

*Fast\_TF*

## Fast MEG/EEG Time-Frequency Analysis

-

VERSION: 5.3.0

CRICM - CENIR - Pitié-Salpêtrière - France



February 1, 2013

# Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
1.1	What's <i>Fast_TF</i> ? . . . . .	6
1.1.1	Acknowledgments . . . . .	6
1.2	Copyright and Disclaimer . . . . .	6
<b>2</b>	<b>Installing <i>Fast_TF</i> program</b>	<b>6</b>
2.1	Pre-Requirements . . . . .	6
2.2	Downloading the Package . . . . .	7
2.3	Unpacking the Software . . . . .	7
2.4	Installing the software . . . . .	7
<b>3</b>	<b>General presentation</b>	<b>7</b>
<b>4</b>	<b>Quick start guide</b>	<b>7</b>
4.1	evoked . . . . .	7
4.2	Power . . . . .	8
4.3	Mean power . . . . .	8
4.4	Z score . . . . .	8
4.5	Mean Z score . . . . .	8
4.6	Phase locking factor . . . . .	8
4.7	Phase . . . . .	9
4.8	Synchrony . . . . .	9
4.8.1	Pairs file format . . . . .	9
4.9	Mean Synchrony along time . . . . .	9
4.10	Coherence . . . . .	10
4.11	Multi-modes computation . . . . .	10
4.12	Visualization with matlab . . . . .	10
4.13	Visualization with Muse . . . . .	10
<b>5</b>	<b>Methodological aspects</b>	<b>10</b>
5.1	Step by step time frequency analysis . . . . .	10
5.1.1	Short non mathematical description of time-frequency analysis . . . . .	10
5.1.2	Simple examples . . . . .	12
5.2	Algorithm design and theoretical background . . . . .	12
5.3	Frequential and temporal resolution of the time-frequency map . . . . .	14
<b>6</b>	<b>General options</b>	<b>16</b>
6.1	Verbose mode . . . . .	16
6.2	Rewrite mode . . . . .	16
6.3	Marker name . . . . .	16
6.4	MEG channels selection . . . . .	16

6.5	EEG channels selection . . . . .	16
6.6	MEG & EEG channels selection . . . . .	16
6.7	Channel selection . . . . .	16
6.8	Strict channel name matching . . . . .	17
<b>7</b>	<b>Window of analysis - Time parameters</b>	<b>17</b>
7.1	Window of analysis . . . . .	17
7.2	Blackman window . . . . .	17
7.3	Baseline window . . . . .	17
<b>8</b>	<b>Input / Output files specification</b>	<b>17</b>
8.1	Files inputs . . . . .	17
8.2	Using standard input . . . . .	18
8.3	Output file . . . . .	19
<b>9</b>	<b>Computation modes</b>	<b>20</b>
9.1	Power . . . . .	20
9.1.1	Purpose - Method . . . . .	20
9.1.2	Input/Output Options . . . . .	21
9.1.3	command line examples . . . . .	21
9.1.4	Results . . . . .	21
9.1.5	Statistical mode . . . . .	21
9.2	Z score . . . . .	21
9.2.1	Purpose - Method . . . . .	21
9.2.2	Input/Output Options . . . . .	22
9.2.3	command line examples . . . . .	22
9.2.4	Results . . . . .	22
9.2.5	Statistical mode . . . . .	23
9.3	Log ratio . . . . .	23
9.3.1	Purpose - Method . . . . .	23
9.3.2	Input/Output Options . . . . .	23
9.3.3	command line examples . . . . .	24
9.3.4	Results . . . . .	24
9.3.5	Statistical mode . . . . .	24
9.4	Phase Locking . . . . .	24
9.4.1	Purpose - Method . . . . .	24
9.4.2	Input/Output Options . . . . .	25
9.4.3	command line examples . . . . .	25
9.4.4	Results . . . . .	26
9.4.5	Statistical mode . . . . .	26
9.5	Evoked fields/Potentials . . . . .	26
9.5.1	Purpose - Method . . . . .	26
9.5.2	Input/Output Options . . . . .	26

9.5.3	command line examples . . . . .	26
9.5.4	Results . . . . .	26
9.6	Synchrony . . . . .	27
9.6.1	Purpose - Method . . . . .	27
9.6.2	Input Parameters . . . . .	27
9.6.3	Pairs file format . . . . .	28
9.6.4	command line examples . . . . .	28
9.6.5	Results . . . . .	29
9.6.6	Statistical mode . . . . .	29
9.7	Time Synchrony . . . . .	29
9.7.1	Purpose - Method . . . . .	29
9.7.2	Input Parameters . . . . .	30
9.7.3	command line examples . . . . .	30
9.7.4	Results . . . . .	30
9.7.5	Statistical mode . . . . .	30
9.8	Coherence . . . . .	30
9.8.1	Purpose - Method . . . . .	30
9.8.2	Input Parameters . . . . .	31
9.8.3	Pairs file format . . . . .	32
9.8.4	command line examples . . . . .	32
9.8.5	Results . . . . .	32
9.8.6	Statistical mode . . . . .	32
9.9	Coherence averaged on time . . . . .	32
9.9.1	Purpose - Method . . . . .	32
9.9.2	Input Parameters . . . . .	34
9.9.3	Pairs file format . . . . .	34
9.9.4	command line examples . . . . .	34
9.9.5	Results . . . . .	35
9.9.6	Statistical mode . . . . .	35
9.10	Mean power . . . . .	35
9.10.1	Purpose - Method . . . . .	35
9.10.2	Input Parameters . . . . .	35
9.10.3	command line examples . . . . .	35
9.10.4	Results . . . . .	36
9.11	Mean Z score . . . . .	36
9.11.1	Purpose - Method . . . . .	36
9.11.2	Input Parameters . . . . .	36
9.11.3	command line examples . . . . .	37
9.11.4	Results . . . . .	37
9.12	Phase . . . . .	37
9.12.1	Purpose - Method . . . . .	37
9.12.2	Input Parameters . . . . .	37

