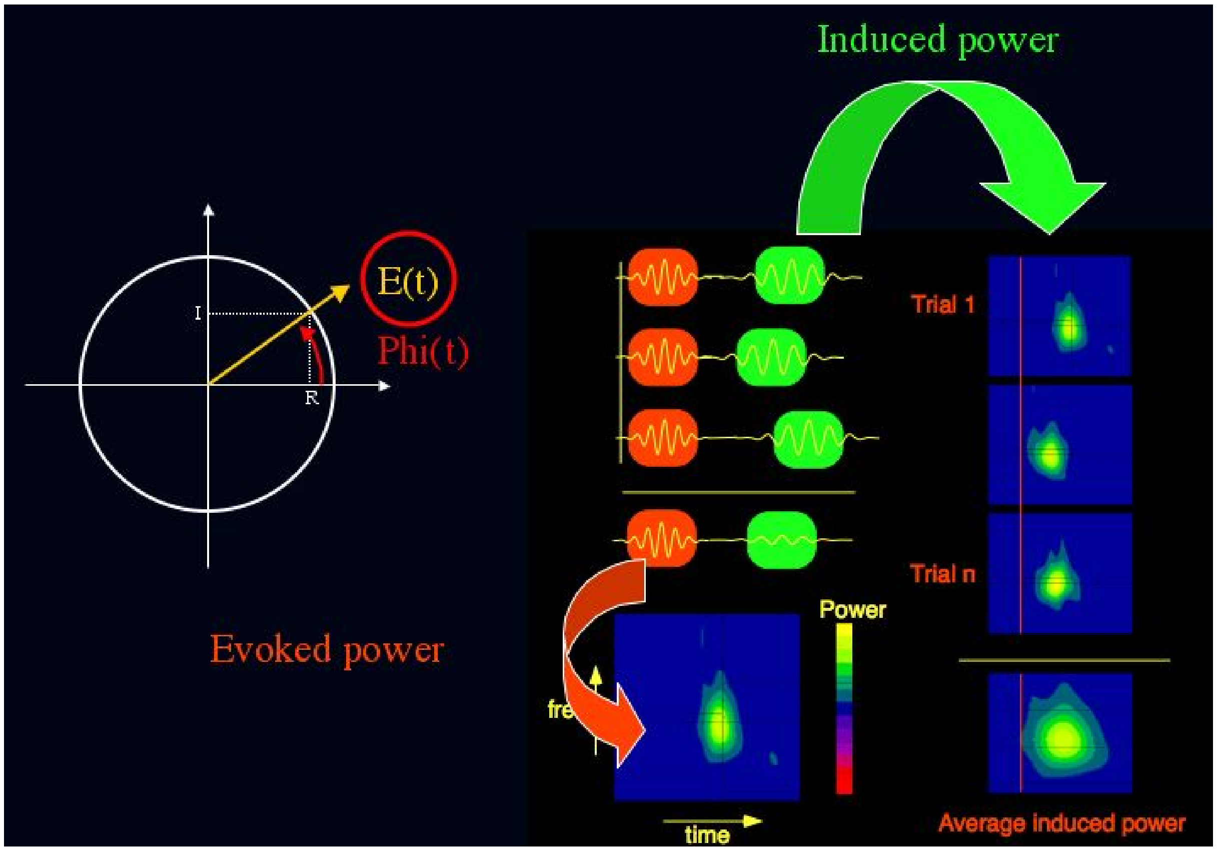


Figure 8: Power computation

$$p = \frac{\sum_{j=1}^{N} f_p^j}{N} \tag{10}$$

The end product is the variation of energy along the time for each frequency studied.

9.1.2 Input/Output Options

–power: Power computation mode
–first_frequency: First frequency of the time-frequency map
–last_frequency: Last frequency of the time-frequency map
–frequency_step: Frequency step of the time-frequency map
–blackman_win: Size of blackman window in seconds
–wavelet_m: Wavelets parameters m
–marker: Marker name
–begin_analysis:: Beginning of the analysis window in seconds
–end_analysis:: End of the analysis window in seconds
–output_file: Results file path
–input_files: Input files path

9.1.3 command line examples

The following command line computes mean time frequency maps of MEG channels, across all occurences of marker AF in run01.ds and run02.ds.

```
fast_tf --power --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti \
--marker AF --begin_analysis -0.7 --end_analysis 1.5 \
--input_files test_tf/run01.ds --input_files test_tf/run02.ds
```

9.1.4 Results

Fast_TF generates a LENA file with a postfix "power" containing the results.

9.1.5 Statistical mode

Non averaged values of power can be saved, trial by trial, by using –power_stat instead of –power. ***Fast_TF*** generates a LENA file with a postfix "powstat" containing the results.

9.2 Z score

9.2.1 Purpose - Method

Compute Z score maps by averaging for each sensor the time-frequency map f_p across all occurences of the markers. Each f_p is normalized by a mean value and by a standard deviation value computed on a baseline between "begin" seconds and "end" seconds with respect to a given marker: Mean baseline is computed as follow:

$$\bar{B}_p = \frac{\sum_{t=begin}^{t=end} f_p^t}{N_{bas}} \tag{11}$$

Standard deviation is then defined as follow:

$$\sigma_{\bar{B}_p} = \frac{\sqrt{\sum_{t=begin}^{t=end} (f_p^t - \bar{B}_p)^2}}{N_{bas}} \tag{12}$$

where N_{bas} is the number of samples in the baseline window.
Finally the z score map is computed as follow:

$$f_z = \frac{\sum_{i=1}^{N} \frac{f_p^i - \bar{B}_p^i}{\sigma_{B_p^i}}}{N} \tag{13}$$

where N is the number of occurences of the choosen marker. The end product is the variation of z-score along the time for each frequency studied.

