EEGLAB Tutorial

Arnaud Delorme and Scott Makeig, August 18, 2003

Table of Contents

I. Data Analysis Tutorial **1**
- I.1. Loading data and visualizing data information 1
  - I.1.1. Getting started 1
  - I.1.2. Opening an existing dataset 2
  - I.1.3. Editing event fields 3
  - I.1.4 − Editing event values 5
  - I.1.5. About this dataset 5
  - I.1.6. Scrolling through the data 6
- I.2. Using channel locations 10
- I.3. Plotting channel spectra and maps 14
- I.4. Preprocessing tools 15
  - I.4.1. Changing the data sampling rate 15
  - I.4.2. Filtering the data 16
  - I.4.3. Re−referencing the data 16
- I.5. Extracting data epochs 17
  - I.5.1. Extracting epochs 17
  - I.5.2. Removing baseline values 18
- I.6. Data averaging 19
  - I.6.1. Plotting the ERP data on a single axis with scalp maps 20
  - I.6.2. Plotting ERP traces in a topographic array 21
  - I.6.3. Plotting ERPs in rectangular array 22
  - I.6.4. Plotting an ERP as a series of maps 23
- I.7. Selecting data epochs and plotting data averages 26
  - I.7.1. Selecting events and epochs for two conditions 26
  - I.7.2. Computing Grand Mean ERPs 28
  - I.7.3. Finding ERP peak latencies 30
  - I.7.4. Comparing ERPs in two conditions 31
- I.8. Plotting ERP images 33
  - I.8.1. Selecting a channel to plot 33
  - I.8.2. Plotting ERP images using pop_erpimage() 35
  - I.8.3. Sorting trials in ERP images 37
  - I.8.4. Plotting ERP images with spectral options 41
  - I.8.5. Plotting spectral amplitude in single trials and additional options 45
- I.9. Performing Independent Component Analysis of EEG data 47
  - I.9.1. Running ICA decompositions 47
  - I.9.2. Plotting 2−D Component Scalp Maps 49
  - I.9.3. Plotting component headplots 51
  - I.9.4. Studying and removing ICA components 52
  - I.9.5. Subtracting ICA components from data 57
  - I.9.6. Retaining multiple ICA weights in a dataset 58
  - I.9.7. Scrolling through component activations 58
- I.10. Working with ICA components 59
  - I.10.1. Rejecting data epochs by inspection using ICA 59
  - I.10.2. Plotting component spectra and maps 61
  - I.10.3. Plotting component ERPs 63
  - I.10.4. Plotting component ERP contributions 64
  - I.10.5. Component ERP−image plotting 67
- I.11. Time/frequency decomposition 69

Table of Contents

I. Data Analysis Tutorial
I.11.1. Decomposing channel data....69
I.11.2. Computing component time/frequency transforms....71
I.11.3. Computing component cross−coherences....72
I.11.4. Plotting ERSP time course and topography....73

II. Importing/exporting data and event or epoch information into EEGLAB....75
II.1. Importing continuous data....75
II.1.1. Importing a Matlab array....75
II.1.2. Importing Biosemi .BDF files....76
II.1.3. Importing European data format .EDF files....77
II.1.4. Importing EGI .RAW continuous files....77
II.1.5. Importing Neuroscan .CNT continuous files....77
II.1.6. Importing Neuroscan .DAT information files....79
II.1.7. Importing Snapmaster .SMA files....80
II.1.8. Importing ERPSS .RAW or .RDF data files....80
II.1.9. Importing data in other data formats....80
II.2. Importing event information for a continuous EEG dataset....80
II.2.1. Importing events from a data channel....80
II.2.2. Importing events from a Matlab array or text file....81
II.2.3. Importing events from a Presentation file....82
II.3. Importing sets of single−trial EEG epochs into EEGLAB....83
II.3.1. Importing .RAW EGI data epoch files....83
II.3.2. Importing Neuroscan .EEG data epoch files....83
II.3.3. Importing epoch info Matlab array or text file into EEGLAB....84
II.4. Importing sets of data averages into EEGLAB....87
II.4.1. Importing data into Matlab....87
II.4.2. Concatenating data averages....87
II.4.3. Importing concatenated data averages in EEGLAB....88
II.5. Exporting data and ICA matrices....89
II.5.1. Exporting data to an ASCII text file....89
II.5.2. Exporting ICA weights and inverse weight matrices....90

III. Rejecting artifacts in continuous and epoched data....91
III.1. Rejecting artifacts in continuous data....91
III.2. Rejecting artifacts in epoched data....93
III.2.1. Rejecting epochs by visual inspection....94
III.2.2. Rejecting extreme values....94
III.2.3. Rejecting abnormal trends....96
III.2.4. Rejecting improbable data....97
III.2.5. Rejecting abnormally distributed data....100
III.2.6. Rejecting abnormal spectra....101
III.2.7. Inspecting current versus previously proposed rejections....103
III.2.8. Inspecting results of all rejection measures....104
III.2.9. Notes and strategy....105
III.3. Rejection based on independent data components....106

Table of Contents

IV. Writing EEGLAB Matlab Scripts......107
IV.1. Why write EEGLAB Matlab scripts?......107
IV.2. Using EEGLAB command history to perform basic EEGLAB script writing......107
IV.3. How do EEGLAB pop_functions work?......110
VI.4. What is the structure of an EEGLAB dataset?......111
IV.4.1. EEG......111
IV.4.2. EEG.chanlocs......112
IV.4.3. EEG.event......113
IV.4.4. EEG.epoch......114
IV.4.5. Modifying the EEG structure......116
IV.5. Sample scripts involving multiple datasets......117
IV.5.1. Performing time−frequency decompositions on multiple datasets......117
IV.5.2. Creating a scalp topography animation......118
IV.5.3. Saving the ICA weight and sphere matrix of the current dataset......119

A1. Options to Maximize Memory and Disk Space......120
A1.1 Maximize memory menu......120
A1.2. The icadefs.m file......121

A2. Equivalent dipole source localization of independent components using BESA......122
A2.1. Component localization by equivalent dipoles: A simple example......122
A2.2. How to localize components using BESA......123
A2.3. Exporting component information......124
A2.4. Importing BESA component locations......125
A2.5. Visualizing dipole locations......126
A2.6. Miscellany......127

I. Data Analysis Tutorial

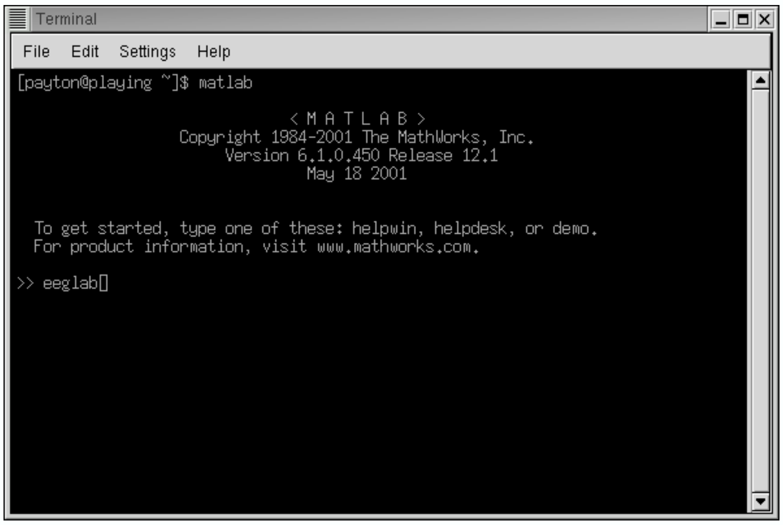
Tutorial outline

This tutorial will demonstrate how to use EEGLAB to interactively preprocess, analyze and visualize the dynamics of event−related EEG or MEG data using the tutorial EEG dataset "**eeglab_data.set**" which you may download (compressed by gzip) here (4.1Mb).

I.1. Loading data and visualizing data information

I.1.1. Getting started

To begin with, we change our working directory (folder) to one containing the EEGLAB dataset we want to analyze. Then start Matlab. To increase Matlab stability, we advise using Matlab without its java graphic interface, "**% matlab −nojvm**". Then run EEGLAB (as below).

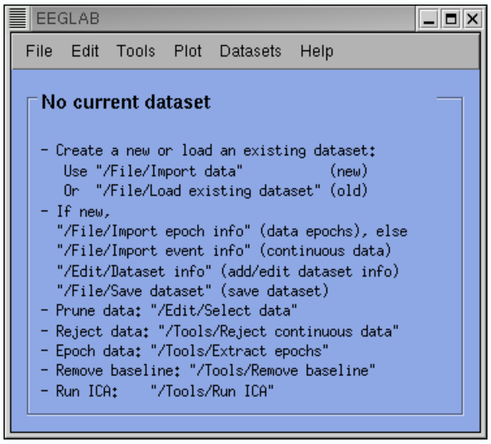


```
[payton@playing ~]$ matlab

                              < M A T L A B >
                  Copyright 1984-2001 The MathWorks, Inc.
                      Version 6.1.0.450 Release 12.1
                               May 18 2001


  To get started, type one of these: helpwin, helpdesk, or demo.
  For product information, visit www.mathworks.com.

>> eeglab
```

The blue main EEGLAB window below should pop−up, with its six menu headings:

File Edit Tools Plot Datasets Help

arranged in typical (left−to−right) order of use:



No current dataset

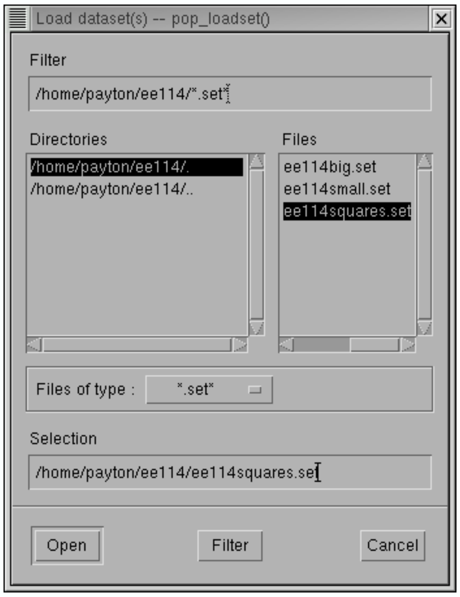
```
- Create a new or load an existing dataset:
   Use "/File/Import data"           (new)
   Or  "/File/Load existing dataset" (old)
- If new,
  "/File/Import epoch info" (data epochs), else
  "/File/Import event info" (continuous data)
  "/Edit/Dataset info" (add/edit dataset info)
  "/File/Save dataset" (save dataset)
- Prune data: "/Edit/Select data"
- Reject data: "/Tools/Reject continuous data"
- Epoch data: "/Tools/Extract epochs"
- Remove baseline: "/Tools/Remove baseline"
- Run ICA:     "/Tools/Run ICA"
```

I.1.2. Opening an existing dataset

First, we load the sample EEGLAB dataset. To learn how to create EEGLAB datasets from your own data, see the tutorial chapter on **Creating datasets**. Select menu item "**File**" and press sub−menu item "**Load existing dataset**". In the rest of the tutorial, we will use the convention:

Menu_item > Submenu_item

to refer to selecting a menu choice (e.g., here select **File > Load existing dataset**). Under Unix, the following window will pop up (the aspect of the window may be different under Windows):



Filter: /home/payton/ee114/*.set

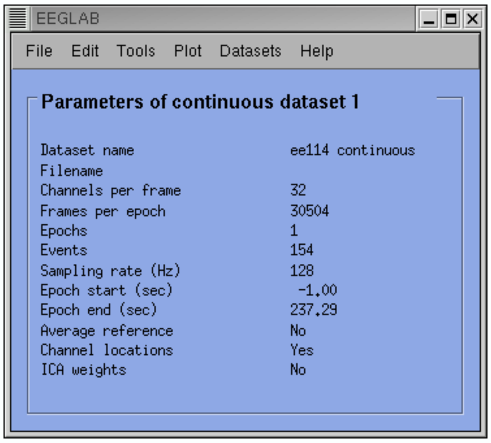
Directories:
/home/payton/ee114/.
/home/payton/ee114/..

Files:
ee114big.set
ee114small.set
ee114squares.set

Files of type : *.set*

Selection: /home/payton/ee114/ee114squares.set

Select the tutorial file "**eeglab_data.set**" which is distributed with the toolbox and press " **Open**". When the dataset is loaded by EEGLAB, the main EEGLAB window shows relevant information about it −− the number of channels, sampling rate, etc...:



Parameters of continuous dataset 1

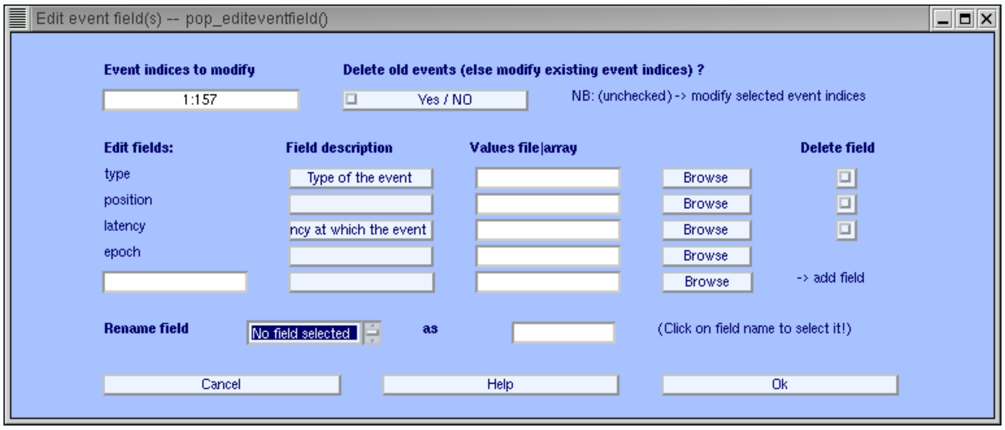
```
Dataset name              ee114 continuous
Filename
Channels per frame        32
Frames per epoch          30504
Epochs                    1
Events                    154
Sampling rate (Hz)        128
Epoch start (sec)          -1.00
Epoch end (sec)           237.29
Average reference         No
Channel locations         Yes
ICA weights               No
```

I.1.3. Editing event fields

This continuous EEG dataset file contains raw 32−channel data plus records of 154 events that occured during the experiment. The main EEGLAB window (above) indicates that event information has been loaded from the dataset file.

Event information for continuous EEGLAB data is stored in a set of fields that give relevant information for each event. Menu item **Edit > Event fields** pops up a window that allows us to edit the event fields. When using events in an EEGLAB dataset, there are two required event fields: "**type**" and "**latency**", plus any number of additional user−defined information fields.

Here, let us add text descriptions explaining the meaning and values in each field of our sample dataset. Note that in general, EEGLAB events require some level of understanding to use correctly. See also the later **EEGLAB epoch and event selection** portion of the tutorial for more details. Here, select menu item **Edit > Event Fields**



Event indices to modify: 1:157
Delete old events (else modify existing event indices) ? Yes / NO
NB: (unchecked) -> modify selected event indices

Edit fields:	Field description	Values file|array	Delete field
type	Type of the event		☐
position			☐
latency	ncy at which the event		☐
epoch			

Rename field: No field selected as
(Click on field name to select it!)

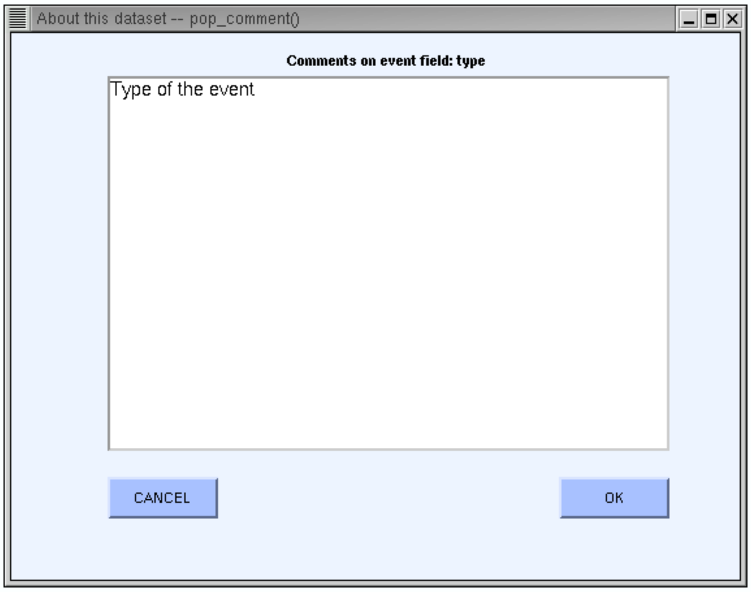
Events in this dataset have three event fields named "**type**", "**position**" and "**latency**". These three pieces of information are attached to each event.

- As we will see later, in this experiment there were two types of recorded events, stimulus presentations and subject button presses. The field "**type**" **always** indicates the type of the event.
- For this experiment, the field "**position**" indicates the screen position (1 or 2) at which the target stimulus was presented.
- The field "**latency**" **always** indicates the latency of the event in seconds from the beginning of the continuous data.

We will give more details about this experiment below. It is important to understand here that the names of the fields were defined by the user creating the dataset, and that it is possible to create, save, and load as many event fields as desired.

Note also that "**type**" and "**latency**" (lowercase) are two keywords recognized by EEGLAB and that these fields **must** be defined by the user unless importing epoch event information (Note: If only field "**latency**" is defined, then EEGLAB will create field "**type**" with a constant default value of 1 for each event). Unless these two fields are defined, EEGLAB will not be able to handle events appropriately to extract epochs, plot reaction times, etc. The **Creating datasets** tutorial explains how to import event information and define fields.