9.12.3	command line examples . . . . .	37
9.12.4	Results . . . . .	38
<b>10</b>	<b>Wavelets parameters</b>	<b>38</b>
<b>11</b>	<b>Advanced options</b>	<b>38</b>
11.1	Advanced baseline handling . . . . .	38
11.2	FFTW configuration . . . . .	39
11.3	Using <i>Fast_TF</i> on multi-cores computers . . . . .	39
11.3.1	Introduction . . . . .	39
11.3.2	Using smp_fast_tf.py . . . . .	40
<b>12</b>	<b>Software License</b>	<b>40</b>
12.1	The GNU General Public License . . . . .	40

## 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. Chau-  
mon, 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).

**DO NOT USE THIS PROGRAM FOR CLINICAL APPLICATION**

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

## 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 occurrences 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 occurrences 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 for each occurrence 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_tfwindows_time -0.2 \
--end_tfwindows_time 0.5 --begin_tfwindows_freq 30 --end_tfwindows_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 occurrences 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 for each occurrence 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_tfwindows_time -0.2 \
--end_tfwindows_time 0.5 --begin_tfwindows_freq 30 --end_tfwindows_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 across all occurrences 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 for each occurrence 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:

	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$
$y_1$	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

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 occurrences 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 occurrence 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 occurrences 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 compute 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 channel 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 developed by the LENA-UPR640 lab. It reads 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 Fourier 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.

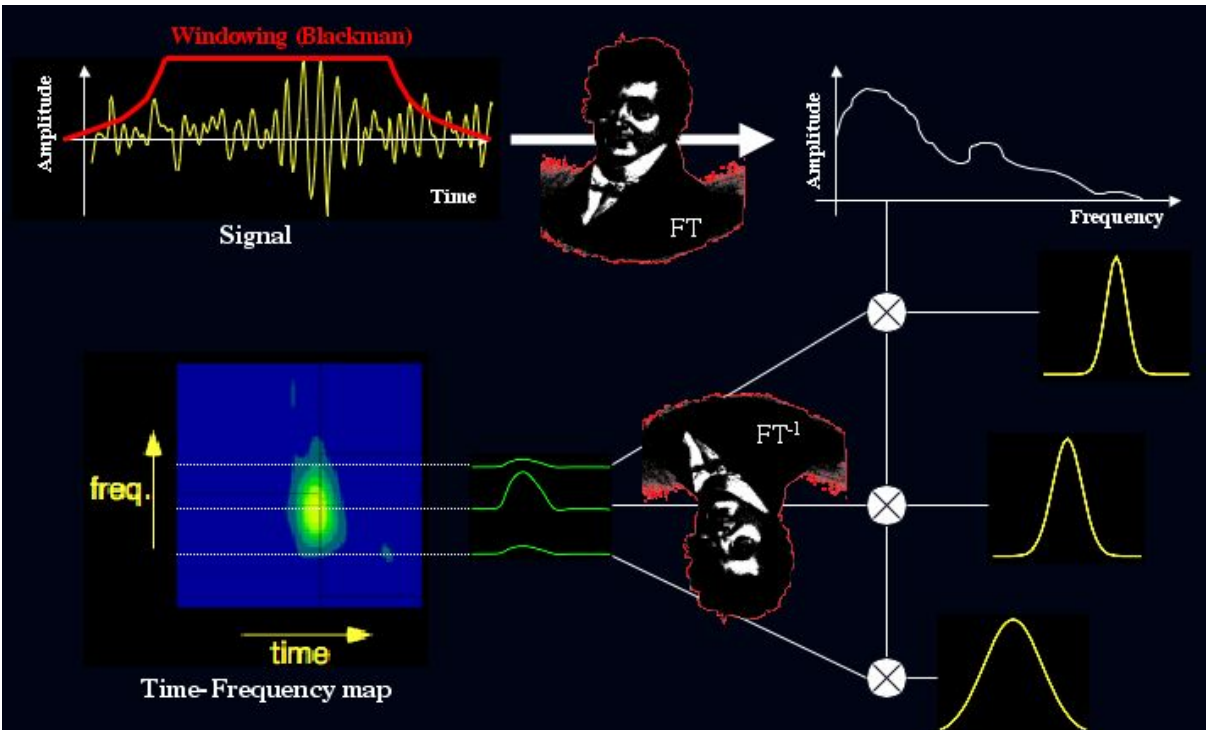


Figure 1: Framework of the method

*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:

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

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

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

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

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

$$F_c = \sum_{k=0}^{N_t-1} s_c e^{2\pi i k n / N} \quad (4)$$

where  $i = 1 \dots N_t$

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

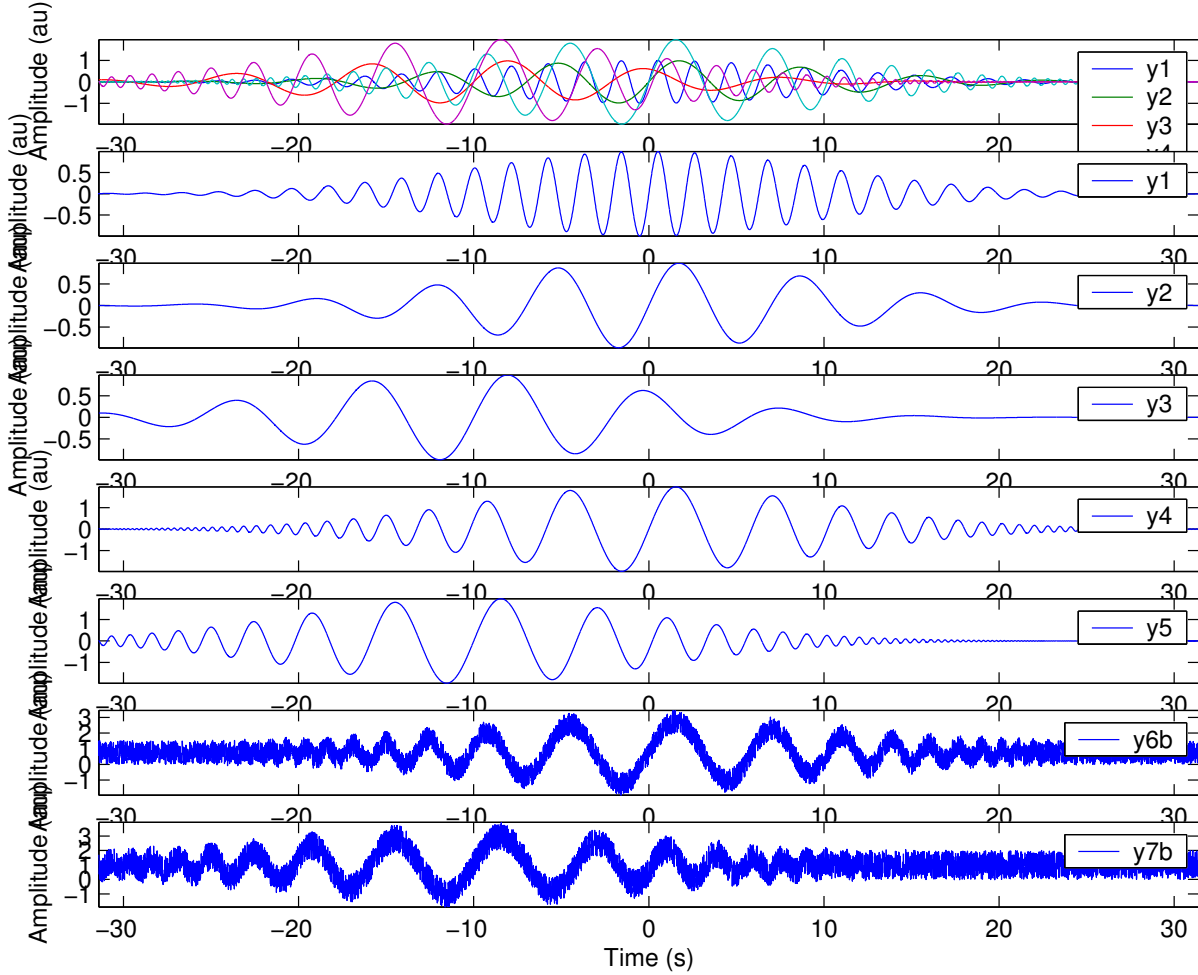


Figure 2: Tests signals

Each wavelets is then convoluted with  $F_c$ :

$$Fwav_c^i = F_c \otimes W_i \quad (5)$$

where  $i = 1 \dots N_f$ .  $Fwav_c^i$  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 Fourier Transform of  $Fwav_c^i$ :

$$fmap_c = \frac{1}{N_t} \sum_{k=0}^{N_t-1} Fwav_c e^{-2\pi i k n / N} \quad (6)$$