9.2.2 Input/Output Options

–z_score: Z-score computation mode
–first_frequency: First frequency of the time-frequency map
–last_frequency: Last frequency of the time-frequency map
–frequency_step: Frequency step of the time-frequency map
–blackman_win: Size of blackman window in seconds
–wavelet_m: Wavelets parameters m
–marker: Marker name
–begin_analysis:: Beginning of the analysis window in seconds
–end_analysis:: End of the analysis window in seconds
–begin_baseline:: Beginning of the baseline in seconds
–end_baseline:: End of the baseline in seconds
–output_file: Results file path
–input_files: Input files path

9.2.3 command line examples

The following command line computes z scores time frequency maps of EEG channels, across all occurences of marker AF in run01.ds and run02.ds.

```
fast_tf --z_score --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti --begin_baseline -0.5 \
--end_baseline -0.2 --marker AF --begin_analysis -0.7 --end_analysis 1.5 \
--input_files test_tf/run01.ds --input_files test_tf/run02.ds
```

9.2.4 Results

Fast_TF generates a LENA file with a postfix “z_score” containing the results.

9.2.5 Statistical mode

Non averaged values of z_score can be saved trial by trial by using –z_score_stat instead of –z_score. ***Fast_TF*** generates a LENA file with a postfix "z_score_stat" containing the results.

9.3 Log ratio

9.3.1 Purpose - Method

Compute log_10 ratio maps by averaging for each sensor the time-frequency map f_p across all occurences of the markers. Each f_p is divided by a mean value computed on a baseline between "begin" seconds and "end" seconds with respect to a given marker: Mean baseline is computed as follow:

$$\bar{B}_p = \frac{\sum_{t=begin}^{t=end} f_p^t}{N_{bas}} \tag{14}$$

where N_{bas} is the number of samples in the baseline window.
Finally the log ratio map is computed as follow:

$$f_l = \frac{\sum_{i=1}^{N} log\left(\frac{f_p^i}{B_p^i}\right)}{N} \tag{15}$$

where N is the number of occurences of the choosen marker. The end product is a normalized time-frequency map with respect to your baseline. Note that your data are gaussian after this transformation (will ease use of standard statistical tests afterward).

9.3.2 Input/Output Options

–log: log ratio computation mode
–first_frequency: First frequency of the time-frequency map
–last_frequency: Last frequency of the time-frequency map
–frequency_step: Frequency step of the time-frequency map
–blackman_win: Size of blackman window in seconds
–wavelet_m: Wavelets parameters m
–marker: Marker name
–begin_analysis:: Beginning of the analysis window in seconds
–end_analysis:: End of the analysis window in seconds
–begin_baseline:: Beginning of the baseline in seconds
–end_baseline:: End of the baseline in seconds
–output_file: Results file path
–input_files: Input files path

9.3.3 command line examples

The following command line computes log ratio time frequency maps of EEG channels, across all occurences of marker AF in run01.ds and run02.ds.

```
fast_tf --log --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti --begin_baseline -0.5 \
--end_baseline -0.2 --marker AF --begin_analysis -0.7 --end_analysis 1.5 \
--input_files test_tf/run01.ds --input_files test_tf/run02.ds
```

9.3.4 Results

Fast_TF generates a LENA file with a postfix "log" containing the results.

9.3.5 Statistical mode

Non averaged values of log ratio can be saved trial by trial by using –log_stat instead of –log. ***Fast_TF*** generates a LENA file with a postfix "log_stat" containing the results.

9.4 Phase Locking

9.4.1 Purpose - Method

Compute phase locking factor for each channel across N trials : normalized real and imaginary part of $Fmap$ are sum across trials :

$$\bar{f^r} = \sum_{t=1}^{t=N} \frac{fmap_t^r}{\sqrt{f_p^t}} \tag{16}$$

$$\bar{f^i} = \sum_{t=1}^{t=N} \frac{fmap_t^i}{\sqrt{f_p^t}} \tag{17}$$

Then phase locking factor is computed as follows:

$$l = \sqrt{\frac{\bar{f^r}^2 + \bar{f^i}^2}{N^2}} \tag{18}$$

The phase locking factor gives you an estimation of phase stability for each frequency you selected (See figure (See figure 9) . You should be able to detect phase reseting at the beginning of an evoked potential for example.