Under the column "**Field Description**" in the "**Edit event fields**" window (above), click on the (empty) button to the right of the field name "**position**". Type in an appropriate description for this field in the text pop up window that appears. Press **Save** when done.


About this dataset -- pop_comment()

Comments on event field: type

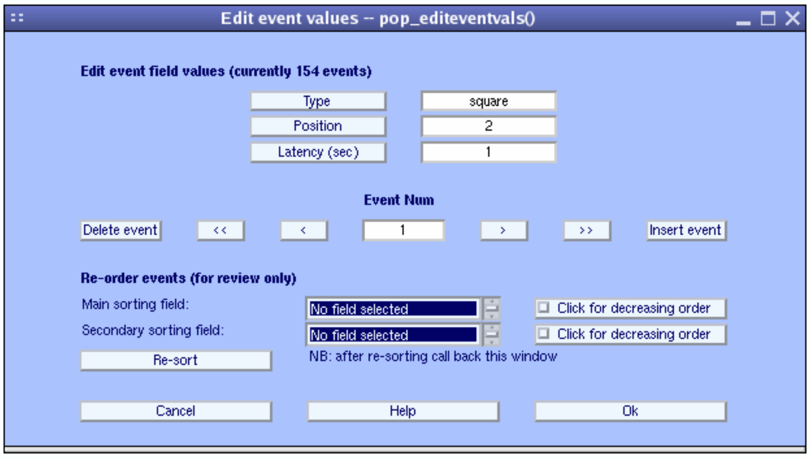
Type of the event

Press **SAVE** in the event field window to save the new comment in the dataset structure and to return to the main window. Note: The new comment has not yet been saved with the dataset to disk.

To do so, use menu item **File > Save current dataset.**

I.1.4 − Editing event values

The fields "**type** ", "**position**" and "**latency**" have different values for each of the 154 events in the dataset. Select menu **Edit > Event Values** to read and/or edit these values:



Edit event values -- pop_editeventvals()

Edit event field values (currently 154 events)

Type	square
Position	2
Latency (sec)	1

Event Num: 1

Re-order events (for review only)

Main sorting field:	No field selected	☐ Click for decreasing order
Secondary sorting field:	No field selected	☐ Click for decreasing order

NB: after re-sorting call back this window

Scroll through the events by pressing the "**>**", "**>>**", " **<** " and "**<<**" keys above.

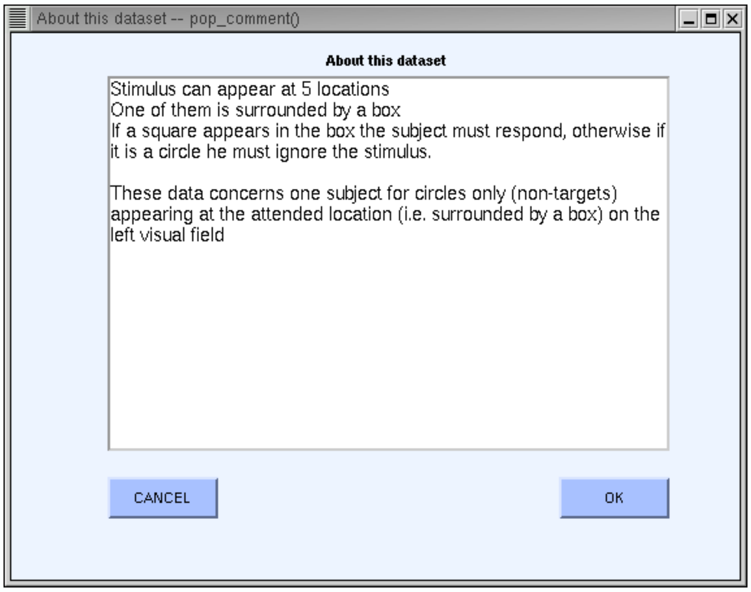
We will now briefly describe the experiment that produced the sample dataset to motivate the analysis steps we demonstrate in the rest of the tutorial.

Sample experiment description

In this experiment, there were two types of events "**square**" and "**rt**", " **square**" events corresponding to the appearance of a green colored square in the display and rt to the reaction time of the subject. The square could be presented at five locations on the screen distributed along the horizontal axis. Here we only considered presentation on the left, i.e. position 1 and 2 as indicated by the "**position**" field at about 3 degree and 1.5 degree of visual angle respectively. In this experiment, the subject had to attend the selected location on the computer screen and had to respond only when a square was presented at this location, and ignore circles when they were presented either at the attended location or at unattended locations. To reduce the amount of data to process, in this small dataset, we concentrated on targets (i.e. "**square**") presented at the two left visual field attended locations mentioned above for a single subject. For more details about the experiment see *Makeig et al, Science, 2002, vol. 295, pp 690−694*.

I.1.5. About this dataset

Selecting **Edit > About this dataset**, a text−editing window pops up allowing the user to write/modify a description of the current dataset. For the sample data, we entered the following description of the task. Press **SAVE** when done.

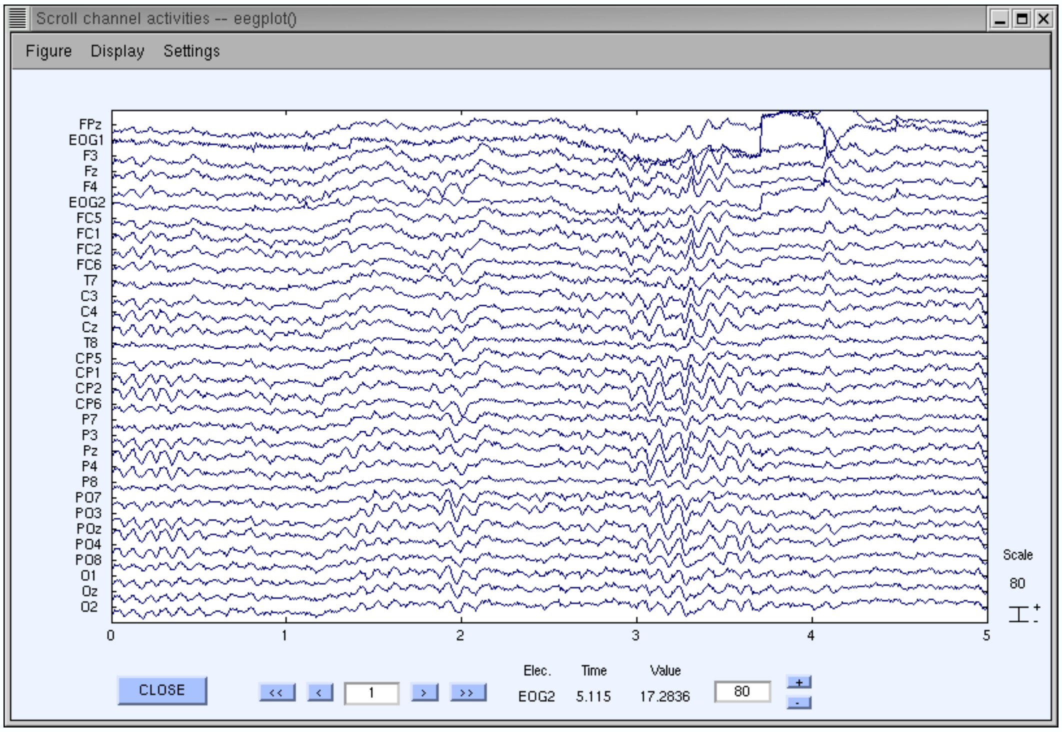


Stimulus can appear at 5 locations
One of them is surrounded by a box
If a square appears in the box the subject must respond, otherwise if it is a circle he must ignore the stimulus.

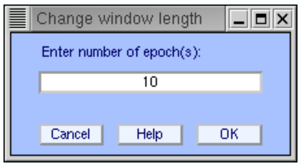
These data concerns one subject for circles only (non-targets) appearing at the attended location (i.e. surrounded by a box) on the left visual field

I.1.6. Scrolling through the data

To scroll through the dataset data, select the top **Plot** menu item, **Plot > Channel data (scroll)** . This pops up the **eegplot()** scrolling data window below. Note that the sample data file does not actually contain continuous EEG data. To reduce (your) download time, this "pseudo−continuous" EEG dataset was actually constructed by concatenating 80 separate three−second data epochs.



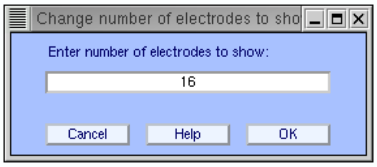
To the right of the plotting window is the scale value (unit, i.e. microvolt). Changing the "**Scale**" edit−text box value to about 50 (either by repeatedly clicking on the "**—**" button or by editing the text value (use your keyboard "Backspace" button, type the characters "50," and press the "Enter" key) displays magnified views of the EEG data channels. To adjust the time range shown, select **eegplot()** menu item **Settings > Time range to display**, and set the desired window length to "10" epochs as shown below



Enter number of epoch(s):

10

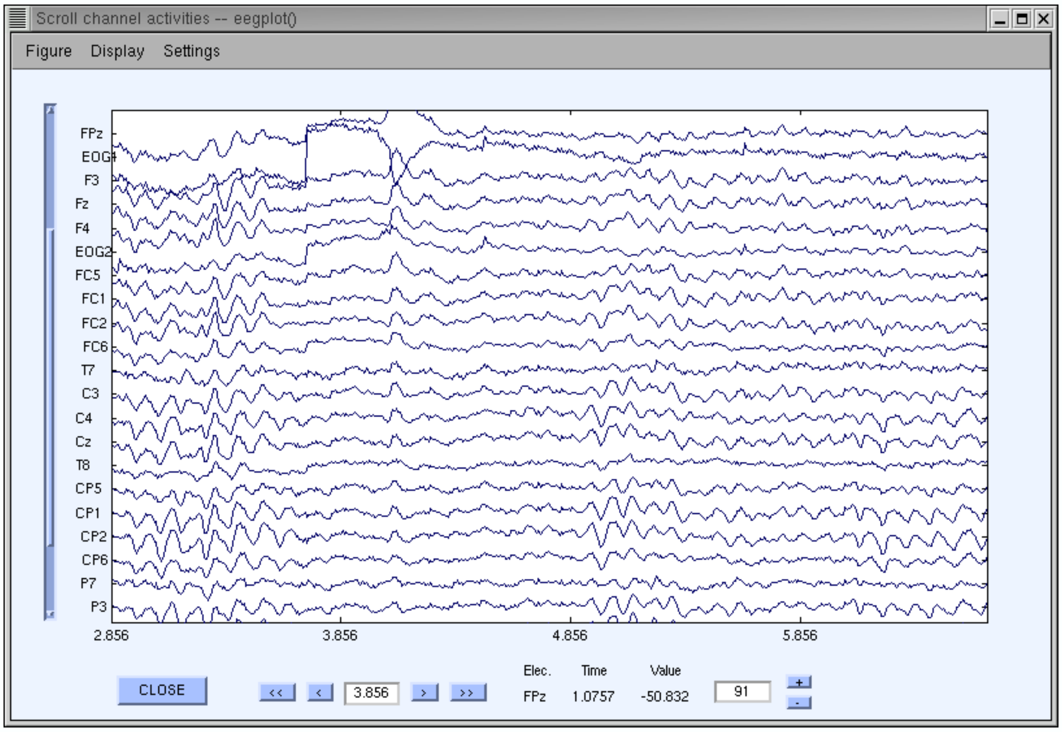
To adjust the number of channels displayed, select **Settings > Number of channels to display** and enter the desired number of channels to display in the screen (for instance "16"). This will return a scrolling **eegplot()** window with a vertical channel−set slider on the left of the plot. Use it to scroll vertically through all the channels.



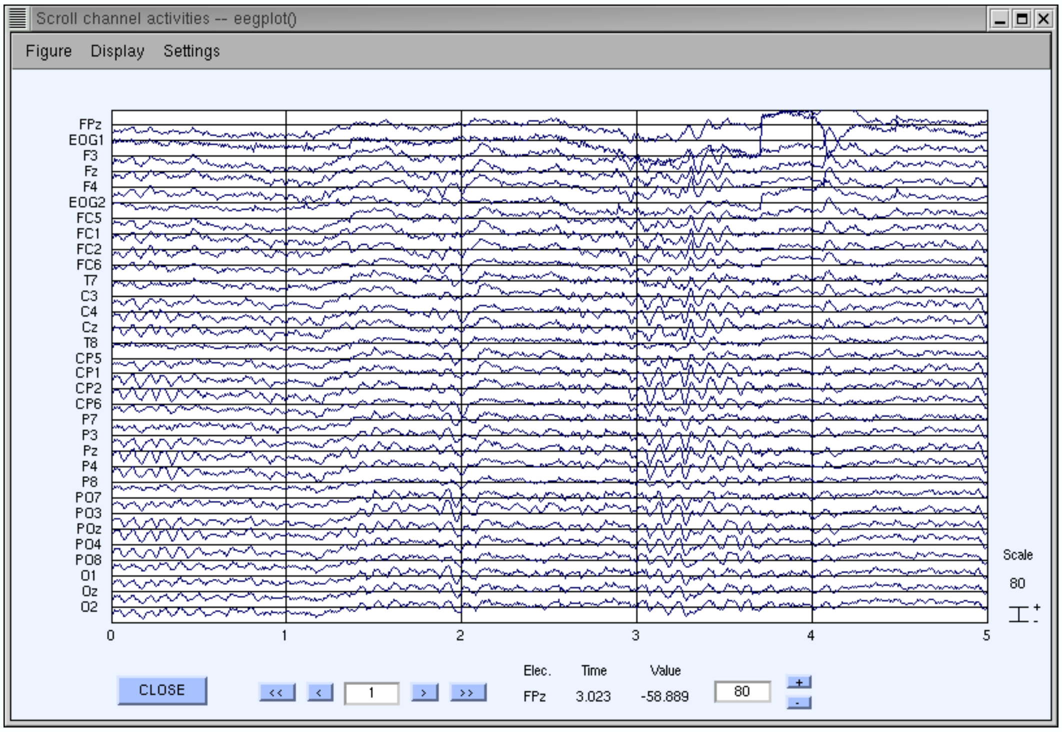
Enter number of electrodes to show:

16

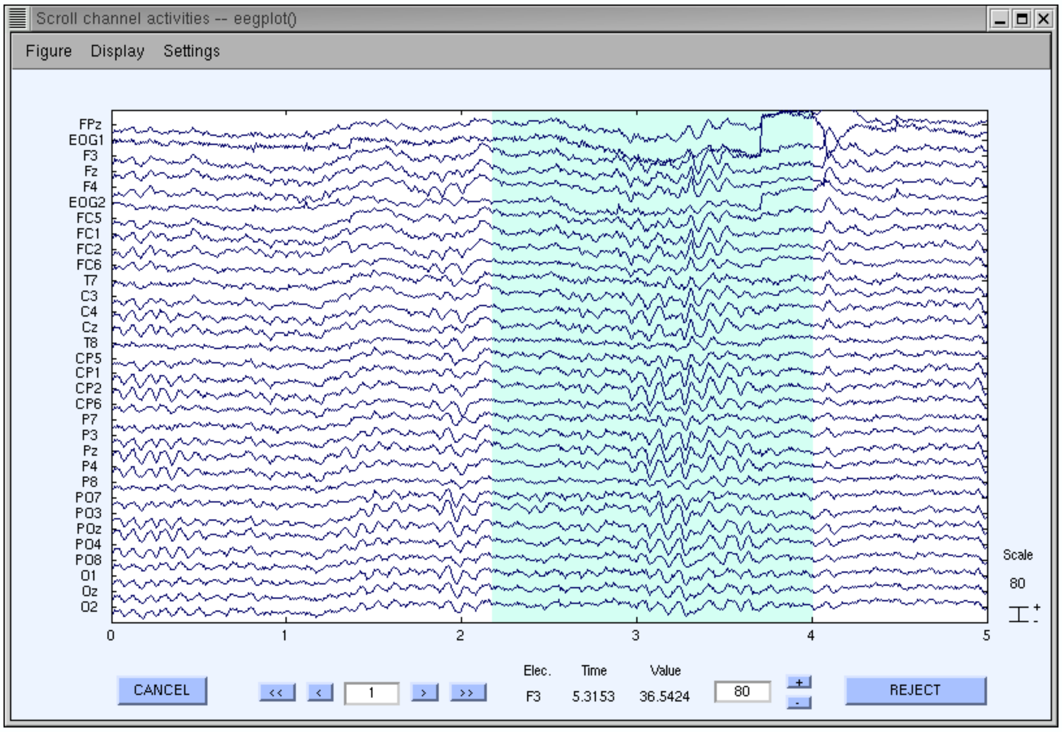
To zoom in on a particular area of a data window, select menu **Settings > Zoom off/on > Zoom both**. Now using your mouse, drag a rectangle around an area of the data to zoom in on. The scrolling window may now look similar to the following. Click the right button on the mouse to zoom out again. Use menu **Setting > Zoom off/on > Disable zoom** to turn off the zoom option.



To display horizontal (x) and vertical (y) grid lines on the data, select **Display > Grid > x axis** or **Display > Grid > y axis**. Repeat this process to turn off either set of grid lines.



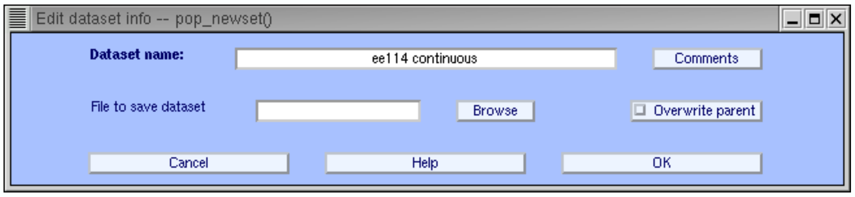
Interestingly, the **eegplot()** window also allows you to reject (erase) arbitrary portions of the continuous data. Actually, in the main EEGLAB menu, **eegplot()** is called both menu items **Plot > Scroll data** and **Tools > Reject continuous data** but when call from **Tools > Reject continuous data** a "**REJECT**" button appears on the bottom right corner: close the current **eegplot()** window and call this menu. To erase a selected portion of the data, first drag the mouse along the area of interest to mark it for rejection. It is possible to mark multiple regions. To undo a rejection mark, click once on the marked region. **Note:** zoom must be disabled in order to select a portion of the data.