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

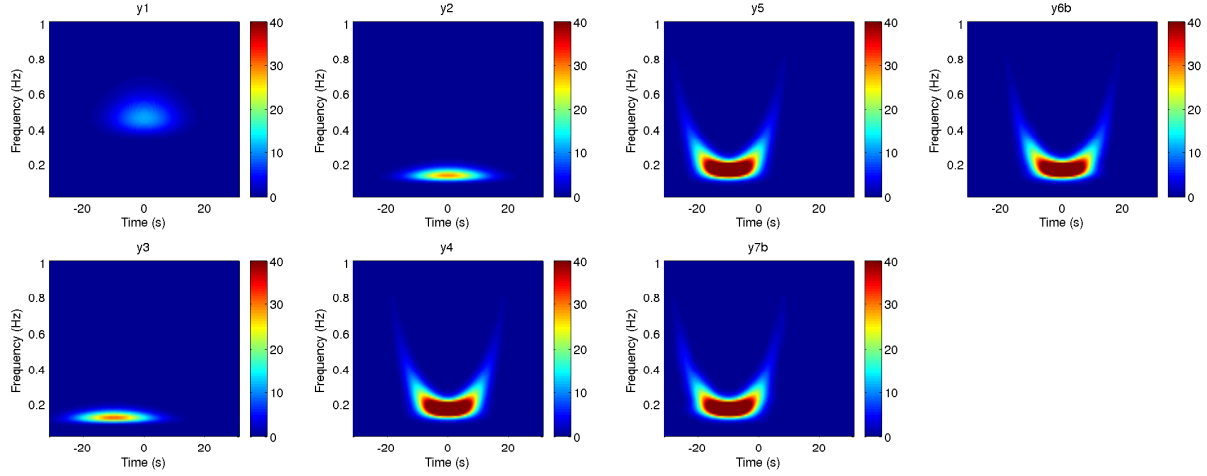


Figure 3: Time-frequency maps (Power)  $m = 5$

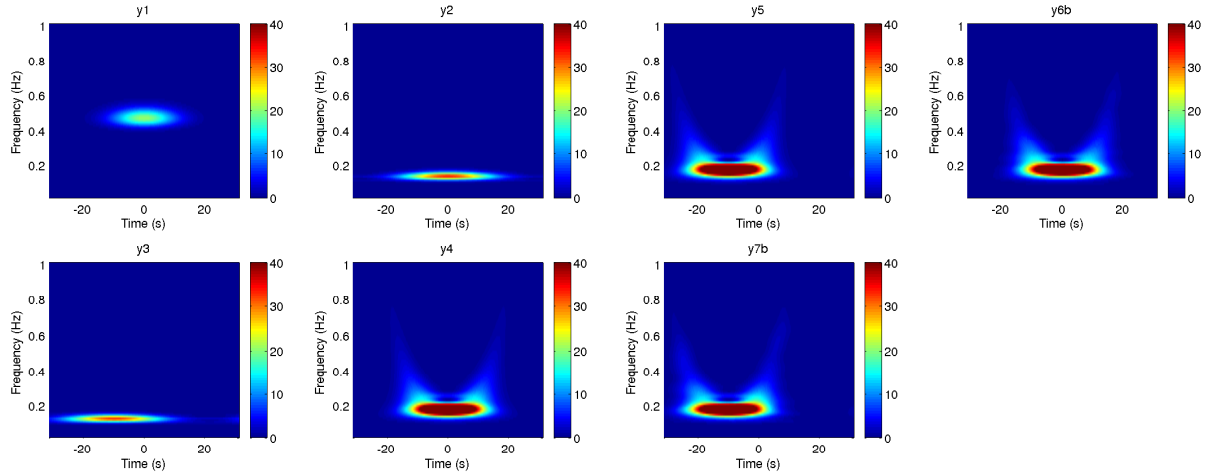


Figure 4: Time-frequency maps (Power)  $m = 10$

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

$$F_p = ((fmap_j^r)^2 + (fmap_j^i)^2) \quad (7)$$

### 5.3 Frequential and temporal resolution of the time-frequency map

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

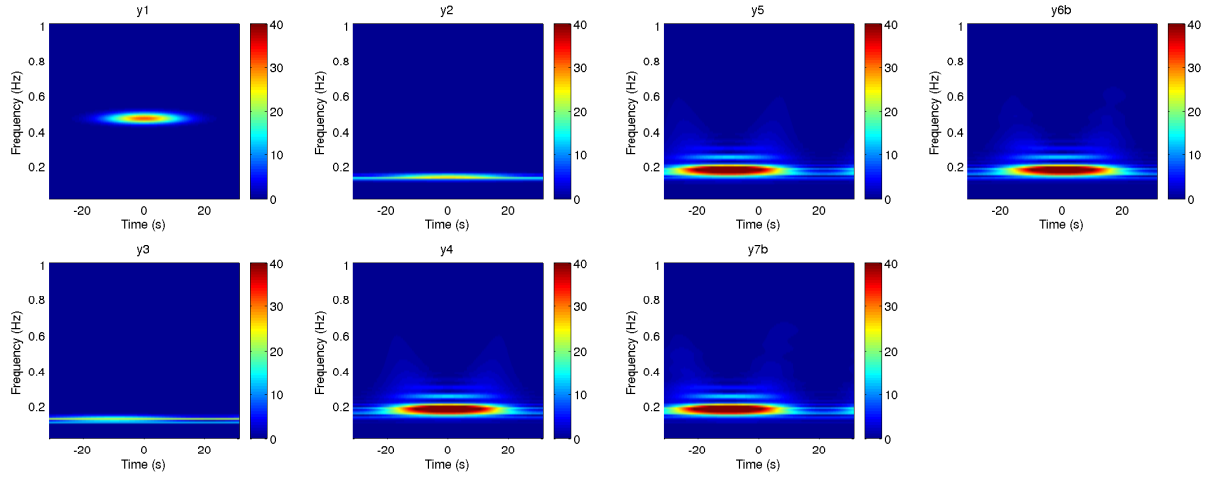


Figure 5: Time-frequency maps (Power)  $m = 20$

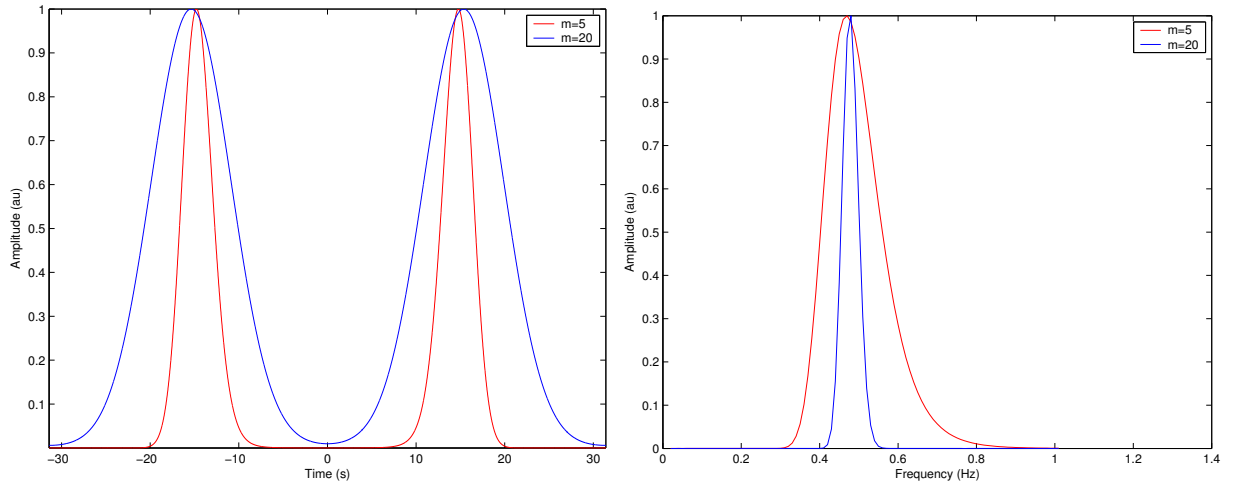


Figure 6: left: sensor  $y_1$  at 0s for  $m = 5$  and  $m = 20$  - Right: sensor  $y_4$  at 0.5Hz for  $m = 5$  and  $m = 20$

Frequential resolution at frequency  $f$  is given by:

$$\sigma_f = \frac{f}{m} \quad (8)$$

Temporal resolution at frequency  $f$  is given by:

$$\sigma_t = \frac{m}{2 * \pi * \sigma_f} \quad (9)$$