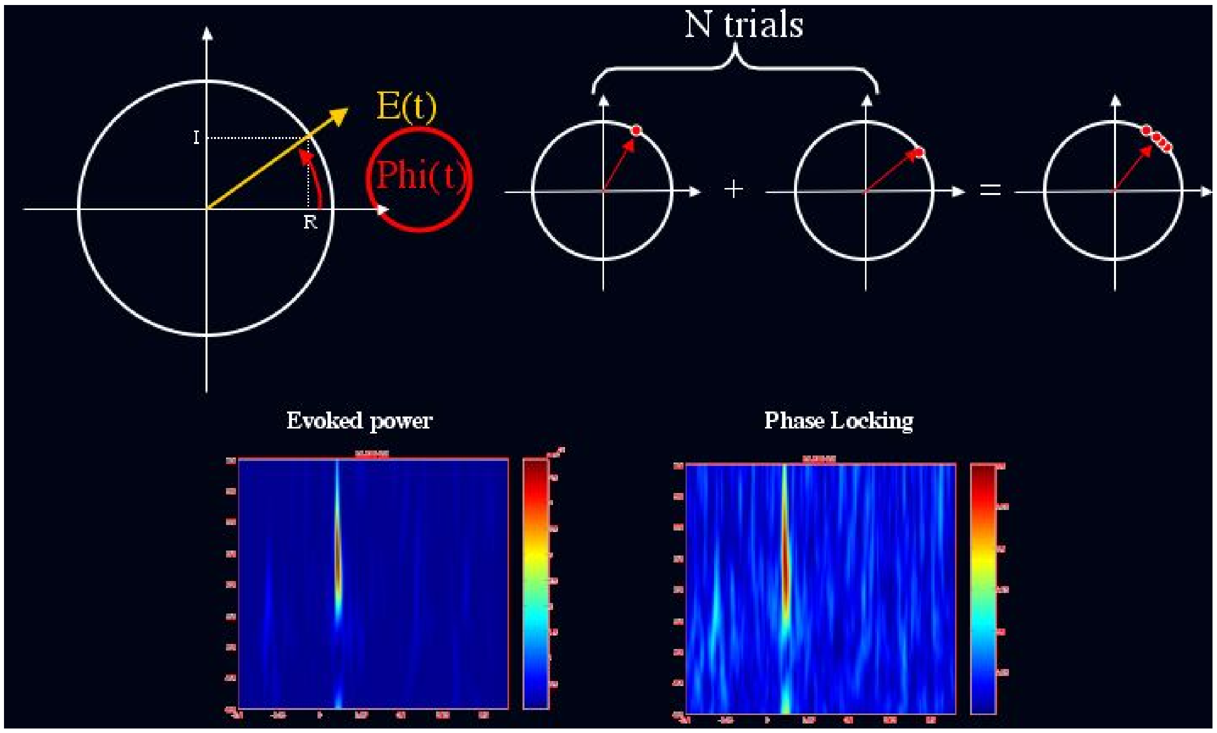


Figure 9: Phase locking computation

9.4.2 Input/Output Options

–**phase_lock**: Phase locking factor computation mode
–**first_frequency**: First frequency of the time-frequency map
–**last_frequency**: Last frequency of the time-frequency map
–**frequency_step**: Frequency step of the time-frequency map
–**blackman_win**: Size of blackman window in seconds
–**wavelet_m**: Wavelets parameter m
–**marker**: Marker name
–**begin_analysis:**: Beginning of the analysis window in seconds
–**end_analysis:**: End of the analysis window in seconds
–**output_file**: Results file path
–**input_file**: Input files path

9.4.3 command line examples

The following command line computes phase locking for all occurences of marker AF in run01.ds and run02.ds.

```
fast_tf --phase_lock --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti --begin_baseline -0.5 \
--end_baseline -0.2 --marker AF --begin_analysis -0.7 --end_analysis 1.5 \
--input_files test_tf/run01.ds --input_files test_tf/run02.ds
```

9.4.4 Results

Fast_TF generates a LENA file with a postfix "phase_lock" containing the results.

9.4.5 Statistical mode

Non averaged values of phase locking can be saved trial by trial by using –phase_lock_stat instead of –phase_lock. ***Fast_TF*** generates a LENA file with a postfix "phase_lock_stat" containing the results.

9.5 Evoked fields/Potentials

9.5.1 Purpose - Method

Compute evoked potential/field by averaging signals for all trials with respect to the markers and given analysis window. This option can not be used alone in this version. You need to specify at least one other computation mode (e.g –power).

9.5.2 Input/Output Options

–evoked: Evoked field/potential computation mode
–marker: Marker name
–begin_analysis:: Beginning of the analysis window in seconds
–end_analysis:: End of the analysis window in seconds
–output_file: Results file path
–input_files: Input files path

9.5.3 command line examples

The following command line computes evoked field and time frequency maps in term of power for all occurences of marker AF in run01.ds and run02.ds.

```
fast_tf --evoked --power --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti \
--marker AF --begin_analysis -0.7 --end_analysis 1.5 \
--input_files test_tf/run01.ds --input_files test_tf/run02.ds
```

9.5.4 Results

Fast_TF generates a LENA file with a postfix "evoked" containing the results.

9.6 Synchrony

9.6.1 Purpose - Method

Fast_TF can compute the amount of synchrony existing between two sensors (a sensors pairs) (See figure 10): At a given time step t, we define the square root of the power for sensor a and b:

$$f_k^p = \sqrt{fmap_k^{r\,2} + fmap_k^{i\,2}}\ with\ k = \{a, b\} \tag{19}$$

Similarly to phase locking computation we sum phase values for each sensor. Difference of phase between a and b are averaged accross trials:

$$R_k = \frac{f_k^r}{f_k^p}\ with\ k = \{a, b\} \tag{20}$$

$$I_k = \frac{f_k^i}{f_k^p}\ with\ k = \{a, b\} \tag{21}$$

$$s^r = \frac{\sum_{t=1}^{t=N} R_a^t * R_b^t + I_a^t * I_b^t}{N} \tag{22}$$

$$s^i = \frac{\sum_{t=1}^{t=N} R_a^t * I_b^t - R_b^t * I_a^t}{N} \tag{23}$$

Module (S^r) and phase (S^r) of synchrony are then computed as follows:

$$S^r = \sqrt{s^{r\,2} + s^{i\,2}} \tag{24}$$

$$S^i = atan(\frac{s^{i\,2}}{s^{r\,2}}) \tag{25}$$

9.6.2 Input Parameters

–sync_trial: Trial synchrony computation mode
–first_frequency: First frequency of the time-frequency map
–last_frequency: Last frequency of the time-frequency map
–frequency_step: Frequency step of the time-frequency map
–blackman_win: Size of blackman window in seconds
–wavelet_m: Wavelets parameter m
–marker: Marker name
–begin_analysis: Beginning of the analysis window in seconds
–end_analysis: End of the analysis window in seconds
–pairs: Path and name of the file containing sensors pairs description (see below)
–output_file: Results file path
–input_files: Input files path