Elec. F3 Time 5.3153 Value 36.5424 80

Now, to erase the marked data regions, click the (lower right) "**REJECT**" button (above). A new dataset will be created with the marked regions removed. (Note: EEGLAB will also add new "rejection boundary" events to the new dataset event list. These insure that subsequent epoch selections will not cross non−contiguous rejection boundaries). For more details about rejecting continuous data regions and data epochs, see the **reject data tutorial**.

Click "**OK**" (below) to create the new dataset with the marked data portions removed.

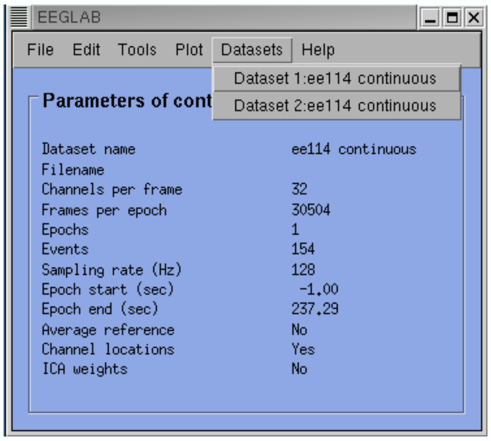


Dataset name: ee114 continuous

File to save dataset

☐ Overwrite parent

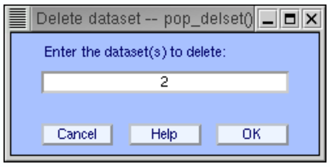
Since we only performed this rejection for illustrative purposes, switch back to the original dataset by selecting main window menu item **Datasets > Dataset 1 eeglab_data.**



Parameters of cont

```
Dataset name                    ee114 continuous
Filename
Channels per frame              32
Frames per epoch                30504
Epochs                          1
Events                          154
Sampling rate (Hz)              128
Epoch start (sec)                -1.00
Epoch end (sec)                 237.29
Average reference               No
Channel locations               Yes
ICA weights                     No
```

To delete the newly created second dataset, select **File > Clear dataset(s)** or **Edit > Delete dataset(s)** and enter the dataset index, "**2**" as shown below, and press "**OK**".

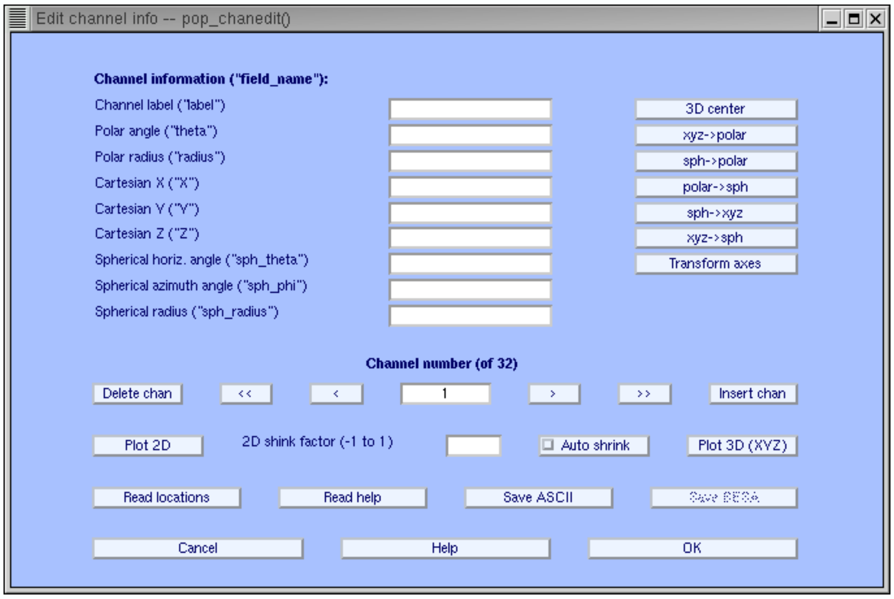


Enter the dataset(s) to delete: 2

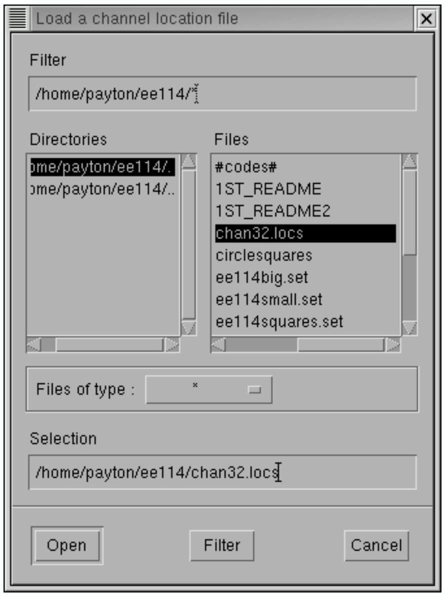
The second dataset will now be removed from the Matlab workspace. Note: it is not necessary to switch back to the first dataset before deleting the second. It is possible to delete several datasets at once from this window by entering their indices separated by spaces.

I.2. Using channel locations

To plot EEG scalp maps in either 2−D or 3−D format, an EEGLAB dataset must contain information about the channel locations. To load or edit channel location information contained in a dataset, select **Edit > Channel locations**



To load a channel location file, press the "**Read**" button and select the sample channel−locs file "**chan32.locs**" (press button "**Read help**" to see suported formats or see below).



Filter: /home/payton/ee114/*

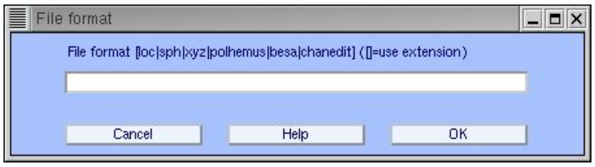
Directories: ome/payton/ee114/. ; ome/payton/ee114/..

Files: #codes#, 1ST_README, 1ST_README2, chan32.locs, circlesquares, ee114big.set, ee114small.set, ee114squares.set

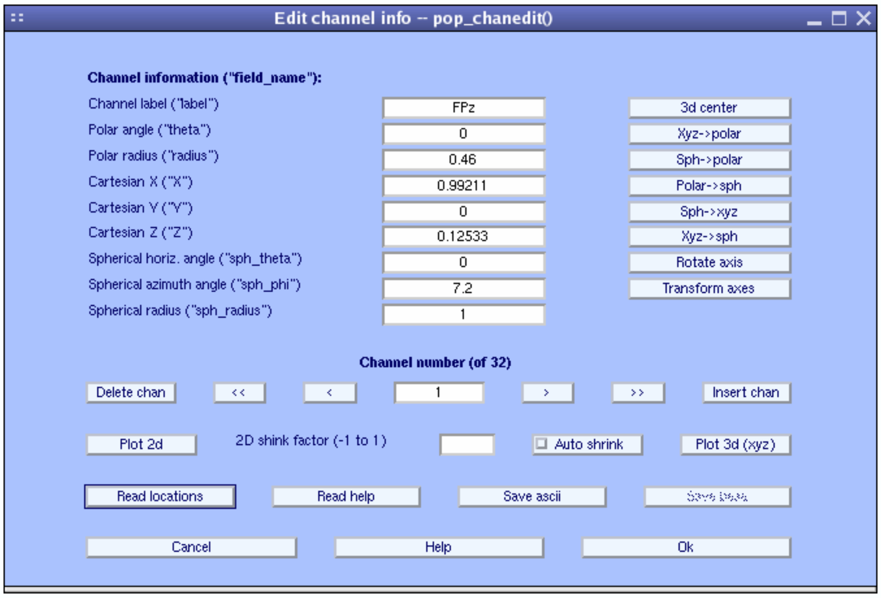
Files of type : *

Selection: /home/payton/ee114/chan32.locs

In the next pop up window, if you do not specify the file format, the **pop_chanedit()** function will attempt to use the filename extension to assess its format. Simply press "**OK**" below.


File format
File format [loc|sph|xyz|polhemus|besa|chanedit] ([]=use extension)
Cancel
Help
OK

Now in the **pop_chanedit()** window the loaded channel labels and polar coordinates are displayed. Scroll through the channel field values using the "**<**" and "**>**" buttons.



Channel information ("field_name"):

Channel label ("label")	FPz
Polar angle ("theta")	0
Polar radius ("radius")	0.46
Cartesian X ("X")	0.99211
Cartesian Y ("Y")	0
Cartesian Z ("Z")	0.12533
Spherical horiz. angle ("sph_theta")	0
Spherical azimuth angle ("sph_phi")	7.2
Spherical radius ("sph_radius")	1

Channel number (of 32): 1

Supported channel location file formats

Four channels from a polar coordinates file (extension **.loc**, no (blue) header line necessary):

#	Deg.	Radius	Label
1	−18	,352	Fp1
2	18	.352	Fp2
3	−90	,181	C3
4	90	,181	C4

The same locations, from a spherical coordinates file (**.sph**):

#	Azimut	Horiz.	Label
1	−63.36	−72	Fp1
2	63.36	72	Fp2
3	32.58	0	C3
4	32.58	0	C4

The same locations from a Cartesian coordinates file (**.xyz**):

#	X	Y	Z	Label
1	−0.8355	−0.2192	−0.5039	Fp1
2	−0.8355	0.2192	0.5039	Fp2
3	0.3956	0	−0.9184	C3

4	0.3956	0	0.9184	C4

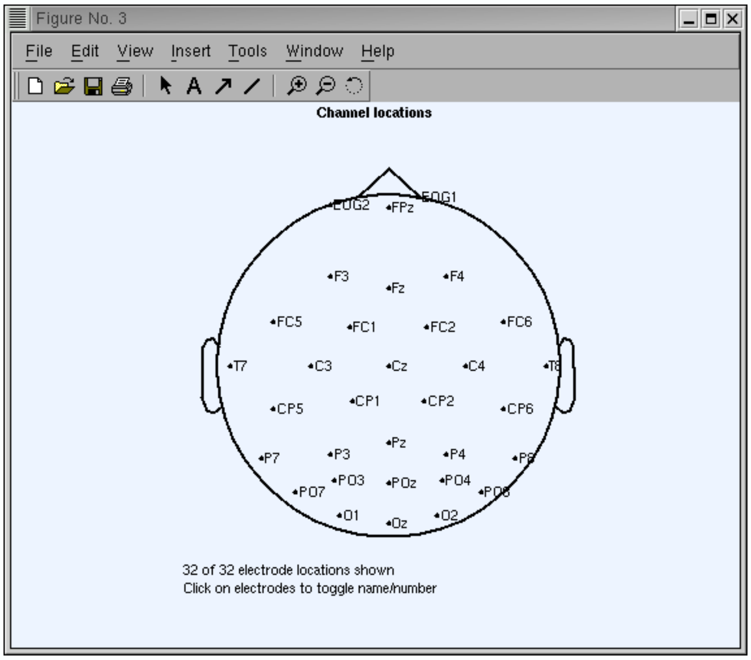
(Note: in all the above examples, the first header line must not be present in the file)

Other supported channel location file formats

- Polhemus (**.elp**) files
- Besa spherical coordinates (**.elp**)
- Location files saved by the **pop_chanedit()** function

Note that **pop_chanedit()** and **readlocs()** can also be called from the command line to convert between location formats. For more details see the help message of **readlocs()**.

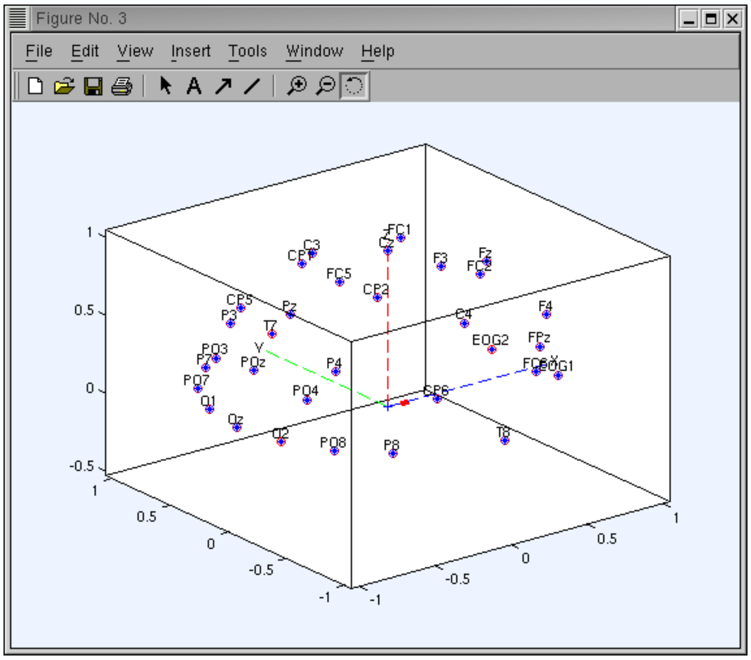
To **view** the 2−D location of the channels, press "**Plot 2D**" above the "**Read Locations**" button. Else, at any time during an EEGLAB session you may refer to a plot showing the channel locations by selecting **Plot > Channel location > By name**. Either pops up a window like that below. Note: In this plot, click on the channel labels to see the channel number.



32 of 32 electrode locations shown
Click on electrodes to toggle name/number

All channels of this montage are visible in the 2−D view. If the montage contained locations for which the radius values (in polar coordinates) were larger than 0.500, those locations would not appear on the top−down 2−D view. In this case, one could enter a 'shrink factor' in the **pop_chanedit()** window to make them appear in the 2−D view. Note: If a shrink factor is applied, EEGLAB still remembers the original coordinates, so the channel locations can still be accurately converted to 3−D (spherical or XYZ).

To visualize the channel locations in 3−D, first convert the locations into spherical coordinates by pressing the "**polar−>sph**" button on the right (press "**OK**" when the warning message appears). Then follow the same procedure using the "**sph−>xyz**" button, and finally press "**Plot 3D (XYZ)**". The window below will appear. The plotting box can be rotated in 3−D using the mouse:

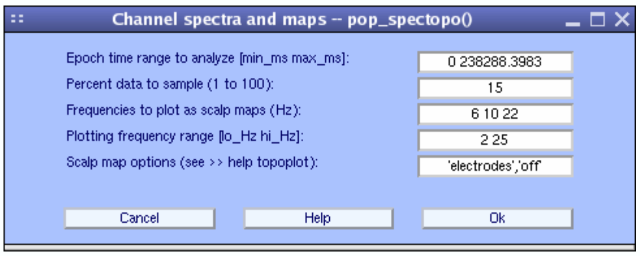


Important: press OK on the main channel editing window so that channel locations are actually imported. In the main EEGLAB window, the "**channel location**" flag now shows "yes".

I.3. Plotting channel spectra and maps

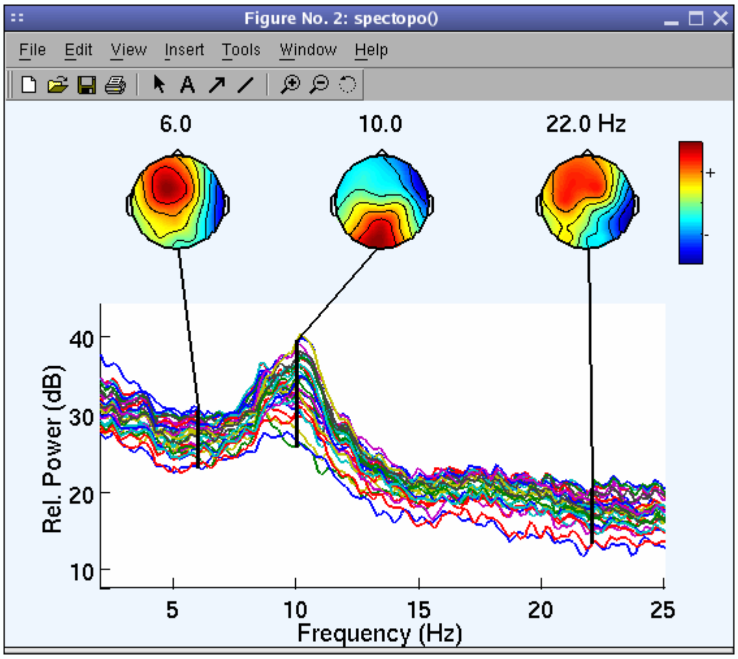
To begin processing the data, we recommend first scrolling the data as shown before (rejecting obviously 'bad' data portions or epochs), then studying their power spectra to be sure that the loaded data are suitable for further analysis. Note that the Matlab signal processing toolbox is required to use these functions.

To plot the channel spectra and associated topographical maps, select **Plot > Channel spectra and maps**. This will pop up the pop_spectopo() window (below). Leave the default settings and press "**OK**".


Channel spectra and maps -- pop_spectopo()

Epoch time range to analyze [min_ms max_ms]:	0 238288.3983
Percent data to sample (1 to 100):	15
Frequencies to plot as scalp maps (Hz):	6 10 22
Plotting frequency range [lo_Hz hi_Hz]:	2 25
Scalp map options (see >> help topoplot):	'electrodes','off'

The function should return a spectopo() plot (below). Note that since we only sampled 15% of the data trials ("**Percent data...**" edit box above), results slightly change for each call (this does not happen if you enter 100 % in the edit box).



Each colored trace represents the spectrum of one electrode or data channel. The leftmost scalp map shows the scalp distribution of power at 6 Hz, which in these data is concentrated above the frontal midline. The other scalp maps indicate the power scalp map at 10 Hz and 22 Hz.