## 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 in the analyzed file. This option is useful if a set of channel names contains similar patterns and you want to analyze only one of them (bipolar channels for example).

**`-strict_channel_name`**: Strict 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* performs its analysis with respect to occurrences of one or several markers (triggers). All the timings 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` defines the time window used to bring the signals to zeros at the beginning AND the end of the window of analysis. It means that these two segments of signals will not contain any meaningful data of the time-frequency maps. This is necessary to avoid FFT artifacts.

### 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)

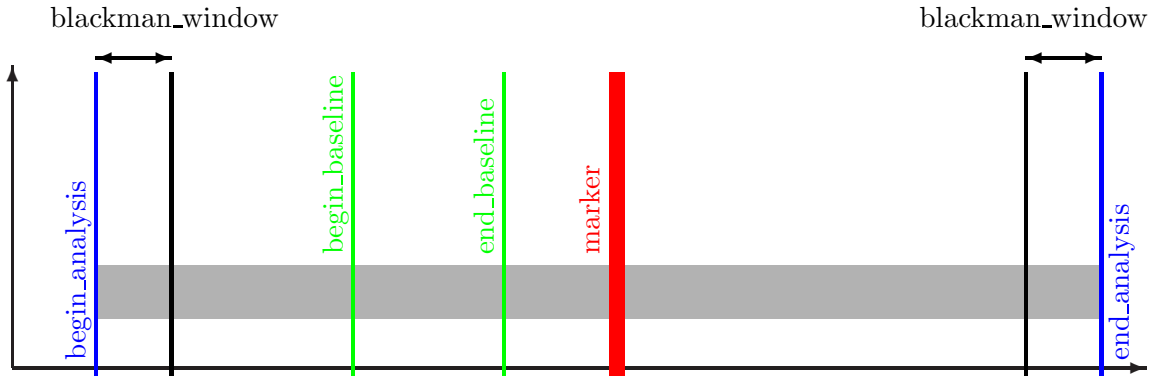


Figure 7: Graphical example of the timing definition in *Fast\_TF*

## 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 kind 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  $N$  occurrences of the markers (See figure 8 ):