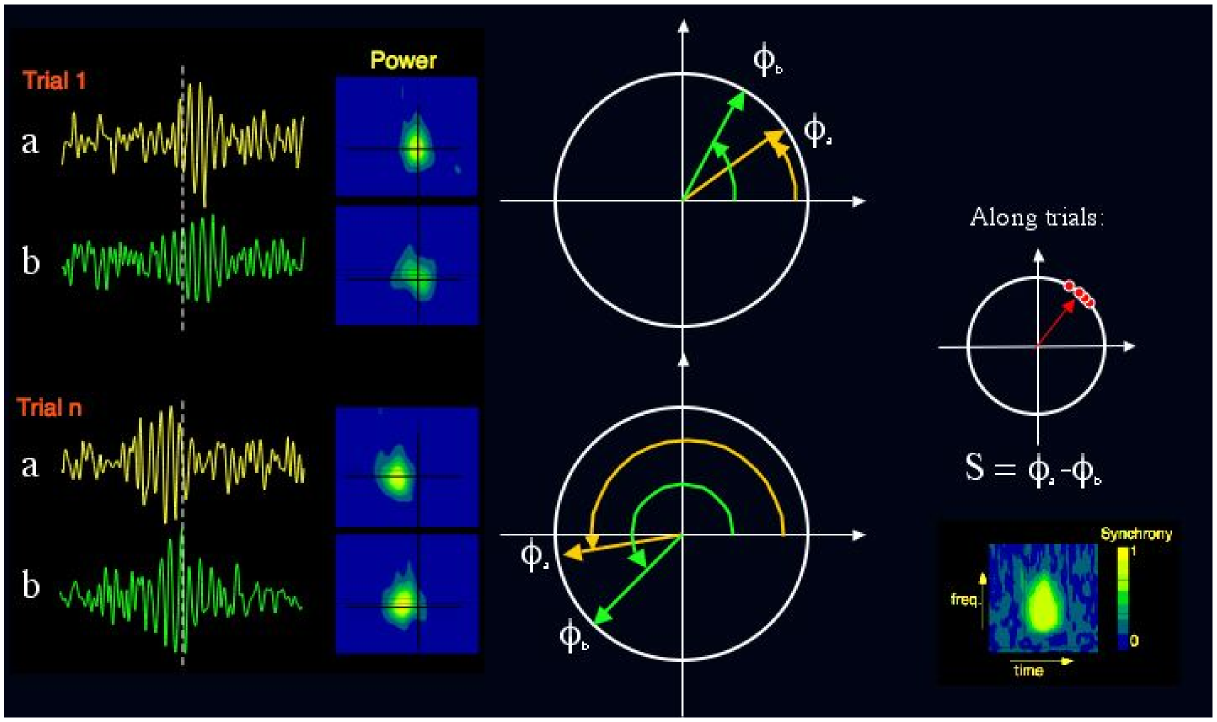


Figure 10: Synchronies computation

9.6.3 Pairs file format

Pairs of sensors are described by an ASCII file with the following format:
First line contains sensor labels separated by space. The file should contain n lines with a sensor label and n numbers (0 or 1) separated by space where n is the number of sensors if the file to be analyzed. If you want to compute synchrony between sensor i and j you should put a 1 at line i column j and a zero otherwise. Here is a very simple example:

$$\begin{array}{cccccc}
 & y_1 & y_2 & y_3 & y_4 & y_5 \\
y_1 & 0 & 1 & 0 & 0 & 1 \\
y_2 & 0 & 0 & 1 & 0 & 0 \\
y_3 & 0 & 0 & 0 & 0 & 1 \\
y_4 & 0 & 0 & 0 & 0 & 0 \\
y_5 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Synchrony will be computed between the following pairs of sensors: (y_1,y_2) (y_1,y_5) (y_2,y_3) (y_3,y_5).

9.6.4 command line examples

The following command line averages synchrony for all accurences of marker AF in run01.ds and run02.ds.

```
fast_tf --sync_trial --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti --pairs test_tf/pairs.txt\
--marker AF --begin_analysis -0.7 --end_analysis 1.5 \
--input_files test_tf/run01.ds --input_files test_tf/run02.ds
```

9.6.5 Results

Fast_TF generates two files LENA file, one with a postfix “sync” containing the module of synchrony for each pair and one with a postfix “sync_trial_phase” containing the phase of synchrony for each pair. For computation involving a lot of pairs of sensors and long duration (more than 100 pairs and more than 1000 samples) you should use smp_fast_tf.py to speed up computation and save disk space during computation (see section 11.3 for detail).

9.6.6 Statistical mode

Non averaged values $R_a^t * R_b^t + I_a^t * I_b^t$ and $R_a^t * I_b^t - R_b^t * I_a^t$ can be saved trial by trial by using –sync_trial_stat instead of –sync_trial. ***Fast_TF*** generates two LENA files with a postfix “_sync_real_phase” and “_sync_img_phase” containing the results.