The pop_spectopo() window menu (above) allows the user to compute and plot spectra in specific time windows in the data. The "**Percent data...**" argument can be used to speed the computation (by entering a number less than the default) or to return definitive measures (by increasing it to 100).

Note: Functions pop_spectopo() and spectopo() can also work with epoched data.

Another menu item, **Plot > Channel properties**, plots the scalp location of a selected channel, its activity spectrum, and an ERP−image plot of its activity in single−epochs.

The next tutorial section deals with preprocessing options available in EEGLAB.

I.4. Preprocessing tools

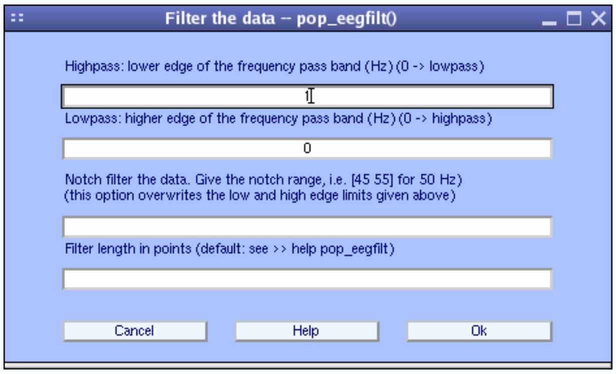
The upper portion of the **Tools** menu may be used to call three data preprocessing routines:

I.4.1. Changing the data sampling rate

The most common use for **Tools > Change sampling rate** is to reduce the sampling rate to save memory and disk storage. A pop_resample() window pops up, asking for the new sampling rate. The function uses Matlab **resample()** (in the Matlab Signal Processing toolbox−− If you do not have this toolbox, the **Change sampling rate** menu selection will be dimmed). We will not use this function now since the tutorial EEG dataset is already at an acceptable sampling rate.

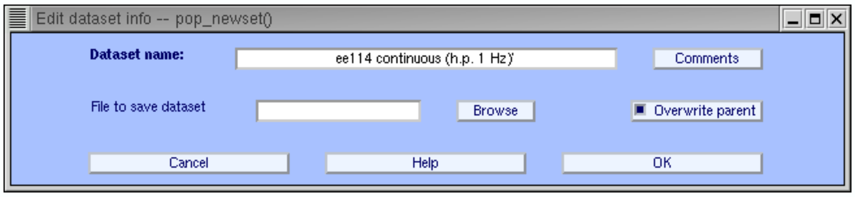
I.4.2. Filtering the data

To remove linear trends, it is sometimes necessary to high pass filter the data. To do so, select **Tools > Filter the data**, enter "**1**" (Hz) as "**Lower edge**" and press "**OK**".



Highpass: lower edge of the frequency pass band (Hz) (0 -> lowpass)
1
Lowpass: higher edge of the frequency pass band (Hz) (0 -> highpass)
0
Notch filter the data. Give the notch range, i.e. [45 55] for 50 Hz)
(this option overwrites the low and high edge limits given above)
Filter length in points (default: see >> help pop_eegfilt)

A **pop_newset()** window pops−up to ask for the name of the new dataset. Above, we choose to modify the dataset name and to overwrite the parent dataset.

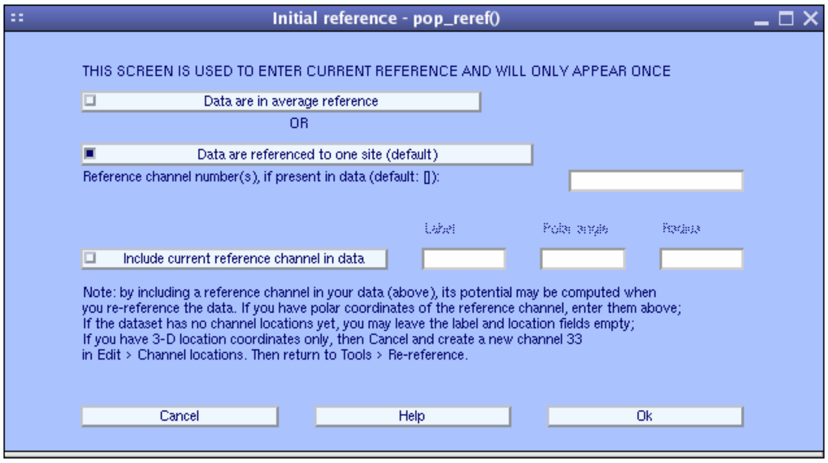


Dataset name: ee114 continuous (h.p. 1 Hz)
File to save dataset
Overwrite parent ☑

Another common use for data filtering is to remove 50 Hz or 60 Hz line noise. Note: we recommend to filter continuous EEG data although epoched data can also be filtered, each epoch being filtered separately. The called Matlab routine (**filtfilt()**) minimizes the appearance of artifacts at epoch boundaries.

I.4.3. Re−referencing the data

Converting data from fixed reference (earlobe or etc.) to average reference is advocated by some researchers, particularly when the electrode montage covers nearly the whole head (as in some high−density montages). Selecting **Tools > Re−reference** help convert the dataset to average reference by calling the **pop_reref** function. When you call this menu for the first time with a dataset, the following window pops up.


Initial reference - pop_reref()

THIS SCREEN IS USED TO ENTER CURRENT REFERENCE AND WILL ONLY APPEAR ONCE

☐ Data are in average reference

OR

■ Data are referenced to one site (default)

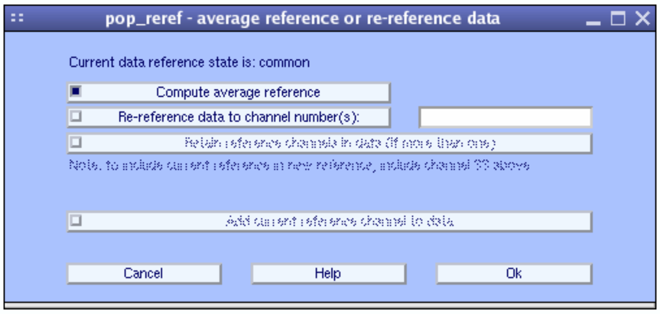
Reference channel number(s), if present in data (default: []):

Label | Polar angle | Radius

☐ Include current reference channel in data

Note: by including a reference channel in your data (above), its potential may be computed when
you re-reference the data. If you have polar coordinates of the reference channel, enter them above;
If the dataset has no channel locations yet, you may leave the label and location fields empty;
If you have 3-D location coordinates only, then Cancel and create a new channel 33
in Edit > Channel locations. Then return to Tools > Re-reference.

The data above was recorded using a mastoid reference. Since we do not wish to include this reference channel neither in the data nor in the average reference calculation, we will not click the "**Include current reference channel in data**" checkbox (click the checkbox for references such as Cz). Now, press the "**Ok**" button and the re−referencing window below appears.


pop_reref - average reference or re-reference data

Current data reference state is: common

■ Compute average reference

☐ Re-reference data to channel number(s):

☐ Retain reference channels in data (if more than one)

Note: to include current reference in new reference, include channel 33 above

☐ Add current reference channel to data

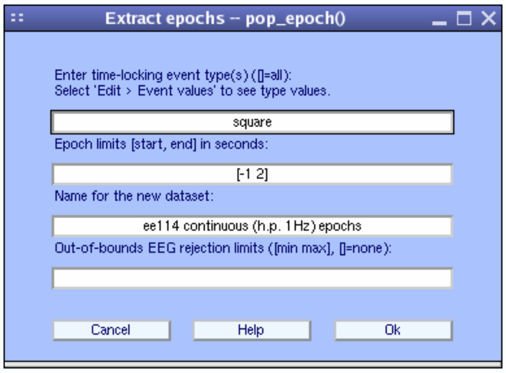
Press the "**Ok**" button to compute the average reference. This fact is then indicated in the main EEGLAB window (not shown).
After data has been average referenced, calling the **Tools > Re−reference** menu still allows you to re−reference to any channel or group of channel (or undo average reference if you previously choose to include the initial reference channel in the data). **Note:** this function also re−reference the ICA weights if any.

The next tutorial deals with extracting data epochs from continuous or epoched datasets.

I.5. Extracting data epochs

I.5.1. Extracting epochs

To study the event−related EEG dynamics of continuously recorded data, we must extract data epochs time locked to events of interest (for example, data epochs time locked to onsets of one class of experimental stimuli). To extract data epochs from continuous data, select **Tools > Extract epochs**. In the top text box of the resulting pop_epochs() window, enter event type "**square**" (onsets of target stimuli in our experiment). Note: To review the available event types for the current dataset, see **Edit > Event values**. Use the default epoch limits (from 1 second before to 2 seconds after the time−locking event), and press "**OK**".

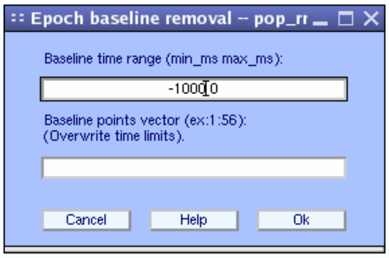


Enter time-locking event type(s) ([]=all):
Select 'Edit > Event values' to see type values.
square
Epoch limits [start, end] in seconds:
[-1 2]
Name for the new dataset:
ee114 continuous (h.p. 1Hz) epochs
Out-of-bounds EEG rejection limits ([min max], []=none):

In this example, the stimulus−locked windows are 3 seconds long. It is often better to extract long time windows, as in this case, to make time−frequency decomposition possible at low frequencies. After the data has been epoched, a pop−up window asks for removing baseline values as shown below.

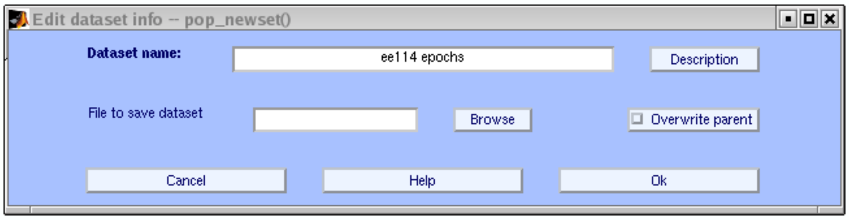
I.5.2. Removing baseline values

Removing a mean baseline value from each epoch is useful when baseline differences between data epochs (e.g., arising from low frequency drifts or artifacts) are not meaningful and might otherwise dominate the data. The following window pops up automatically after the data has been epoched. It is also possible to call it by selecting menu **Tools > Remove baseline**. Specify the baseline period of each epoch (in ms or points) to remove the baseline from your dataset (the original epoched dataset is automatically overwriten by the baseline−removed dataset). **Note:** There is no uniformly 'optimum' method of selecting either the baseline period or the baseline value. Using the mean value in the pre−stimulus period (the pop_rmbase() default) may be efficient for many datasets. Press "**OK**" to subtract baseline ("**Cancel**" would preserve the original baseline).



Baseline time range (min_ms max_ms):
-1000 0
Baseline points vector (ex:1:56):
(Overwrite time limits).

Then another window pops up to query for the name of the new dataset. We choose to use the default, "**eeglab_data epochs**". At this point, it is useful to edit the dataset comments to record the exact nature of the new dataset (by pressing "**Comments**").

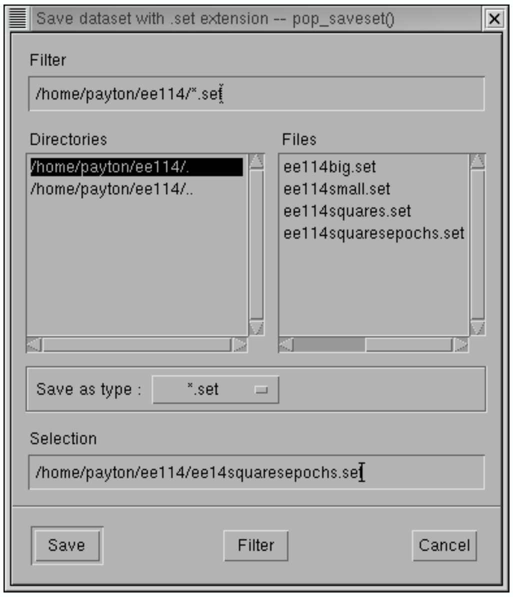


Dataset name: ee114 epochs

File to save dataset

☐ Overwrite parent

It may be better to save the epoched dataset under a new name (press "**Browse**"), for we will be working extensively with these data epochs, but want to preserve the continuous dataset on the disk separately to allow later arbitrary re−epoching. The file−browser window below appears. Enter a name for the dataset (ending with **.set**) and press "**SAVE**" (below) and then "**OK**" (above). **Note:** To save a dataset at any moment, select **File > Save current dataset** from the main EEGLAB window.



Filter: /home/payton/ee114/*.set

Directories:
- /home/payton/ee114/.
- /home/payton/ee114/..

Files:
- ee114big.set
- ee114small.set
- ee114squares.set
- ee114squaresepochs.set

Save as type : *.set

Selection: /home/payton/ee114/ee14squaresepochs.set

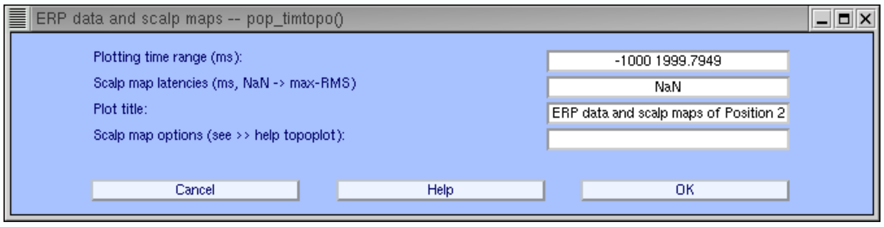
The next tutorial deals with averaging epoched datasets.

I.6. Data averaging

In a recent paper, we have argued that event−related EEG dynamics occurring in a set of data epochs should not be confused with features of their time−locked average, the Event−Related Potential (ERP). EEGLAB contains several functions for plotting 1−D dataset (ERP) averages. We prefer to expand the study of 1−D (potential across times) ERPs of a set of event−related EEG epochs by, first, studying 2−D (potential at times−by−epochs) 'ERP−image' transforms of epoched data, in which potential data epochs are first sorted along some relevant dimension (for example, subject reaction time, theta power, 'P300' amplitude, alpha−phase at stimulus onset, etc.), then (optionally) smoothed and finally color−coded and visualized as a 2−D rectangular image.

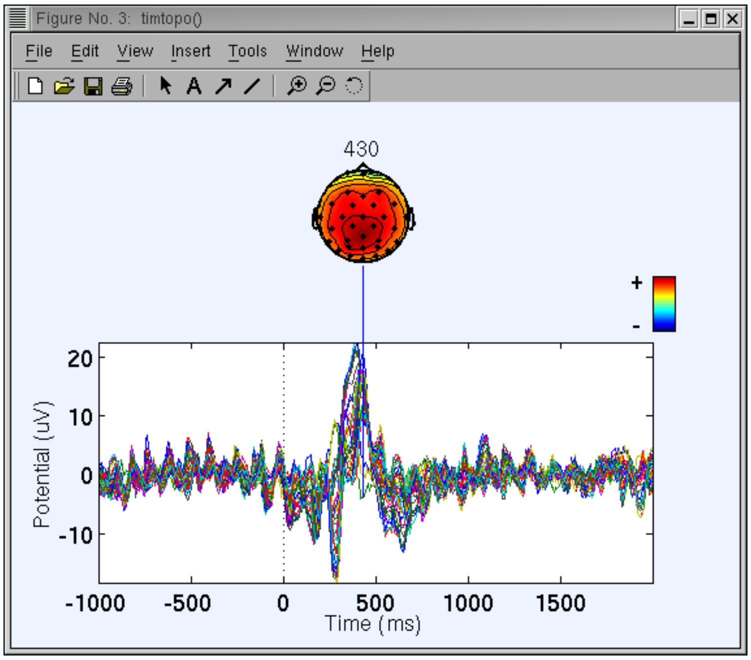
I.6.1. Plotting the ERP data on a single axis with scalp maps

To plot the (ERP) average of all dataset epochs, with selected scalp maps, select **Plot > Channel ERP > With scalp maps.** As a simple illustration, we keep the default settings in the resulting **pop_timtopo()** window, entering only a plot title and pressing "**OK**".


ERP data and scalp maps -- pop_timtopo()

Plotting time range (ms):	-1000 1999.7949
Scalp map latencies (ms, NaN -> max-RMS)	NaN
Plot title:	ERP data and scalp maps of Position 2
Scalp map options (see >> help topoplot):	

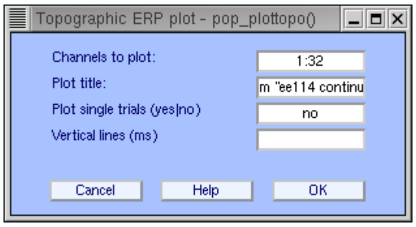
A **timtopo()** figure (below) appears. Each trace plots the averaged ERP at one channel. The scalp map shows the topographic distribution of average potential at latency 430 ms (the epoch latency with maximum variance in the ERP). One or more latencies for plotting scalp maps can also be specified in the pop−window above.



Function **timtopo()** plots show the relative time courses of the averaged ERP at all channels, plus "snapshots" of the changing scalp potential distribution during the ERP epoch. **Note:** To visualize **all** the scalp−map snapshots as an "ERP movie" (i.e., in native ERP format!), call function **eegmovie()** from the command line.