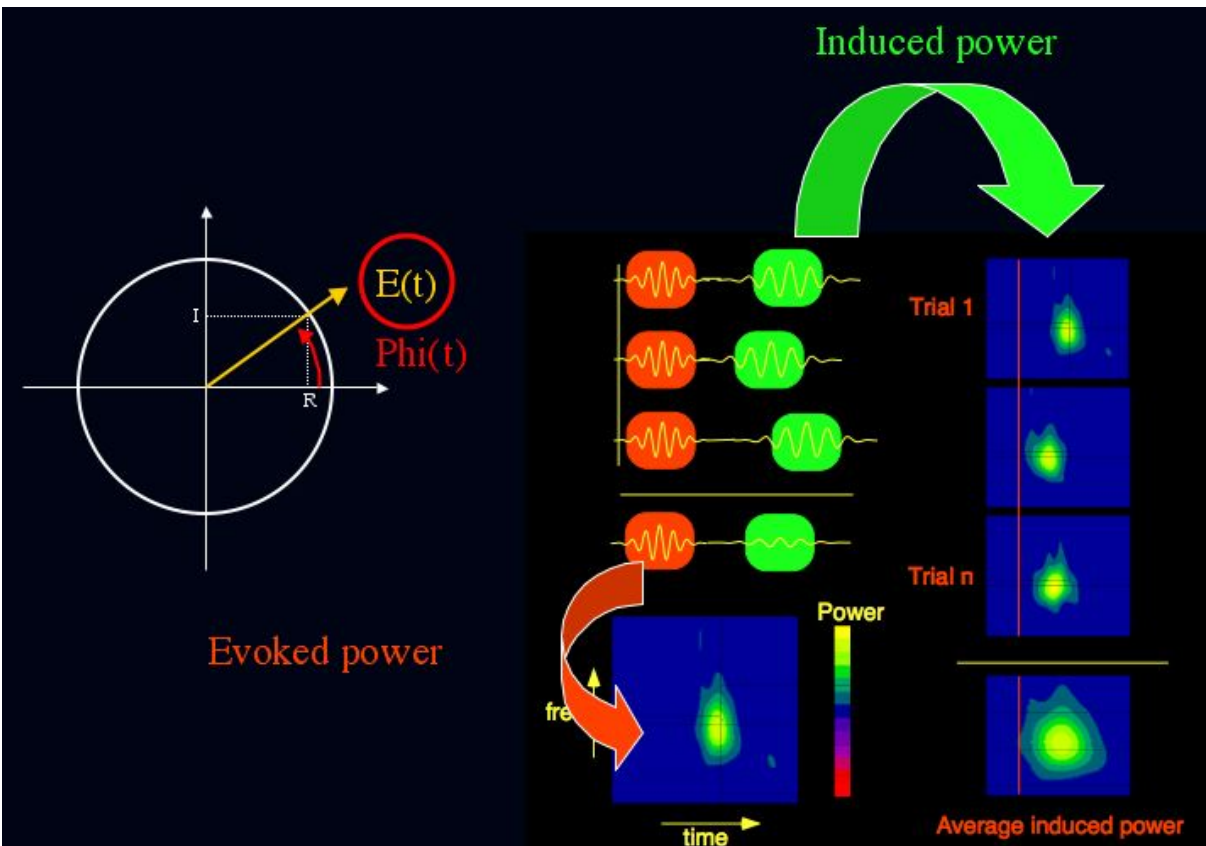


Figure 8: Power computation

$$p = \frac{\sum_{j=1}^N f_p^j}{N} \quad (10)$$

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 occurrences 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 occurrences 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:

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

Standard deviation is then defined as follow:

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

where  $N_{bas}$  is the number of samples in the baseline window.

Finally the z score map is computed as follow:

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

where  $N$  is the number of occurrences of the chosen 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 occurrences 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 occurrences 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:

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

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

$$f_l = \frac{\sum_{i=1}^N \log\left(\frac{f_p^i}{\bar{B}_p}\right)}{N} \quad (15)$$

where  $N$  is the number of occurrences of the chosen 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 occurrences 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 :

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

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

Then phase locking factor is computed as follows:

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

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 resetting at the beginning of an evoked potential for example.