9.7 Time Synchrony

9.7.1 Purpose - Method

Fast_TF can compute the amount of synchrony existing between two sensors (a sensors pair) averaged on a window of time defined as *wtime*: At a given time step t, we define the square root of the power for sensor a and b:

$$f_k^p = \sqrt{fmap_k^{r\,2} + fmap_k^{i\,2}} \; with \; k = \{a, b\} \tag{26}$$

Then we compute real (s^r) and imaginary part (s^i) of synchrony averaged on N trials:

$$R_k = \frac{f_k^r}{f_k^p} \; with \; k = \{a, b\} \tag{27}$$

$$I_k = \frac{f_k^i}{f_k^p} \; with \; k = \{a, b\} \tag{28}$$

$$s^r = \frac{\sum_{t=1}^{t=N} \left(\sum_{wtime} R_a^t * R_b^t + I_a^t * I_b^t \right)}{N} \tag{29}$$

$$s^i = \frac{\sum_{t=1}^{t=N} \left(\sum_{wtime} R_a^t * I_b^t - R_b^t * I_a^t \right)}{N} \tag{30}$$

Module (S^r) and phase (S^r) of synchrony are then computed as follows:

$$S^r = \sqrt{s^{r2} + s^{i2}} \tag{31}$$

$$S^i = atan(\frac{s^{i2}}{s^{r2}}) \tag{32}$$

9.7.2 Input Parameters

–sync_time: Time synchrony computation mode
–first_frequency: First frequency of the time-frequency map
–last_frequency: Last frequency of the time-frequency map
–frequency_step: Frequency step of the time-frequency map
–blackman_win: Size of blackman window in seconds
–wavelet_m: Wavelets parameter m
–marker: Marker name
–begin_analysis: Beginning of the analysis window in seconds
–end_analysis: End of the analysis window in seconds
–time_synchrony_begin: Beginning of the window used to compute time synchrony
–time_synchrony_end: End of the window used to compute time synchrony
–pairs: Path and name of the file containing sensors pairs description (see above)
–output_file: Results file path
–input_files: Input files path

9.7.3 command line examples

The following command line averages synchrony between .2 second before and 0.5 second after each occurence of marker AF in run01.ds and run02.ds.

```
fast_tf --sync_time --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti --pairs test_tf/pairs.txt\
--marker AF --begin_analysis -0.7 --end_analysis 1.5 --time_synchrony_begin -0.2\
--time_synchrony_end 0.5 --input_files test_tf/run01.ds --input_files test_tf/run02.ds
```

9.7.4 Results

Fast_TF generates two files LENA file, one with a postfix "sync_time" containing the module of synchrony for each pair and one with a postfix "sync_time_phase" containing the phase of synchrony for each pair.

9.7.5 Statistical mode

Averaged values $R_a^t{*}R_b^t{+}I_a^t{*}I_b^t$ and $R_a^t{*}I_b^t{-}R_b^t{*}I_a^t$ can be saved trial by trial by using –sync_time_stat instead of –sync_time. ***Fast_TF*** generates two LENA files with a postfix "_time_real_phase" and "_time_img_phase" containing the results.

9.8 Coherence

9.8.1 Purpose - Method

Fast_TF can compute the coherence between two sensors (a sensors pairs) : At a given time step t, we define the square root of the power for sensor a and b:

$$f_k^p = \sqrt{fmap_k^{r\,2} + fmap_k^{i\,2}}\ with\ k = \{a, b\} \tag{33}$$

For a given frequency f, we compute real and imaginary parts of the cross-spectrum between a and b sensors and the power spectrum for a and b :

$$R_k = \frac{f_k^r}{f_k^p} \text{ with } k = \{a, b\} \tag{34}$$

$$I_k = \frac{f_k^i}{f_k^p} \text{ with } k = \{a, b\} \tag{35}$$

$$s^r(t) = R_a(t) * R_b(t) + I_a(t) * I_b(t) \tag{36}$$

$$s^i(t) = R_a(t) * I_b(t) - R_b(t) * I_a(t) \tag{37}$$

$$p(t) = f^a(t) * f^b(t) \tag{38}$$

s^r, s^i and p are then averaged accross trials and the coherence is computed as follow:

$$S^r(t) = \frac{\sum_{trial=1}^{N} s^r(t)}{N} \tag{39}$$

$$S^i(t) = \frac{\sum_{trial=1}^{N} s^i(t)}{N} \tag{40}$$

$$S^p(t) = \frac{\sum_{trial=1}^{N} p(t)}{N} \tag{41}$$

$$C^f(t) = \frac{S^r(t)^2 + S^i(t)^2}{P(t)} \tag{42}$$

$$\tag{43}$$

9.8.2 Input Parameters

–coherence: coherence computation mode
–first_frequency: First frequency of the time-frequency map
–last_frequency: Last frequency of the time-frequency map
–frequency_step: Frequency step of the time-frequency map
–blackman_win: Size of blackman window in seconds
–wavelet_m: Wavelets parameter m
–marker: Marker name
–begin_analysis: Beginning of the analysis window in seconds
–end_analysis: End of the analysis window in seconds
–pairs: Path and name of the file containing sensors pairs description (see below)
–output_file: Results file path
–input_files: Input files path

9.8.3 Pairs file format

Pairs of sensors are described by an ASCII file with the following format:
First line contains sensor labels separated by space. The file should contain n lines with a sensor label and n numbers (0 or 1) separated by space where n is the number of sensors if the file to be analyzed. If you want to compute synchrony between sensor i and j you should put a 1 at line i column j and a zero otherwise. Here is a very simple example:

$$\begin{array}{cccccc} & y_1 & y_2 & y_3 & y_4 & y_5 \\ y_1 & 0 & 1 & 0 & 0 & 1 \\ y_2 & 0 & 0 & 1 & 0 & 0 \\ y_3 & 0 & 0 & 0 & 0 & 1 \\ y_4 & 0 & 0 & 0 & 0 & 0 \\ y_5 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Coherence will be computed between the following pairs of sensors: (y_1,y_2) (y_1,y_5) (y_2,y_3) (y_3,y_5).