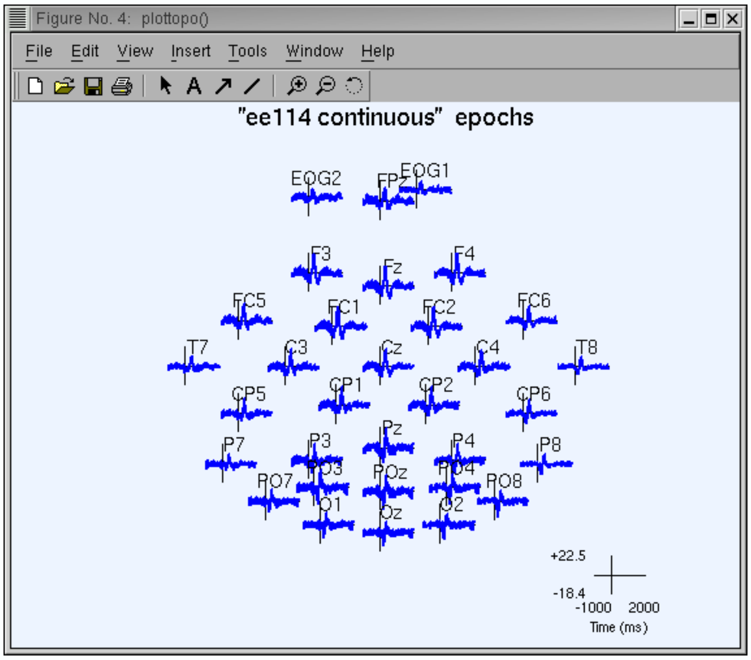
I.6.2. Plotting ERP traces in a topographic array

To plot the ERP average of an epoched dataset as single−channel traces in topographic order, select **Plot > Average ERP > In scalp array**. Using the default settings, adding a **Plot title**, and pressing **OK** in the resulting pop_plottopo() window (below)

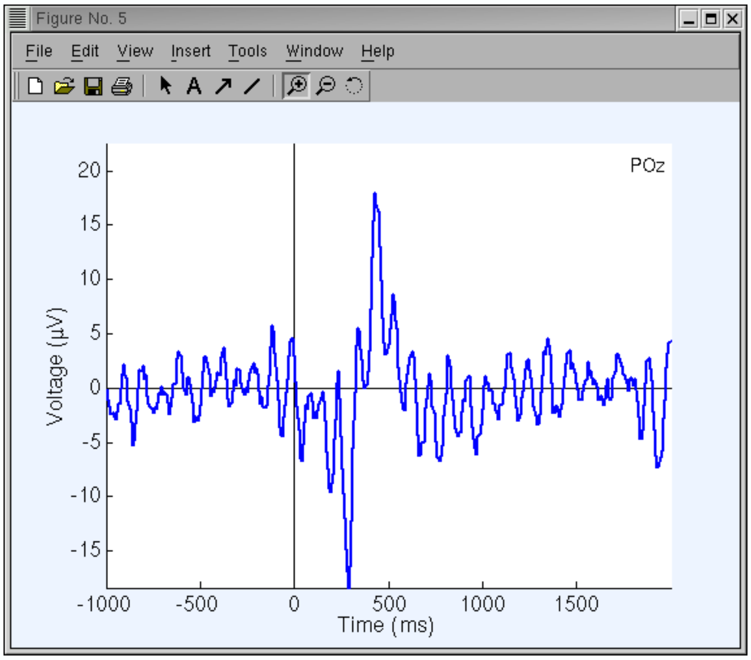

Topographic ERP plot - pop_plottopo()

Channels to plot:	1:32
Plot title:	m "ee114 continu
Plot single trials (yes|no)	no
Vertical lines (ms)	

produces the following plottopo() figure.



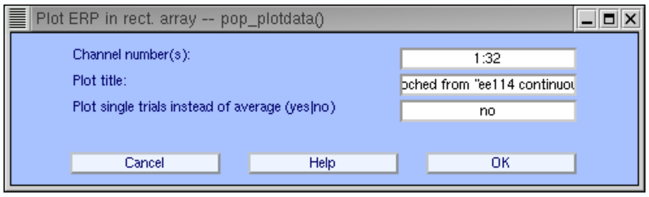
You may visualize a specific channel time course by clicking on its trace (above) producing a pop−up sub−axis view (as below)



Many EEGLAB plotting routines use toolbox function **axcopy()** to pop−up a sub−axis plotting window whenever the users clicks on the main plot window. Sub−axis windows, in turn, do not have **axcopy()** enabled, allowing the user to use the standard Matlab mouse Zoom In/Out feature.

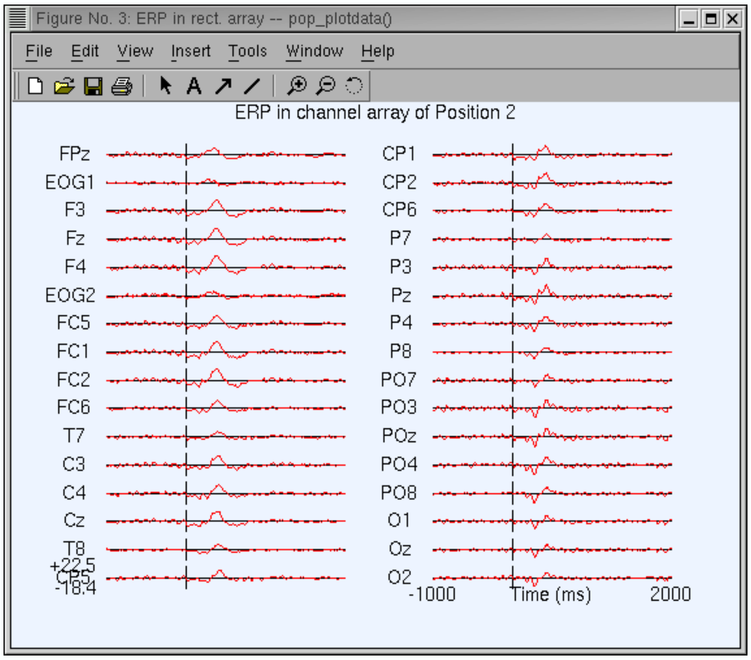
I.6.3. Plotting ERPs in rectangular array

To plot (one or more) average ERP data traces in a rectangular array, go to the menu **Plot > Average ERP > In rect. array**. Choose to use the default settings from the resulting **pop_plotdata()** window and press **OK**.


Plot ERP in rect. array -- pop_plotdata()

Channel number(s): 1:32
Plot title: pched from "ee114 continuou
Plot single trials instead of average (yes|no) no

The resulting **plotdata()** figure (below) appears.

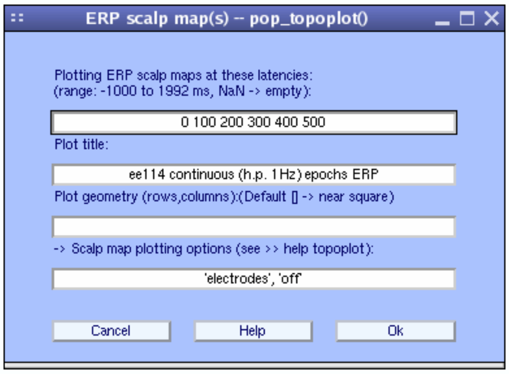


As for the previous plot, clicking on a trace above produces a pop−up sub−axis view.

I.6.4. Plotting an ERP as a series of maps

I.6.4.1. Plotting a series of 2−D ERP scalp maps

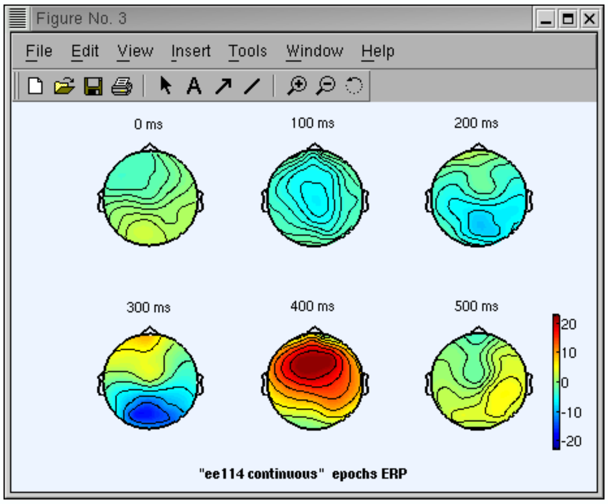
To plot ERP data for a series of 2−D scalp maps representing potential distributions at a selected series of times during the epoch, select **Plot > ERP map series > In 2−D**. In the top text box of the resulting **pop_topoplot()** window (below), type the desired latencies for the ERP scalp maps. **Note:** In this or any other numeric text entry box, you may enter any numeric Matlab expression For example, instead of "**0 100 200 300 400 500**", try "**0:100:500** ". Expressions such as "**−6000+3*(0:20:120)**" are also accepted.


ERP scalp map(s) -- pop_topoplot()

Plotting ERP scalp maps at these latencies:
(range: -1000 to 1992 ms, NaN -> empty):
0 100 200 300 400 500
Plot title:
ee114 continuous (h.p. 1Hz) epochs ERP
Plot geometry (rows,columns):(Default [] -> near square)

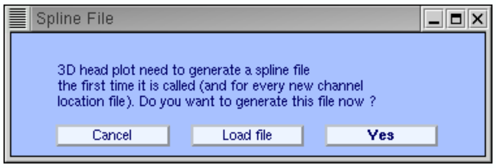
-> Scalp map plotting options (see >> help topoplot):
'electrodes', 'off'

The **topoplot()** window (below) appears containing ERP scalp maps at the specified latencies. Here, the plot grid has 3 columns and 2 rows; other plot geometries can be specified in the interactive window (above) via the **Plot geometry** text box.



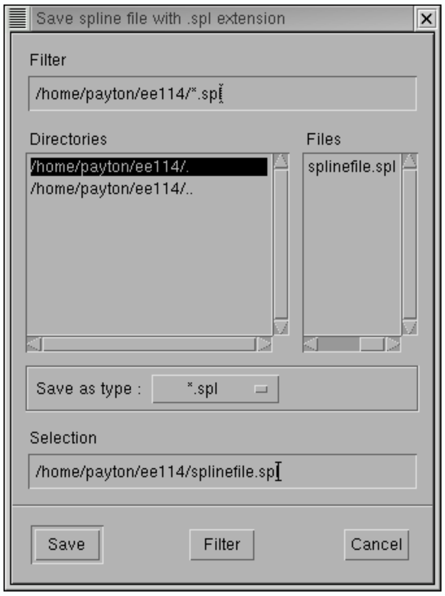
I.6.4.2. Plotting ERP data as a series of 3−D maps

To plot ERP data as a series of 3−D scalp maps, go to the menu **Plot > ERP map series > In 3−D**. The query window (below) should pop up, asking you to create and save a new ***3−D spline file***. This process must be done only once for every montage. Click **YES** to begin this process.



3D head plot need to generate a spline file
the first time it is called (and for every new channel
location file). Do you want to generate this file now ?

Enter a spline file name (ending in **.spl**) in the resulting file−browser window (below).



Filter
/home/payton/ee114/*.spl

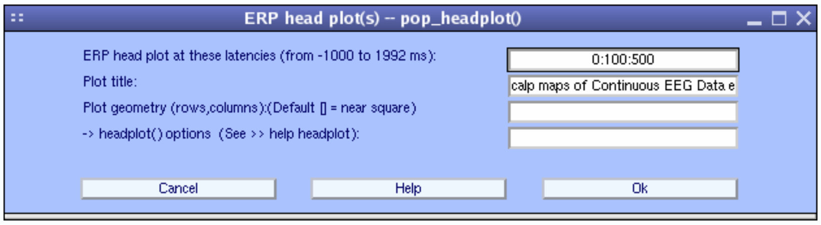
Directories
/home/payton/ee114/.
/home/payton/ee114/..

Files
splinefile.spl

Save as type : *.spl

Selection
/home/payton/ee114/splinefile.spl

Now **pop_headplot()**, the 3−D plotting function, will create the spline file. This process may require several minutes or more. A progress bar will pop up to indicate when it will be done. When the montage spline file has been generated, select **Plot > ERP map series > In 3−D** and now that the spline file has been generated, a different interactive window appears. As for the 2D scalp maps, in the first row type in the desired latencies, and press **OK**.



ERP head plot(s) -- pop_headplot()

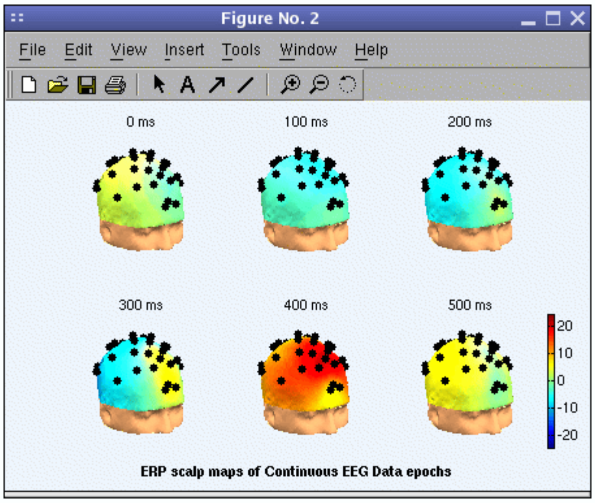
ERP head plot at these latencies (from -1000 to 1992 ms): 0:100:500

Plot title: calp maps of Continuous EEG Data e

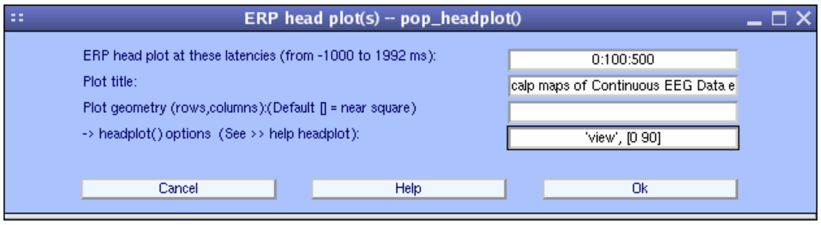
Plot geometry (rows,columns):(Default [] = near square)

-> headplot() options (See >> help headplot):

The **headplot()** figure (below) appears



As usual, clicking on a head plot makes it pop up in a sub−axis window, where it can be rotated using the mouse. To plot the heads at a specified angle, use the **headplot()** **'view'** option as in the example below.



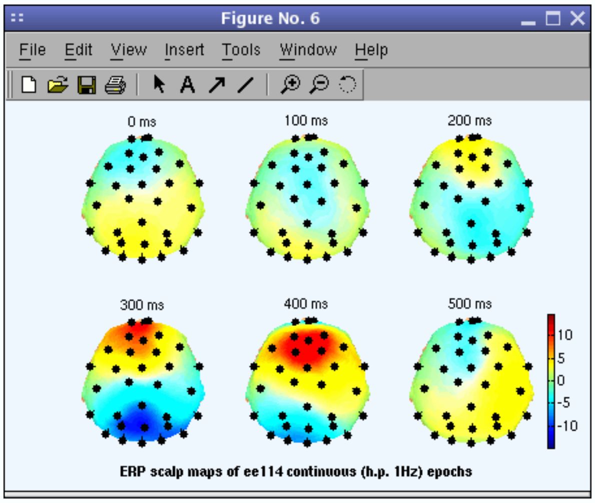
ERP head plot(s) -- pop_headplot()

ERP head plot at these latencies (from -1000 to 1992 ms): 0:100:500

Plot title: calp maps of Continuous EEG Data e

Plot geometry (rows,columns):(Default [] = near square)

-> headplot() options (See >> help headplot): 'view', [0 90] The headplot() window (below) then appears

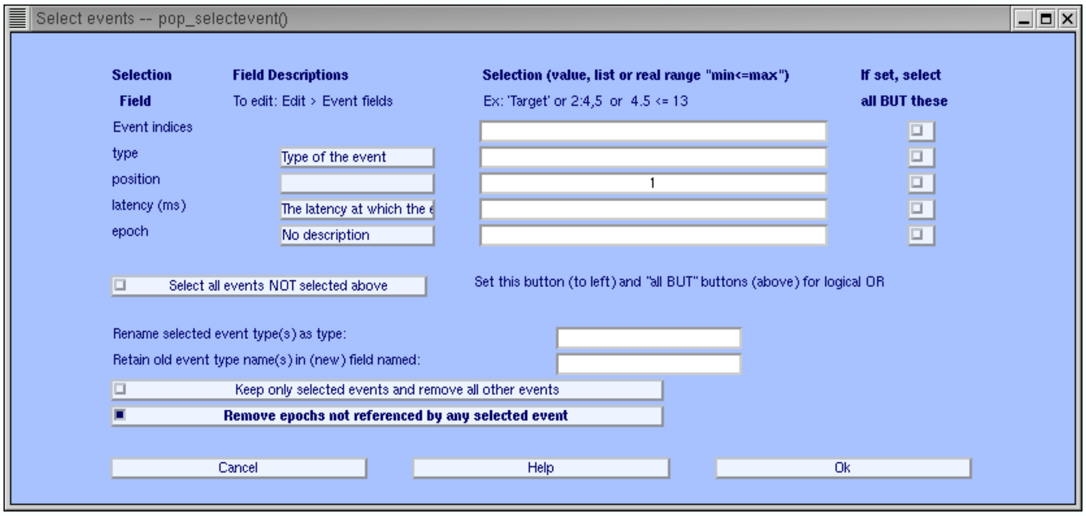


In the next tutorial, we show how to use EEGLAB to compare ERPs from two conditions.