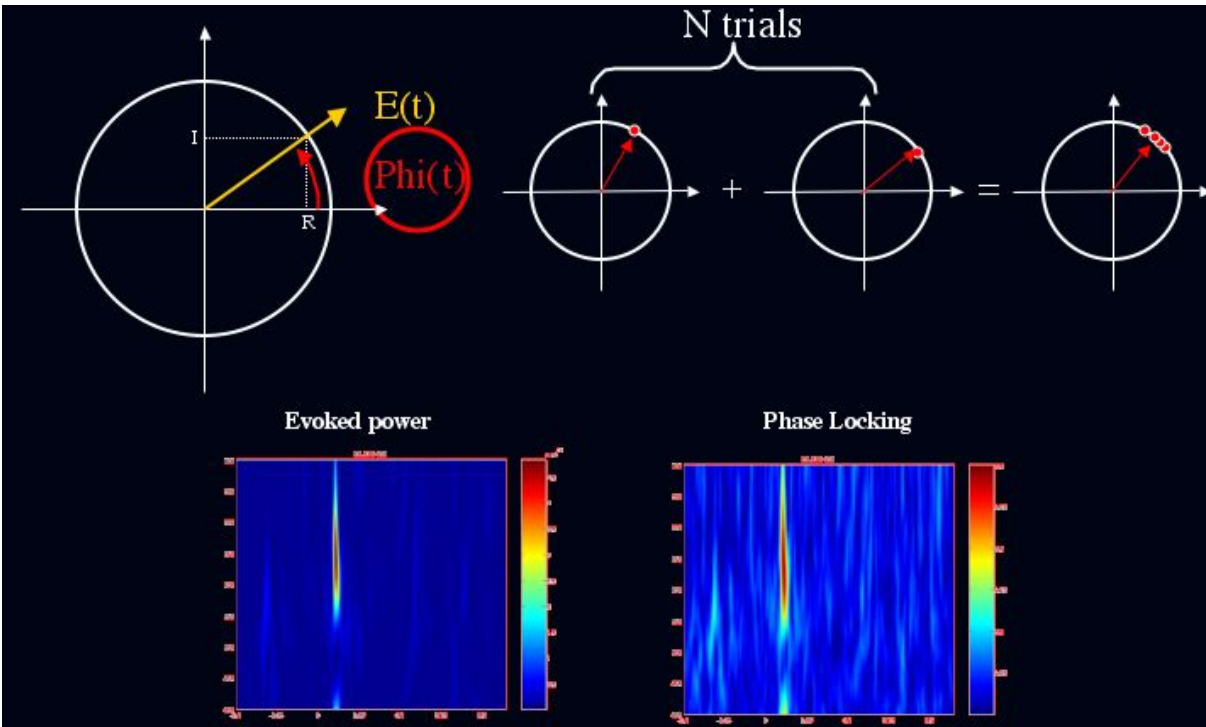


Figure 9: Phase locking computation

### 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 occurrences 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 occurrences 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  $b$ :

$$f_k^p = \sqrt{fmap_k^{r2} + fmap_k^{i2}} \text{ with } k = \{a, b\} \quad (19)$$

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

$$R_k = \frac{f_k^r}{f_k^p} \text{ with } k = \{a, b\} \quad (20)$$

$$I_k = \frac{f_k^i}{f_k^p} \text{ with } k = \{a, b\} \quad (21)$$

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

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

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

$$S^r = \sqrt{s^{r2} + s^{i2}} \quad (24)$$

$$S^i = atan\left(\frac{s^{i2}}{s^{r2}}\right) \quad (25)$$

### 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

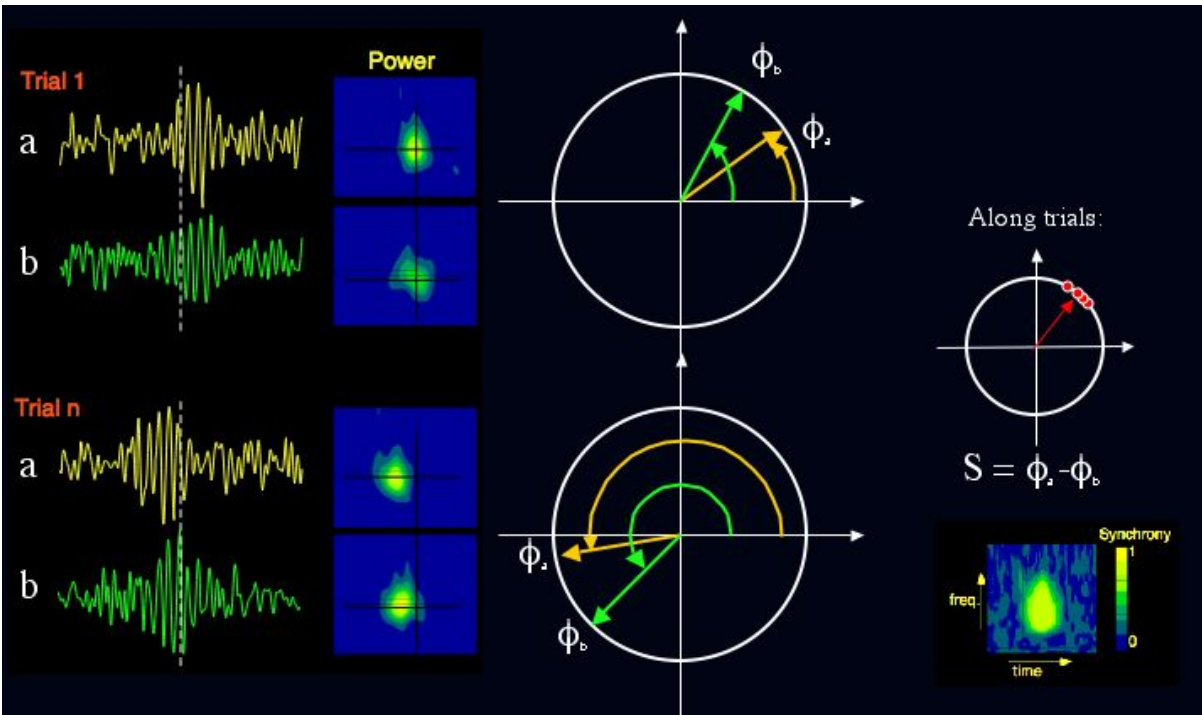


Figure 10: Synchronies computation

### 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:

	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$
$y_1$	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