9.8.4 command line examples

The following command line averages coherence for all accurences of marker AF in run01.ds and run02.ds.

```
fast_tf --coherence --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti --pairs test_tf/pairs.txt\
--marker AF --begin_analysis -0.7 --end_analysis 1.5 \
--input_files test_tf/run01.ds --input_files test_tf/run02.ds
```

9.8.5 Results

Fast_TF generates one LENA file with a postfix “coherence” containing the values of coherence for each pair and each frequency.

9.8.6 Statistical mode

Non averaged values of coherence can be saved trial by trial by using –coherence_stat instead of –coherence. ***Fast_TF*** generates three LENA files with a postfix “_coherence_real”,“_coherence_img” and “_coherence_pow” containing respectively s^r, s^i and p.

9.9 Coherence averaged on time

9.9.1 Purpose - Method

Fast_TF can compute the coherence between two sensors (a sensors pairs) averaged on a window of time : At a given time step t, we define the square root of the power for sensor a and b:

$$f_k^p = \sqrt{fmap_k^{r\,2} + fmap_k^{i\,2}}\ with\ k = \{a, b\} \tag{44}$$

For a given frequency f, we compute real and imaginary parts of cross-spectrum between a and b sensors and power spectrum for a and b :

$$R_k = \frac{f_k^r}{f_k^p} \ with \ k = \{a, b\} \tag{45}$$

$$I_k = \frac{f_k^i}{f_k^p} \ with \ k = \{a, b\} \tag{46}$$

$$s^r(t) = R_a(t) * R_b(t) + I_a(t) * I_b(t) \tag{47}$$

$$s^i(t) = R_a(t) * I_b(t) - R_b(t) * I_a(t) \tag{48}$$

$$p(t) = f^a(t) * f^b(t) \tag{49}$$

s^r, s^i and p are then averaged along the window of time $wtime$ containing Nw time samples. These values are averaged accross trials and the coherence is computed as follow:

$$S^r = \frac{\sum_{trial=1}^{N} \left(\frac{\sum_{wtime} s^r(t)}{Nw} \right)}{N} \tag{50}$$

$$S^i = \frac{\sum_{trial=1}^{N} \left(\frac{\sum_{wtime} s^i(t)}{Nw} \right)}{N} \tag{51}$$

$$S^p = \frac{\sum_{trial=1}^{N} \left(\frac{\sum_{wtime} s^p(t)}{Nw} \right)}{N} \tag{52}$$

$$C^f(t) = \frac{S^r(t)^2 + S^i(t)^2}{P(t)} \tag{53}$$

$$\tag{54}$$

9.9.2 Input Parameters

–coherence_time: time coherence computation mode
–first_frequency: First frequency of the time-frequency map
–time_synchrony_begin: Beginning of the window used to compute time coherence
–time_synchrony_end: End of the window used to compute time coherence
–last_frequency: Last frequency of the time-frequency map
–frequency_step: Frequency step of the time-frequency map
–blackman_win: Size of blackman window in seconds
–wavelet_m: Wavelets parameter m
–marker: Marker name
–begin_analysis: Beginning of the analysis window in seconds
–end_analysis: End of the analysis window in seconds
–pairs: Path and name of the file containing sensors pairs description (see below)
–output_file: Results file path
–input_files: Input files path

9.9.3 Pairs file format

Pairs of sensors are described by an ASCII file with the following format:
First line contains sensor labels separated by space. The file should contain n lines with a sensor label and n numbers (0 or 1) separated by space where n is the number of sensors if the file to be analyzed. If you want to compute synchrony between sensor i and j you should put a 1 at line i column j and a zero otherwise. Here is a very simple example:

$$\begin{array}{cccccc}
 & y_1 & y_2 & y_3 & y_4 & y_5 \\
y_1 & 0 & 1 & 0 & 0 & 1 \\
y_2 & 0 & 0 & 1 & 0 & 0 \\
y_3 & 0 & 0 & 0 & 0 & 1 \\
y_4 & 0 & 0 & 0 & 0 & 0 \\
y_5 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Coherence will be computed between the following pairs of sensors: (y_1,y_2) (y_1,y_5) (y_2,y_3) (y_3,y_5).

9.9.4 command line examples

The following command line averages coherence for all accurences of marker AF in run01.ds and run02.ds.

```
fast_tf --coherence_time --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti --pairs test_tf/pairs.txt\
--marker AF --begin_analysis -0.7 --end_analysis 1.5 \
--input_files test_tf/run01.ds --input_files test_tf/run02.ds
```

9.9.5 Results

Fast_TF generates one LENA file with a postfix "coherence_time" containing the values of coherence for each pair and each frequency.

9.9.6 Statistical mode

Non averaged values of coherence can be saved trial by trial by using –coherence_time_stat instead of –coherence. ***Fast_TF*** generates one LENA file with a postfix "_coherence_time_stat".