I.7. Selecting data epochs and plotting data averages

I.7.1. Selecting events and epochs for two conditions

To compare event−related EEG dynamics for two or more conditions from the same experiment, it is first necessary to create datasets containing epochs for each condition. In the experiment of our sample dataset, half the targets appeared at position 1 and the other half at position 2 (see experiment description). Select **Edit > Select events**. The pop_selectevent() window (below) will appear. Select all epochs in which the target appeared in position "1" as below:


Select events -- pop_selectevent()

Selection Field	Field Descriptions (To edit: Edit > Event fields)	Selection (value, list or real range "min<=max") Ex: 'Target' or 2:4,5 or 4.5 <= 13	If set, select all BUT these
Event indices			☐
type	Type of the event		☐
position		1	☐
latency (ms)	The latency at which the e		☐
epoch	No description		☐

☐ Select all events NOT selected above — Set this button (to left) and "all BUT" buttons (above) for logical OR

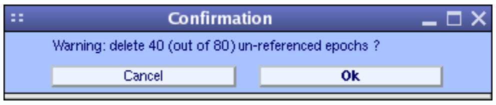
Rename selected event type(s) as type:

Retain old event type name(s) in (new) field named:

☐ Keep only selected events and remove all other events

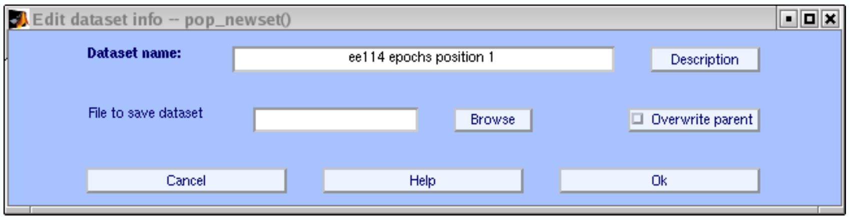
■ **Remove epochs not referenced by any selected event**

Press **Yes** in the resulting query window (below):


Confirmation

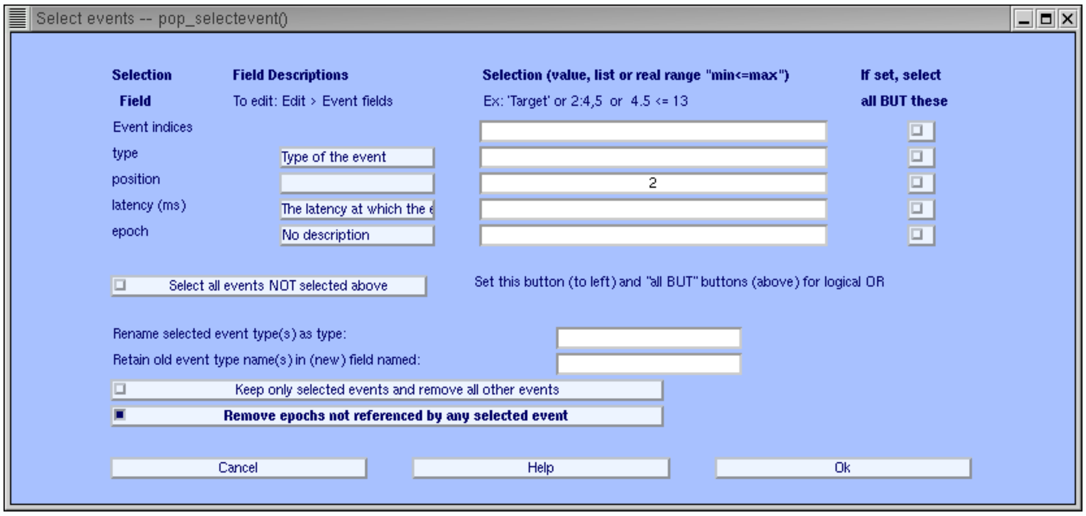
Warning: delete 40 (out of 80) un-referenced epochs ?

Now a **pop_newset()** window for saving the new dataset pops up. We name this new dataset "**Position 1**" and press "**OK**".


Edit dataset info -- pop_newset()

Dataset name:	ee114 epochs position 1		Description
File to save dataset		Browse	☐ Overwrite parent

Now, repeat the process to create a second dataset consisting of epochs in which the target appeared at position 2. First, **retrieve the original dataset** in the **Datasets** menu. Select the 2nd dataset ("**eeglab_data epochs**"). Select **Edit > Select epoch/events**. In the resulting **pop_selectevent()** window, enter "**2**" in the text box to the right of the "**position**" field. Press **OK**, then name the new dataset "**Position 2**".


Select events -- pop_selectevent()

Selection Field	**Field Descriptions** To edit: Edit > Event fields	**Selection (value, list or real range "min<=max")** Ex: 'Target' or 2:4,5 or 4.5 <= 13	**If set, select all BUT these**
Event indices			☐
type	Type of the event		☐
position		2	☐
latency (ms)	The latency at which the e		☐
epoch	No description		☐

☐ Select all events NOT selected above — Set this button (to left) and "all BUT" buttons (above) for logical OR

Rename selected event type(s) as type:

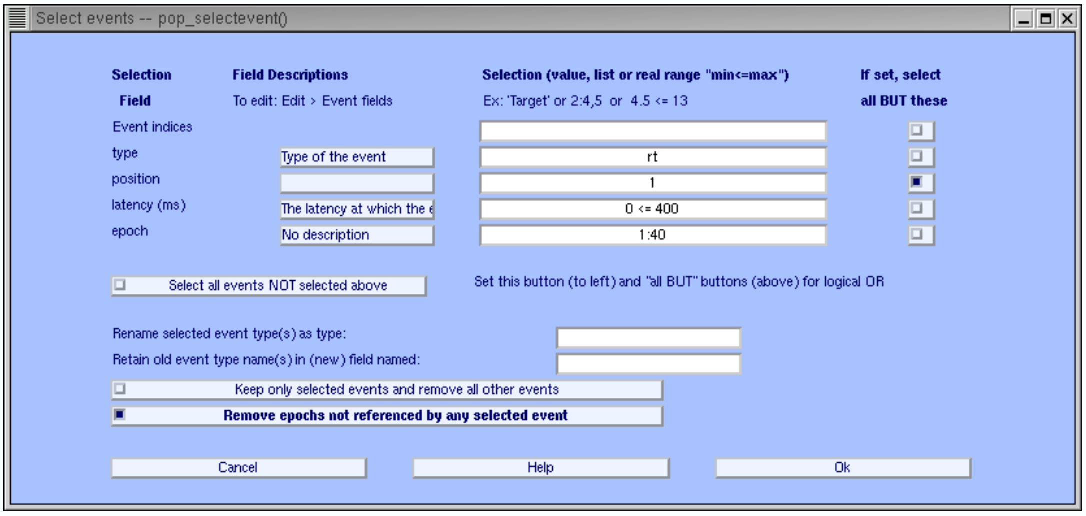
Retain old event type name(s) in (new) field named:

☐ Keep only selected events and remove all other events

■ **Remove epochs not referenced by any selected event**

About selecting events and epochs

Note that you can specify complex combinations of event field selections and ranges as the criterion to select trials. For instance the window below would select epochs containing events of type **rt**, **not** following stimuli presented at position 1, in which subject reaction time latency is between 0 and 400 ms from among the 40 first epochs of the parent dataset. Note the option set above the **Cancel** button (below): "**Remove epochs not referenced by any selected event**". If this checkbox was left unset and the checkbox "**Keep only selected events and remove all other events**", the function would simply select the specified events but not remove epochs not containing those events. Note: To select events outside the given range, check the "**Select events NOT selected above**" box to the right of the field range entry. It is possible to rename the type of the selected events (while keeping the old event type name in a new field (optional)) using the last two edit boxes.



Select events -- pop_selectevent()

Selection Field	Field Descriptions (To edit: Edit > Event fields)	Selection (value, list or real range "min<=max") Ex: 'Target' or 2:4,5 or 4.5 <= 13	If set, select all BUT these
Event indices			☐
type	Type of the event	rt	☐
position		1	■
latency (ms)	The latency at which the e	0 <= 400	☐
epoch	No description	1:40	☐

☐ Select all events NOT selected above

Set this button (to left) and "all BUT" buttons (above) for logical OR

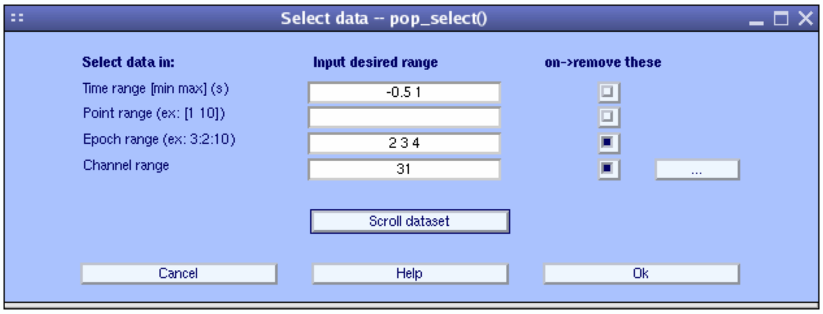
Rename selected event type(s) as type:

Retain old event type name(s) in (new) field named:

☐ Keep only selected events and remove all other events

■ Remove epochs not referenced by any selected event

Another function that can be useful for selecting a dataset subset is the function pop_select() called by selecting **Edit > Select data**. The example below would select data sub−epochs with the epoch time range from −500 ms to 1000 ms. It would, further, remove dataset epochs 2, 3 and 4 and remove channel 31 completely

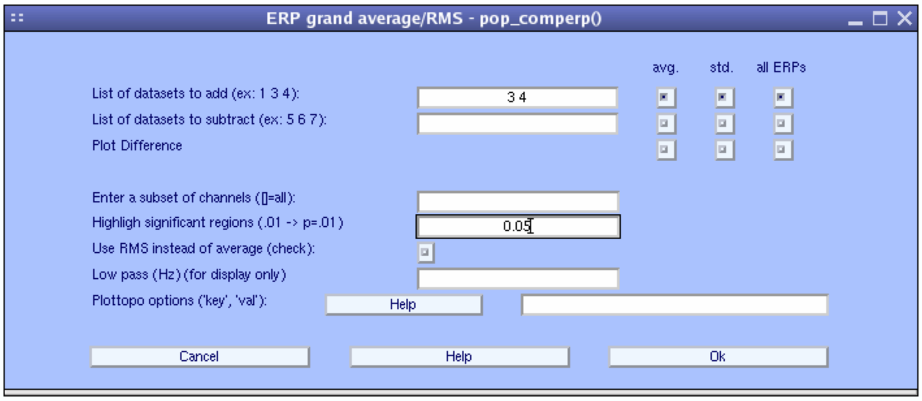


Select data -- pop_select()

Select data in:	Input desired range	on->remove these
Time range [min max] (s)	-0.5 1	☐
Point range (ex: [1 10])		☐
Epoch range (ex: 3:2:10)	2 3 4	■
Channel range	31	■

I.7.2. Computing Grand Mean ERPs

Normally, ERP researchers report results on grand mean ERPs averaged across subjects. As an example, we will use EEGLAB to compute the ERP average of the two conditions above. Select **Plot > Plot/Compare ERPs**. In the top text−entry boxes of the resulting pop_comperp() window (below), enter the indices of datasets 3 and 4. On the first row, click the "**avg.**" box to display grand average, the "**std.**" box to display standard deviation, and the "**all ERPs**" box to display ERP averages for each dataset. Finally 0.05 for the t−test probability (p) value. Then press **OK**.


ERP grand average/RMS - pop_comperp()

		avg.	std.	all ERPs
List of datasets to add (ex: 1 3 4):	3 4	☒	☒	☒
List of datasets to subtract (ex: 5 6 7):		☐	☐	☐
Plot Difference		☐	☐	☐

Enter a subset of channels ([]=all):

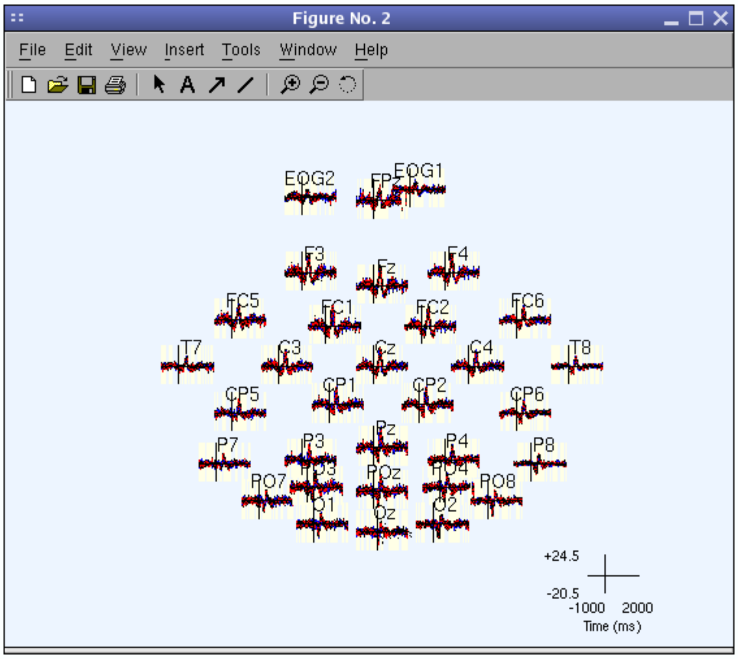
Highligh significant regions (.01 -> p=.01): 0.05

Use RMS instead of average (check): ☐

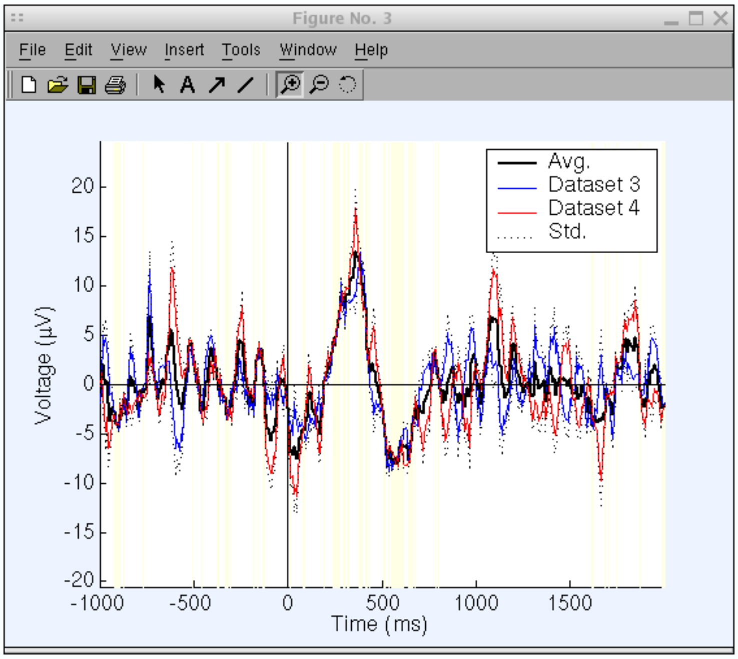
Low pass (Hz) (for display only)

Plottopo options ('key', 'val'): Help

The plot below appears.



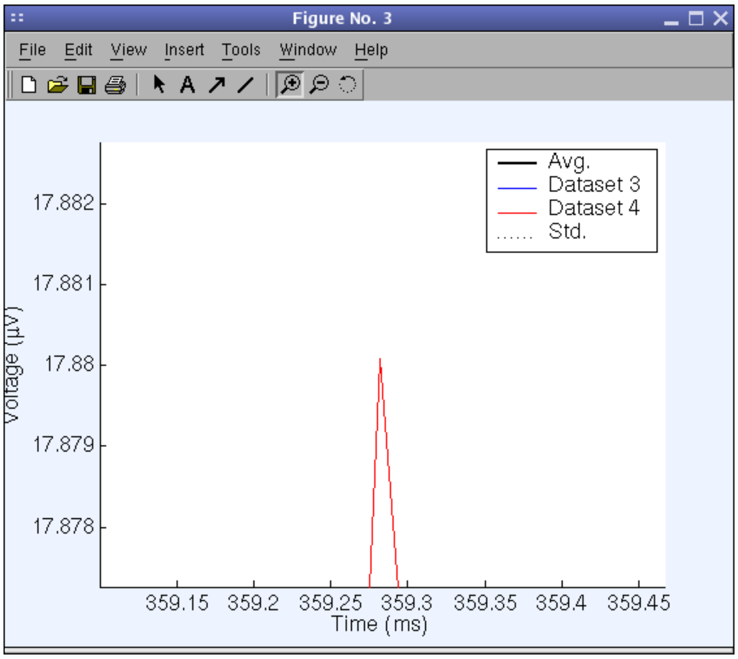
Now, click on the traces at electrode position FPz. <span style="color:red">**Note:**</span> if you prefer to use negative up view for the y scale, enter "**'ydir', −1**" in the "**Plottopo options**" field.



The ERPs for datasets 3 and 4 are shown in blue and red. The grand average ERP for the two conditions is shown in bold black, and the standard deviation of the two ERPs in dotted black. Regions which are significantly different from 0 are higlighted using a two−tailed t−test at each time point. This test compares the current ERP value distribution with a distribution of the same variance and 0 mean). Note that the t−test is not corrected for multiple comparisons. The p values at each time point can be obtained from a command line call to the function **pop_comperp()**.

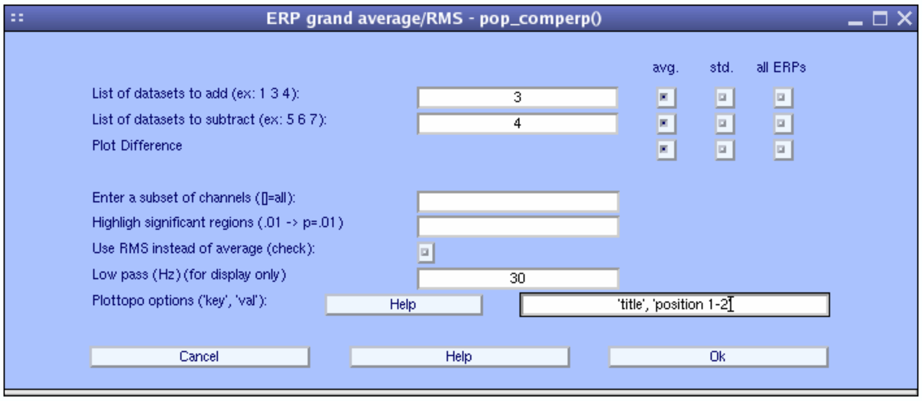
I.7.3. Finding ERP peak latencies

Although EEGLAB currently does not have tools for automatically determining ERP peak amplitudes and latencies, one can use the Matlab zoom facility to manually determine the latency of a peak in any Matlab figure. For example, in the figure above select the magnifying−glass icon having the the "**+**" sign. Then, zoom in on the main peak of the red curve as shown below (click on the left mouse button to zoom in and on the right button to zoom out). Read the peak latency and amplitude to any desired precision from the axis scale. **Note:** It may be desirable to first use the low pass filtering edit box of the **pop_comperp()** interface to smooth average data peaks before measuring latency peaks.