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 occurrences 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  $b$ :

$$f_k^p = \sqrt{fmap_k^{r2} + fmap_k^{i2}} \text{ with } k = \{a, b\} \quad (26)$$

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

$$R_k = \frac{f_k^r}{f_k^p} \text{ with } k = \{a, b\} \quad (27)$$

$$I_k = \frac{f_k^i}{f_k^p} \text{ with } k = \{a, b\} \quad (28)$$

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

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

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

$$S^r = \sqrt{s^{r2} + s^{i2}} \quad (31)$$

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

## 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 occurrence 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:

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

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  $b$  :

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

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

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

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

$$p(t) = f^a(t) * f^b(t) \quad (38)$$

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

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

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

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

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

$$(43)$$

### 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:

	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$
$y_1$	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

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 occurrences 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  $b$ :

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



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  $b$  :

$$R_k = \frac{f_k^r}{f_k^p} \text{ with } k = \{a, b\} \quad (45)$$

$$I_k = \frac{f_k^i}{f_k^p} \text{ with } k = \{a, b\} \quad (46)$$

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

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

$$p(t) = f^a(t) * f^b(t) \quad (49)$$

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

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

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

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

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

$$(54)$$

## 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:

	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$
$y_1$	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

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 occurrences 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  $f_{max}$ ) across trials. The resulting value is stored trial by trial. For a given trial mean power is computed as follows:

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

### 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_tfwindows_time`: Beginning in time (s) of the time-frequency window used to average power
- `-end_tfwindows_time`: End in time (s) of the time-frequency window used to average power
- `-begin_tfwindows_freq`: Beginning in frequency (Hz) of the time-frequency window used to average power
- `-end_tfwindows_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_tfwindows_time -0.2\
--end_tfwindows_time 0.5 --begin_tfwindows_freq 30 --end_tfwindows_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  $f_{max}$ ) across 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):

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

#### 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\_tfwindows\_time**: Beginning in time (s) of the time-frequency window used to average power
- end\_tfwindows\_time**: End in time (s) of the time-frequency window used to average power
- begin\_tfwindows\_freq**: Beginning in frequency (Hz) of the time-frequency window used to average power
- end\_tfwindows\_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_tfwindows_time -0.2 \
--end_tfwindows_time 0.5 --begin_tfwindows_freq 30 --end_tfwindows_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:

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

### 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.

<b>-m_wavelet</b> : $m$ parameter controlling 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 another 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 occurrence of the marker of baseline for  $N$  occurrences of marker for active condition,
- or  $N$  occurrences of marker of baseline for  $N$  occurrences 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 occurrences 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

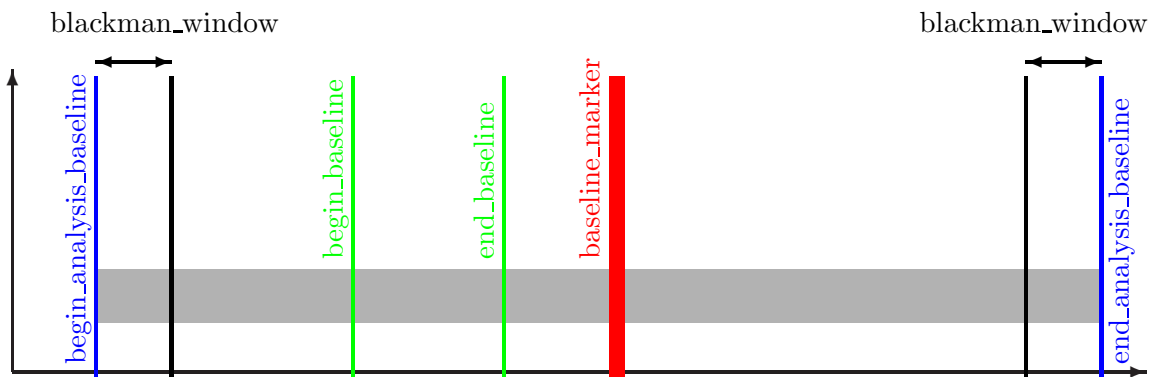


Figure 11: Graphical example of the temporal definition of the baseline with a specific marker

## 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 be 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

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.

59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.



## Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation’s software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author’s protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors’ reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone’s free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

## TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The “Program”, below, refers to any such program or work, and a “work based on the Program” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term “modification”.) Each licensee is addressed as “you”.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
  - (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
  - (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
  - (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

- 4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
- 5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.
- 6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

## NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

## END OF TERMS AND CONDITIONS

### Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

one line to give the program’s name and a brief idea of what it does.

Copyright (C) yyyy name of author

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

**DO NOT USE THIS SOFTWARE FOR CLINICAL APPLICATION**

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) yyyy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it under certain conditions; type
'show c' for details.
```

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
'Gnomovision' (which makes passes at compilers) written by James Hacker.
```

```
signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.