9.10 Mean power

9.10.1 Purpose - Method

Fast_TF can compute mean power over a given time-frequency window (t_{min} to t_{max} and f_{min} to f_{max}) accross trials. The resulting value is stored trial by trial. For a given trial mean power is computed as follows:

$$p = \frac{\sum_{i=t_{min}}^{t_{max}} \sum_{j=f_{min}}^{f_{max}} f_i^j}{N_t * N_f} \tag{55}$$

9.10.2 Input Parameters

–mean_power: mean power computation mode
–first_frequency: First frequency of the time-frequency map
–last_frequency: Last frequency of the time-frequency map
–frequency_step: Frequency step of the time-frequency map
–blackman_win: Size of blackman window in seconds
–wavelet_m: Wavelets parameter m
–marker: Marker name
–begin_analysis: Beginning of the analysis window in seconds
–end_analysis: End of the analysis window in seconds
–begin_tfwin_time: Beginning in time (s) of the time-frequency window used to average power
–end_tfwin_time: End in time (s) of the time-frequency window used to average power
–begin_tfwin_freq: Beginning in frequency (Hz) of the time-frequency window used to average power
–end_tfwin_freq: End in frequency (Hz) of the time-frequency window used to average power
–output_file: Results file path
–input_files: Input files path

9.10.3 command line examples

The following command line compute mean power between 0.2 second before and 0.5 second after marker AF and between 30 Hz and 70 Hz for run01.ds and run02.ds.

```
fast_tf --mean_power --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti \
--marker AF --begin_analysis -0.7 --end_analysis 1.5 --begin_tfwin_time -0.2\
--end_tfwin_time 0.5 --begin_tfwin_freq 30 --end_tfwin_freq 70 --input_files test_tf/run01.ds\
 --input_files test_tf/run02.ds
```

9.10.4 Results

Fast_TF generates one LENA file with a postfix "mean_power" containing mean power for each sensor for each trial.

9.11 Mean Z score

9.11.1 Purpose - Method

Fast_TF can compute mean z score over a given time-frequency window (t_{min} to t_{max} and f_{min} to f_{max}) accross trials. The resulting value is stored trial by trial. For a given trial mean z score is computed as follows (f_z is defined in equation 13):

$$p = \frac{\sum_{i=t_{min}}^{t_{max}} \sum_{j=f_{min}}^{f_{max}} f_z^{i,j}}{N_t * N_f} \tag{56}$$

9.11.2 Input Parameters

–mean_z_score: Mean z score computation mode
–first_frequency: First frequency of the time-frequency map
–last_frequency: Last frequency of the time-frequency map
–frequency_step: Frequency step of the time-frequency map
–blackman_win: Size of blackman window in seconds
–wavelet_m: Wavelets parameter m
–marker: Marker name
–begin_analysis: Beginning of the analysis window in seconds
–end_analysis: End of the analysis window in seconds
–begin_baseline:: Beginning of the baseline in seconds
–end_baseline:: End of the baseline in seconds
–begin_tfwin_time: Beginning in time (s) of the time-frequency window used to average power
–end_tfwin_time: End in time (s) of the time-frequency window used to average power
–begin_tfwin_freq: Beginning in frequency (Hz) of the time-frequency window used to average power
–end_tfwin_freq: End in frequency (Hz) of the time-frequency window used to average power
–output_file: Results file path
–input_files: Input files path

9.11.3 command line examples

The following command line compute mean z score between 0.2 second before and 0.5 second after marker AF and between 30 Hz and 70 Hz for run01.ds and run02.ds with a baseline estimated between -0.2 second before and 0.1 second after trigger AF.

```
fast_tf --mean_z_score --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti \
--marker AF --begin_analysis -0.7 --end_analysis 1.5 --begin_tfwin_time -0.2\
--end_tfwin_time 0.5 --begin_tfwin_freq 30 --end_tfwin_freq 70 --begin_baseline -0.2 \
--end_baseline 0.1 --input_files test_tf/run01.ds --input_files test_tf/run02.ds
```

9.11.4 Results

Fast_TF generates one LENA file with a postfix "mean_zscore" containing mean zscore for each sensor for each trial.

9.12 Phase

9.12.1 Purpose - Method

Fast_TF can compute phase in degree along time for each sensor. The resulting value is stored trial by trial. For a given trial the phase is computed as follows for a given sensor at a given time:

$$f_{ph} = atan\left(\frac{fmap^i}{fmap^r}\right) \tag{57}$$

9.12.2 Input Parameters

–phase: Phase computation mode
–first_frequency: First frequency of the time-frequency map
–last_frequency: Last frequency of the time-frequency map
–frequency_step: Frequency step of the time-frequency map
–blackman_win: Size of blackman window in seconds
–wavelet_m: Wavelets parameter m
–marker: Marker name
–begin_analysis: Beginning of the analysis window in seconds
–end_analysis: End of the analysis window in seconds
–output_file: Results file path
–input_files: Input files path

9.12.3 command line examples

The following command line compute phase between 0.7 second before and 1.5 second after marker AF and between 20 Hz and 70 Hz for run01.ds and run02.ds.

```
fast_tf --phase --first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti \
--marker AF --begin_analysis -0.7 --end_analysis 1.5 \
--input_files test_tf/run01.ds --input_files test_tf/run02.ds
```

9.12.4 Results

Fast_TF generates one LENA file with a postfix "phase" containing phase in degree along time for each sensor for each trial.

10 Wavelets parameters

The unique parameter controlling the shape of wavelets is m, its effects are discussed in the description of the algorithm.