I.7.4. Comparing ERPs in two conditions

To compare ERP averages for the two conditions (targets presented in positions 1 and 2), select **Plot > Plot/Compare ERPs**. In the top text−entry box of the resulting **pop_comperp()** window (below), enter the indices of the datasets to compare. Click all boxes in the "**avg.**" column. Enter "**35**" for the low pass frequency and "**'title', 'Position 1−2'**" in the **topoplot()** option edit box. Then press **OK**.


ERP grand average/RMS - pop_comperp()

avg. std. all ERPs

List of datasets to add (ex: 1 3 4): 3

List of datasets to subtract (ex: 5 6 7): 4

Plot Difference

Enter a subset of channels ([]=all):

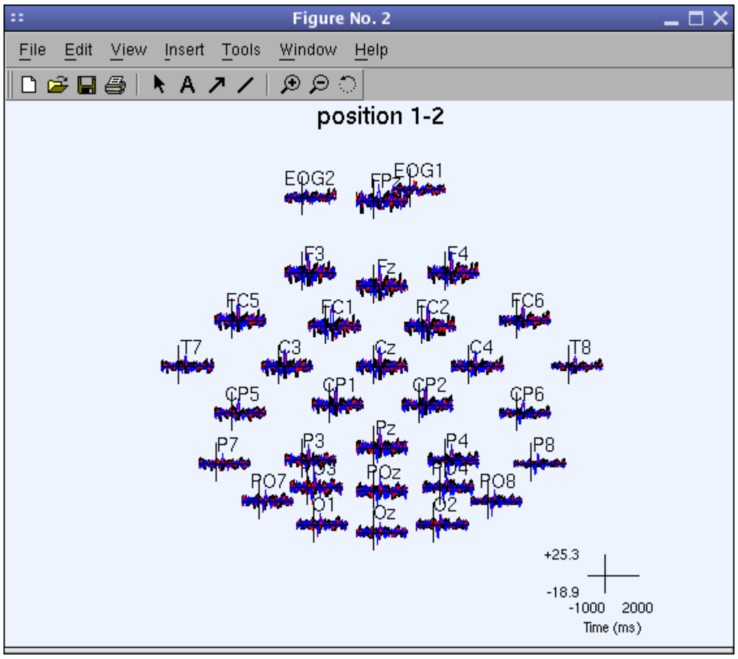
Highligh significant regions (.01 -> p=.01)

Use RMS instead of average (check):

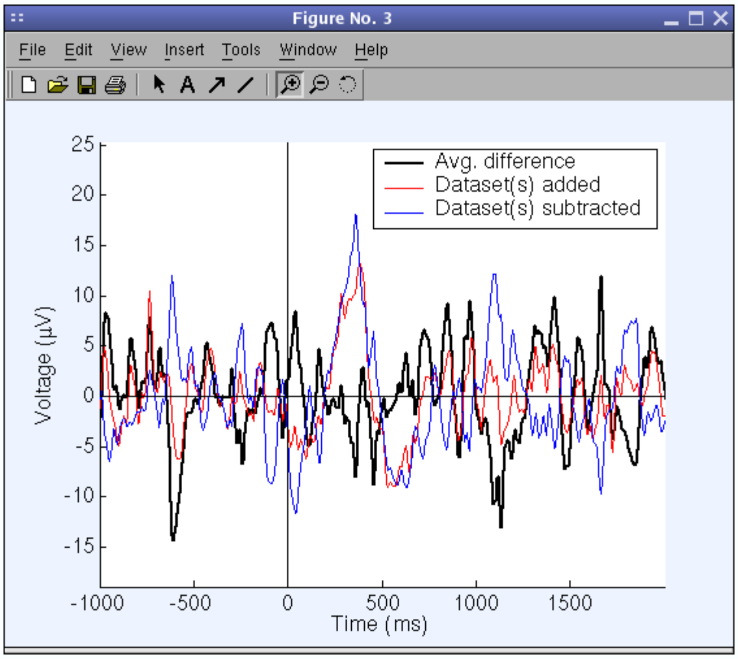
Low pass (Hz) (for display only) 30

Plottopo options ('key', 'val'): 'title', 'position 1-2'

The **plottopo()** figure (below) appears.



Again, individual electrode traces can be plotted in separate windows by clicking on electrode locations of interest in the figure (above). Note that here, since the two conditions are similar (only the position of the stimulus on the screen changes), the ERP difference is close to 0.



This function can also compute and plot grand−mean ERP differences between conditions across several subjects, and can assess significant differences between two conditions using a paired t−test (two−tailed). To do so, load the datasets for the two conditions for each subject into EEGLAB and enter the appropriate indices in the **pop_comperp()** window (Note: future versions of EEGLAB will allow processing datasets directly from disk, without loading them, to minimize memory use).

Before pursuing, **go back to the second dataset** using the **Datasets** top menu. You may also delete datasets number 3 and 4 by selecting **File > Delete dataset(s)**. In the next tutorial chapter, we show how to use EEGLAB to make 3−D ERP−image plots of collections of single−trial data.

I.8. Plotting ERP images

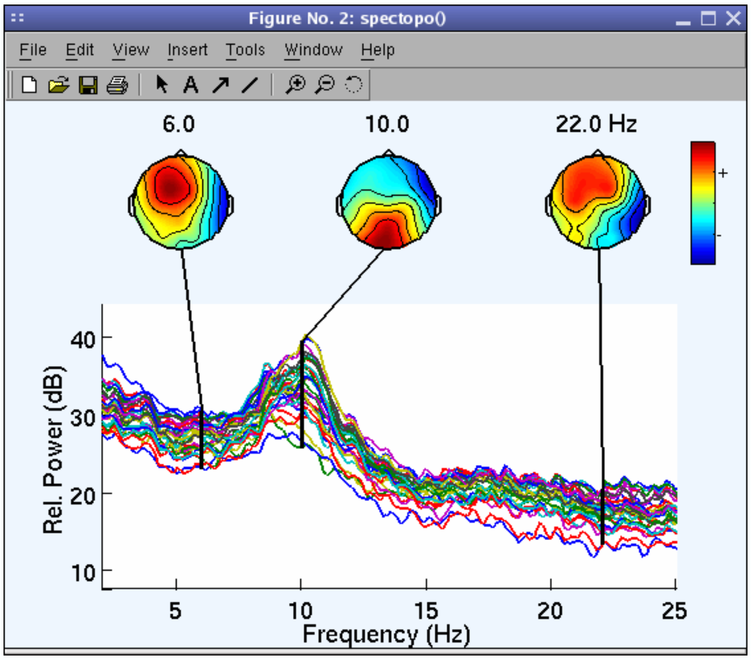
The field of electrophysiological data analysis has been dominated by analysis of 1−dimensional event−related potential (ERP) averages. The ERP image is a related, but more general 2−D (values at times−by−epochs) view of the data. '**ERP images**' are 2−D image transforms of epoched data, in which data epochs are first sorted along ***some*** relevant dimension (for example, subject reaction time or alpha−phase at stimulus onset, etc.), then (optionally) smoothed (cross adjacent trials) and finally color−coded and imaged.

As opposed to the ERP, which as usually defined exists in only one form (the time−locked average). The number of possible ERP images of a set of single trials is quite large −− the trial data can be sorted and imaged along any path through the space of trials. However, not all sorting orders give equal insights into the brain dynamics expressed in the data. It is up to the user to decide which ERP images to study (by default, trials are sorted in the order of appearance in the experiment).

It is also easy to misinterpret or over−interpret an ERP image. For example, using phase−sorting at one frequency (demonstrated below) can blind the user to the presence of other oscillatory phenomena in the same data at different frequencies. Again, the responsibility is the user's to correctly weight and interpret the evidence that a 2−D ERP image gives relative to the hypothesis of interest −− just as it is the user's responsibility to correctly interpret 1−D ERP time series.

I.8.1. Selecting a channel to plot

To plot an ERP image of activity at one data channel in the single trials of our dataset, we must first choose a channel to plot. Let us, for example, choose a channel with high alpha band power (near 10 Hz). Previously in the tutorial we obtained the **spectopo()** plot reproduced below.



The plot above shows that alpha band power (e.g., at 10 Hz) is concentrated over the central occipital scalp. To find which electrodes are located in this region, we can simply plot the electrode names and locations by selecting **Plot > Channel locations > By name**, producing the figure below.

In the figure below, we see that electrode POz is the channel at which alpha power is largest. Click on the **POz** channel label (below) to display its number (27). **Note:** It is also possible to plot electrode locations in the spectral graph by entering " **'electrodes', 'on'** " in the lowest text box ("**Scalp map options**") of the interactive pop_spectopo() window.

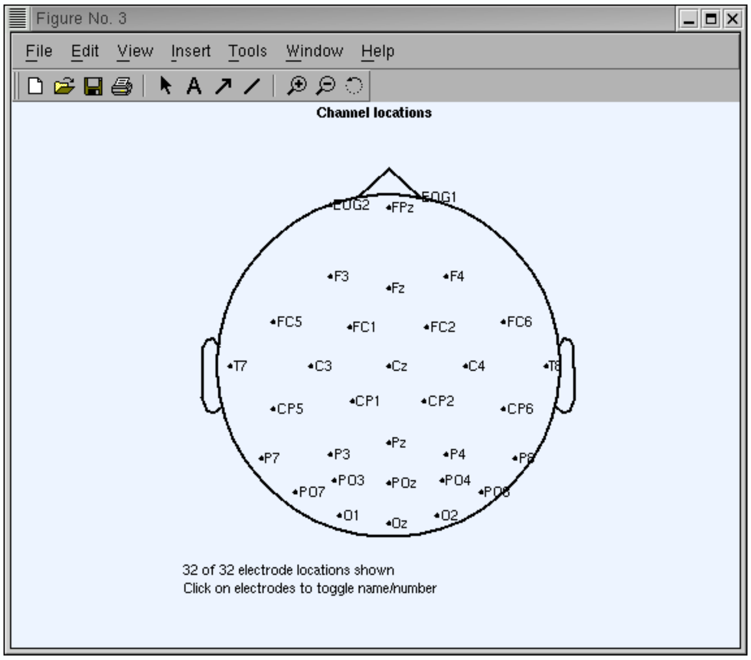
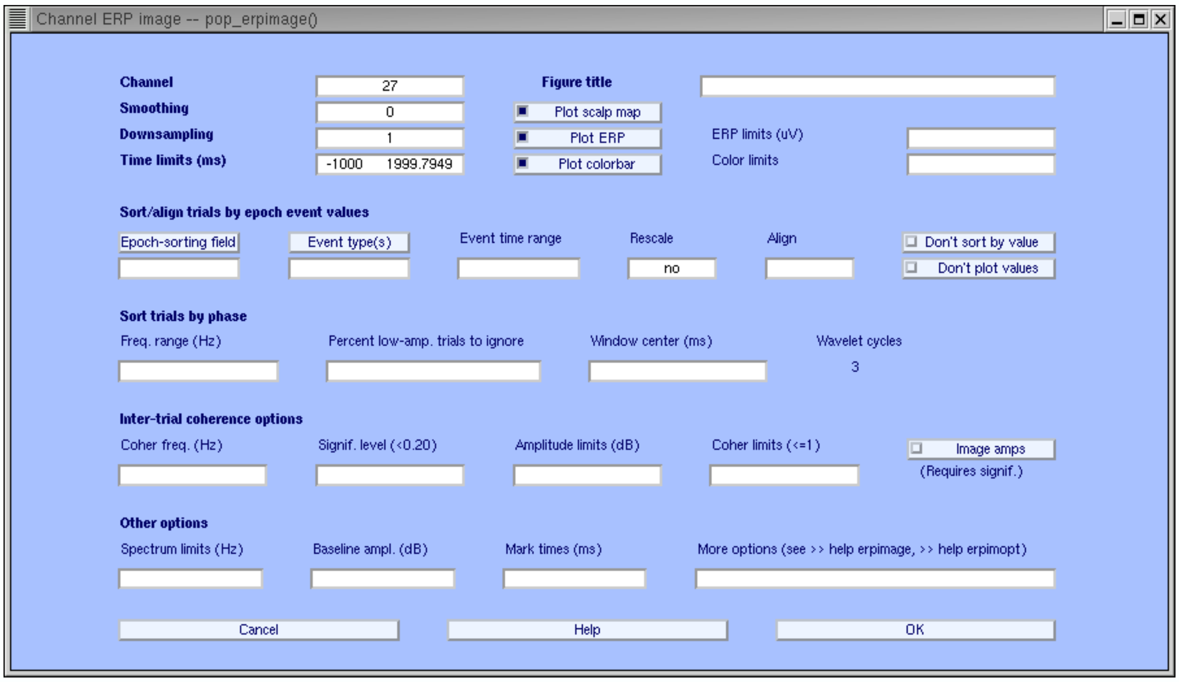


Figure No. 3
File Edit View Insert Tools Window Help
Channel locations
EOG2
EOG1
FPz
F3
Fz
F4
FC5
FC1
FC2
FC6
T7
C3
Cz
C4
T8
CP5
CP1
CP2
CP6
Pz
P7
P3
P4
P8
PO7
PO3
POz
PO4
PO8
O1
Oz
O2
32 of 32 electrode locations shown
Click on electrodes to toggle name/number

I.8.2. Plotting ERP images using pop_erpimage()

Now that we know the number of the channel whose activity we want to study, we can image its activity in single trials in the form of an ERP−image plot. Select **Plot > Channel ERP image** . This brings up the **pop_erpimage()** window (below). Enter the channel number (27), a trial−smoothing value of "**1**", and press **OK**.


Channel ERP image -- pop_erpimage()

Channel 27
Smoothing 0
Downsampling 1
Time limits (ms) -1000 1999.7949

Figure title
■ Plot scalp map
■ Plot ERP
■ Plot colorbar
ERP limits (uV)
Color limits

Sort/align trials by epoch event values
Epoch-sorting field | Event type(s) | Event time range | Rescale: no | Align
☐ Don't sort by value
☐ Don't plot values

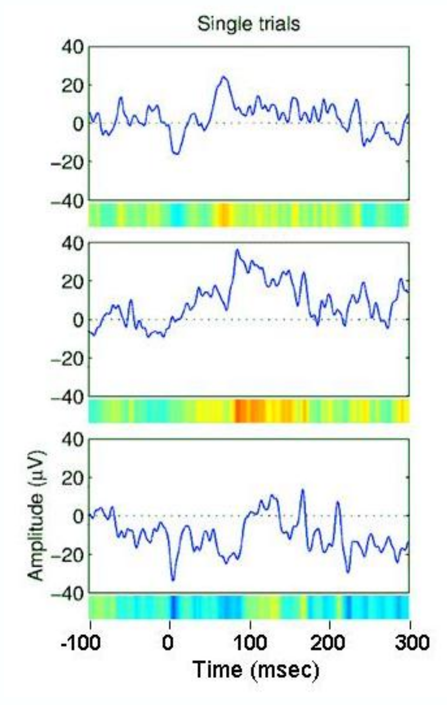
Sort trials by phase
Freq. range (Hz) | Percent low-amp. trials to ignore | Window center (ms) | Wavelet cycles: 3

Inter-trial coherence options
Coher freq. (Hz) | Signif. level (<0.20) | Amplitude limits (dB) | Coher limits (<=1)
☐ Image amps (Requires signif.)

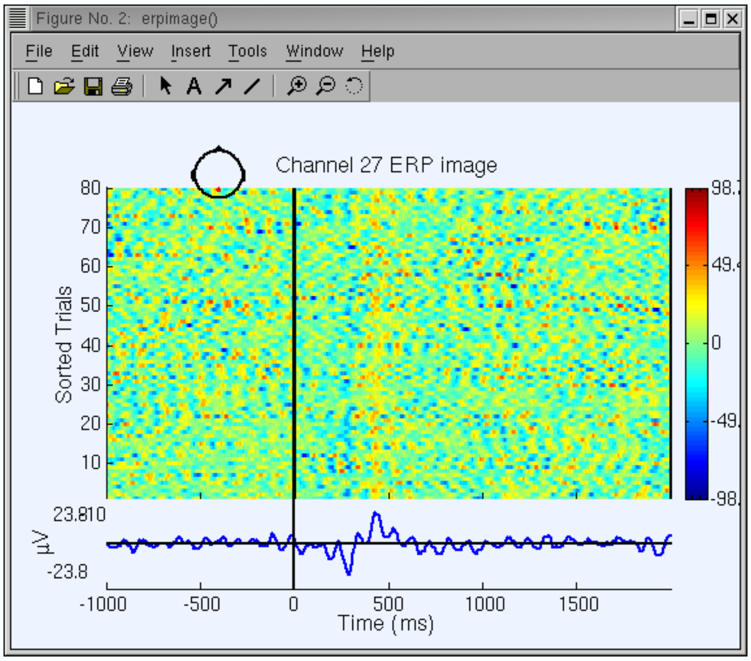
Other options
Spectrum limits (Hz) | Baseline ampl. (dB) | Mark times (ms) | More options (see >> help erpimage, >> help erpimopt)

Cancel | Help | OK

An ERP image is a rectangular colored image in which every horizontal line represents activity occurring in a single experimental trial (or a vertical moving average of adjacent single trials). The figure below (not an ERP image) explains the process of constructing ERP−image plots. Instead of plotting activity in single trials such as left−to−right traces in which potential is encoded by the height of the trace, we color−code their values in left−to−right straight lines, the changing color value indicating the potential value at each time point in the trial. For example, in the following image, three different single−trial epochs (blue traces) would be coded as three different colored lines (below).