–m_wavelet: m parameter controling wavelets temporal and frequential resolution

11 Advanced options

11.1 Advanced baseline handling

Fast_TF offers the possibility to dissociate markers for active conditions and markers for baseline in z score and log ratio computation. By using –marker_baseline and –begin_baseline_analysis, –end_baseline_analysis, –begin_baseline, –end_baseline, you can specify a marker for the baseline and an other marker for the "active" condition by using –marker. However, the following restrictions must be met to insure coherency between markers used for baseline condition and markers used for active condition:

- Either, it must exist 1 occurence of the marker of baseline for N occurences of marker for active condition,
- or N occurences of marker of baseline for N occurences of marker for active condition,
- If several markers of active condition are used, the same number of markers for baseline must be used,
- Nothing else (if number of occurences is different the program may crash or produce invalid results)

The options –begin_baseline_analysis, –end_baseline_analysis define the window of analysis for the baseline. The options –begin_baseline, –end_baseline define the temporal window where the baseline is computed (must be within the baseline analysis window). As for the "active" condition, Blackman window is applied within the baseline window of analysis - basically window of analysis for the active condition and baseline window of analysis should be more or less equal in term of time duration if you want to insure coherence between the frequency composition of your windows.

This option is very new and should be used with care – all misconfigurations are not tested !!!

–baseline_marker: Name of the baseline marker (between quotes if several markers are specified)
–begin_baseline_analysis: Start of the analysis window for the baseline
–end_baseline_analysis: End of the analysis window for the baseline
–begin_baseline: Start of the temporal window where the baseline is computed
–end_baseline: End of the temporal window where the baseline is computed

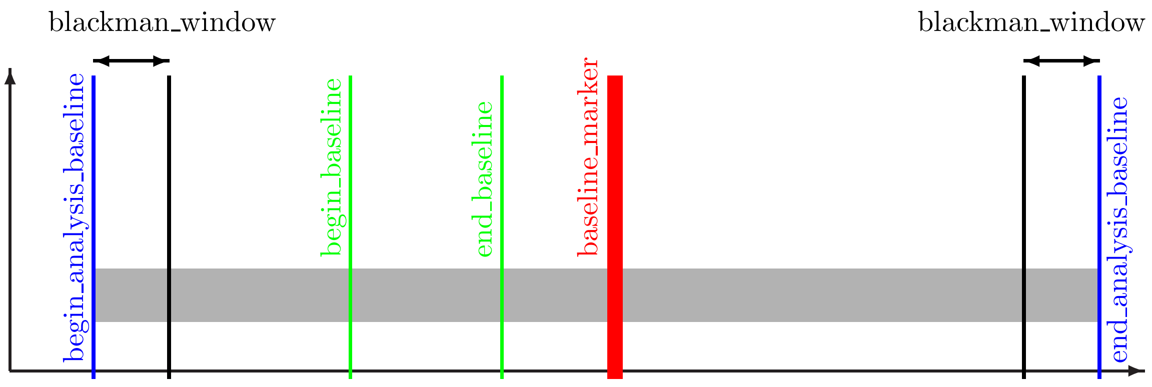


Figure 11: Graphical example of the temporal definition of the baseline with a specific marker

11.2 FFTW configuration

Fast_TF uses FFTW3 to compute FFT. This library offers nice options which can be used to improve your computation times using multi-processing capabilities. Obviously, improvements will bee seen on multi-cpu computers. FFTW3 can decompose its computation in several pieces working in parallel (named threads) and then further accelerate the computation (from our extensive testing number of cpus + 2 seems to be the optimal number of threads on Intel boxes).

–thread: Maximum number of threads used during FFT (set to 0 to disable)

11.3 Using ***Fast_TF*** on multi-cores computers

11.3.1 Introduction

It is now more and more common for computers to use cpu with multiple cores. Basically it means that your computer can do several things at the same moment. ***Fast_TF*** is capable of using these cores in two ways: by increasing the number of threads used in FFT (see section 11.2 for details and options) or by using a python script launching several instance of ***Fast_TF*** synchronously. Each instance is doing a part of the computation (i.e. the computation for one frequency) and the results are concatenated in one file at the end of the processing. Theoretically, you should be able to speed up your computation by the numbers of cpu cores on your machine. In practice expect to divide your computation by 75% of the the numbers of cpu cores on your machine. This script works only with the following options :

- –power
- –z_score
- –phase_lock
- –sync_trial
- –sync_time

Statistical modes and modes involving averaging over frequencies are not supported.

11.3.2 Using smp_fast_tf.py

It is very simple, you just have to replace fast_tf by the call to smp_fast_tf.py in your command line:

Original ***Fast_TF*** call:

```
fast_tf --evoked --power --phase_lock --sync_time --sync_trial \
--first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti --pairs test_tf/pairs.txt\
--marker AF --begin_analysis -0.7 --end_analysis 1.5 --time_synchrony_begin -0.2\
--time_synchrony_end 0.5 --input_files test_tf/run01.ds --input_files test_tf/run02.ds
```

becomes:

```
python path_to_your_fast_tf_installation/bin/smp_fast_tf.py --evoked \
--power --phase_lock --sync_time --sync_trial \
--first_frequency 20 --last_frequency 70 --frequency_step 5 \
--blackman_win 0.1 --wavelet_m 10 --output_file testmulti --pairs test_tf/pairs.txt\
--marker AF --begin_analysis -0.7 --end_analysis 1.5 --time_synchrony_begin -0.2\
--time_synchrony_end 0.5 --input_files test_tf/run01.ds --input_files test_tf/run02.ds
```

that's all. The output of smp_fast_tf.py will inform you of the progression of the computation.

12 Software License

12.1 The GNU General Public License

No Warranty