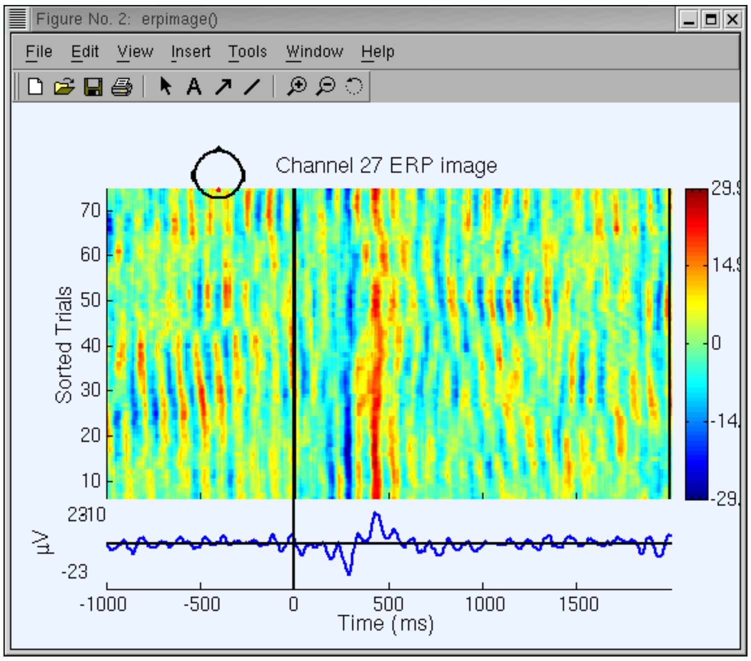


By stacking above each other the color−sequence lines for all trials in a dataset, we produce an ERP image. In the standard **erpimage()** output figure (below), the trace below the ERP image shows the average of the single−trial activity, i.e. the ERP average of the imaged data epochs. The head plot (top left) containing a red dot indicates the position of the selected channel in the montage. (Note: Both of these plotting features (as well as several others) can be turned off in the **pop_erpimage()** pop−up window (above). See checkboxes "**plot ERP**" and "**plot scalp map**").



Since activity in single trials contains many variations, it may be useful to smooth the activity (vertically) over trials using a (boxcar) moving average. Again call up the **pop_erpimage()**

interactive window and set the smoothing width to **10** instead of **1** (note that parameters from the last call are remembered. If you experience a problem with this feature, type **>> h(0)** on Matlab command line to clear the history). Now (see below) it is easier to see the dominant alpha−band oscillations in single trials.

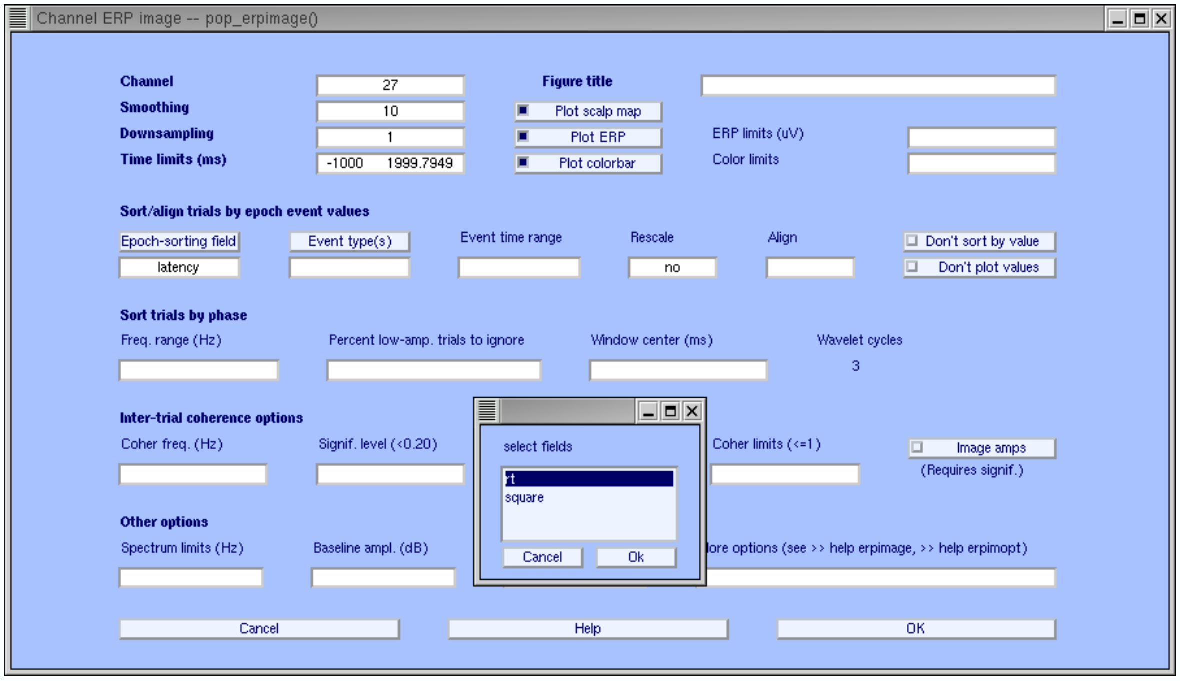


When plotting a large number of trials, it is not necessary to plot each (smoothed) trial as a horizontal line. (The screen and/or printer resolution may be insufficient to display them all). To reduce the imaging delay (and to decrease the saved plot file size), one can decimate some of the (smoothed) ERP−image lines. Entering **4** in the "**Downsampling**" box of the **pop_erpimage()** window would decimate (reduce) the number of lines in the ERP image by a factor of **4**. If the **Smoothing** width is greater than **2*4 = 8** , no information will be lost from the (smoothed) image. **Note:** To image our sample dataset, it is not necessary to decimate, since we have relatively few (80) trials.

I.8.3. Sorting trials in ERP images

In the ERP−image figures above, trials were imaged in (bottom−to−top) order of their occurrence during the experiment. It is also possible to sort them in order of any other variable that is coded as an event field belonging to each trial in the dataset. Below, we demonstrate sorting the same trials in order of response time event latency (reaction time).

In the **pop_erpimage()** window again, first press the button "**Sorting field**", and select "**Latency**". Next, press the button "**Event type**", and select **rt** as shown below. In the ensuing ERP image, trials will be sorted by the **latency** of **rt** events (our sample data have one **rt** event per epoch (if it had not been the case, **erpimage()** would only have plotted epochs with rt events). Enter **Plotting time limits** of **[−200 800]** ms to plot activity immediately following stimulus onsets.



Channel ERP image -- pop_erpimage()

Channel: 27
Smoothing: 10
Downsampling: 1
Time limits (ms): -1000 1999.7949
Figure title:
■ Plot scalp map
■ Plot ERP
■ Plot colorbar
ERP limits (uV):
Color limits:

Sort/align trials by epoch event values

Epoch-sorting field: latency
Event type(s):
Event time range:
Rescale: no
Align:
☐ Don't sort by value
☐ Don't plot values

Sort trials by phase

Freq. range (Hz):
Percent low-amp. trials to ignore:
Window center (ms):
Wavelet cycles: 3

Inter-trial coherence options

Coher freq. (Hz):
Signif. level (<0.20):
Coher limits (<=1):
☐ Image amps (Requires signif.)

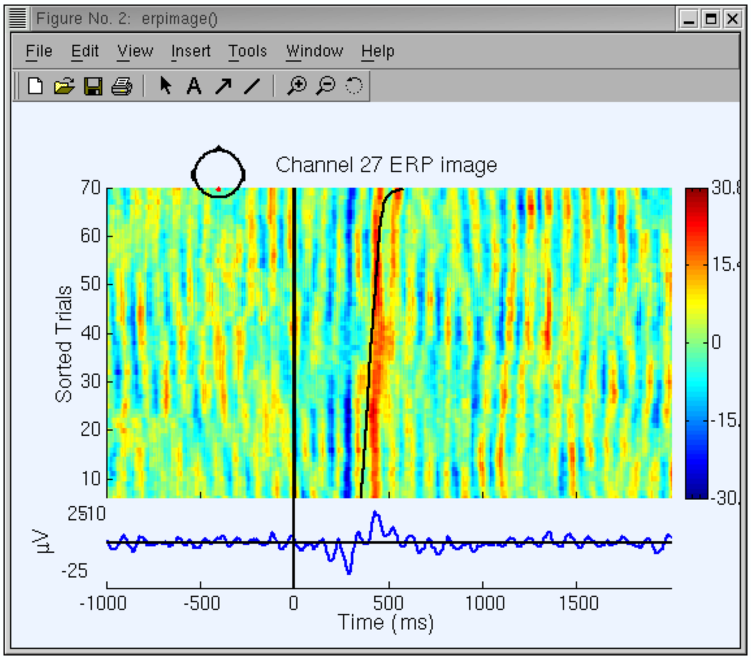
Other options

Spectrum limits (Hz):
Baseline ampl. (dB):
More options (see >> help erpimage, >> help erpimopt)

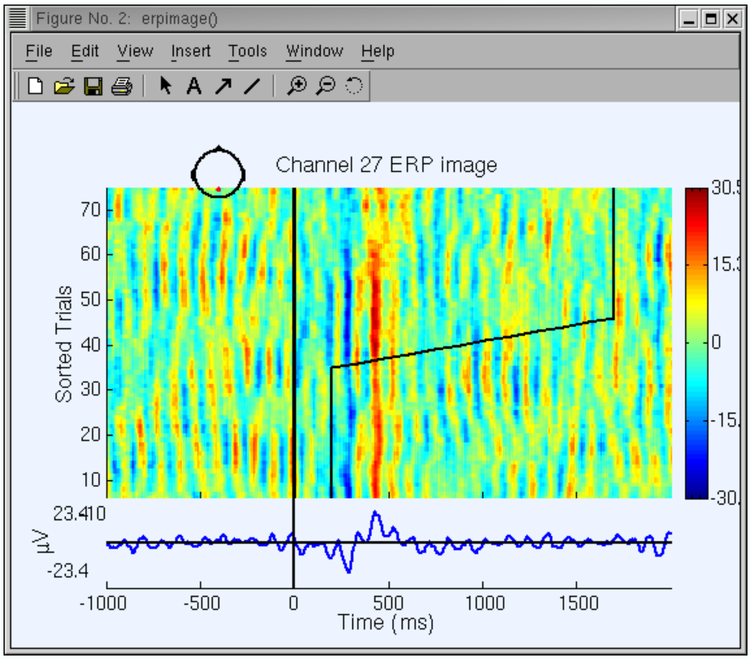
Cancel | Help | OK

select fields
rt
square
Cancel | Ok

Note: In this and some other interactive pop−windows, holding the mouse cursor over the label above a text−entry box for a few seconds pops up an explanatory comment. Now, the erpimage() figure below appears.

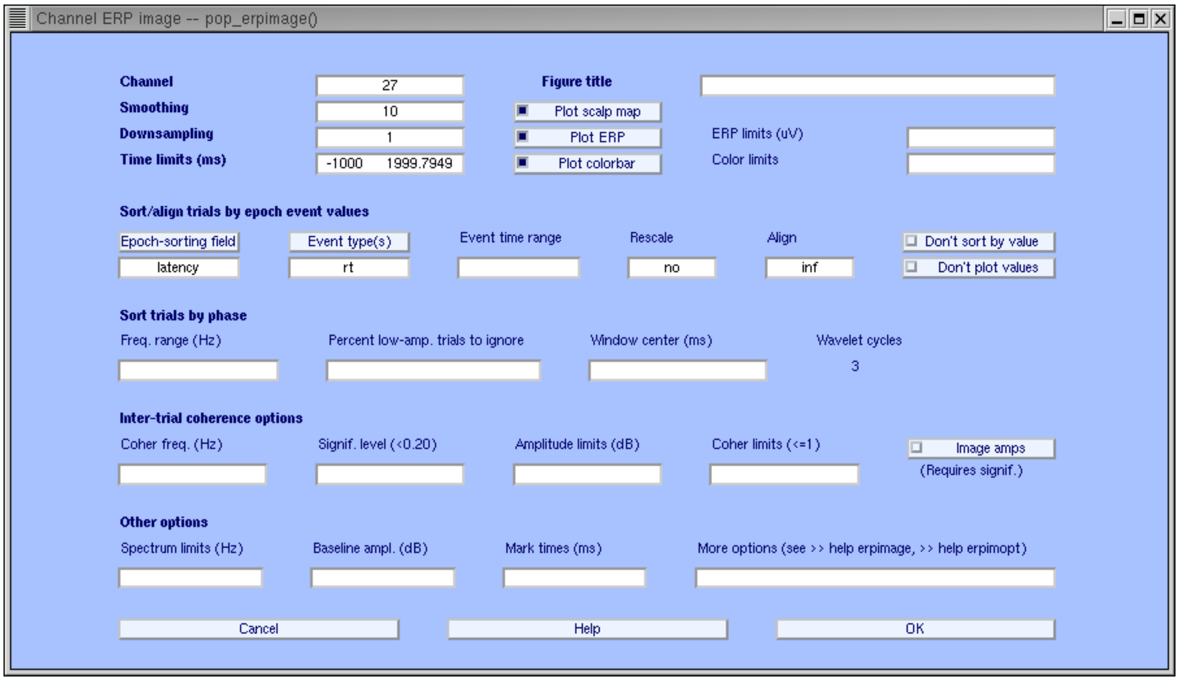


In general, the user can sort on any event field value. For example call back the pop_erpimage() window, press the "**Sorting Field**" button, and select **position** instead of **latency**. Remove **rt** from the **Event type** box. Finally enter **yes** under the **Rescale** box. Press **OK**. In the resulting erpimage() plot, trials are sorted by stimulus position (1 or 2, automatically normalized values to fit the post−stimulus space for display). Note that the smoothing width (10) is applied to both the single−trial data and to the sorting variable. This explains the oblique line connecting the low (1) and high (2) sorting variable regions. **Note:** One can also enter a Matlab expression to normalize the sorting variable explicitly (see **erpimage()** help).



Now, reselect the **latency** of the **rt** events as the trial−sorting variable (press the **Sorting field** button to select "**latency**" and press the **Event type** button to select "**rt**"). Enter "**no"** under **Rescale** (else, reaction times would be automatically normalized). The **Time window** input box allows you to select a subset of events in a given time range. Here, input "**0 350**" to select event latencies (**rt**, in our case) between 0 and 350 ms only.

Use the **Align** input to re−align the single−trial data on sorting variable and the change time limits. The latency value given in **Align** will be used as time 0. To select the median of the trial−sorting values (here, median reaction time) as the new time 0, input "**Inf**", the Matlab symbol for infinity in this box (as below). **Note:** Temporal realignment of data epochs, relative to one another, will result in missing data at the lower−left and upper−right "corners" of the ERP image. The ERP−image function shows these as green (0) and returns these values as "**NaN**"s (Matlab not−a−number).



Channel ERP image -- pop_erpimage()

Channel	27	Figure title	
Smoothing	10	■ Plot scalp map	
Downsampling	1	■ Plot ERP	ERP limits (uV)
Time limits (ms)	-1000 1999.7949	■ Plot colorbar	Color limits

Sort/align trials by epoch event values

Epoch-sorting field	Event type(s)	Event time range	Rescale	Align	☐ Don't sort by value
latency	rt		no	inf	☐ Don't plot values

Sort trials by phase

Freq. range (Hz)	Percent low-amp. trials to ignore	Window center (ms)	Wavelet cycles
			3

Inter-trial coherence options

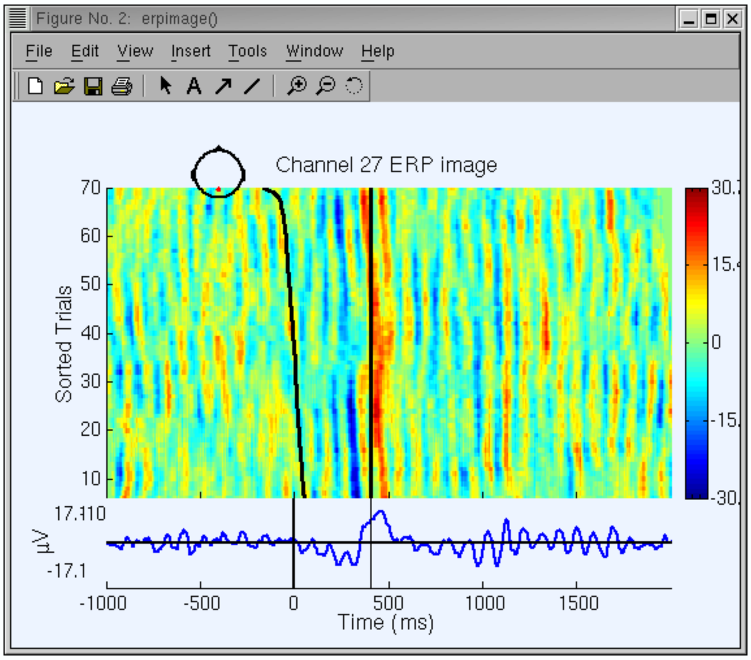
Coher freq. (Hz)	Signif. level (<0.20)	Amplitude limits (dB)	Coher limits (<=1)	☐ Image amps (Requires signif.)

Other options

Spectrum limits (Hz)	Baseline ampl. (dB)	Mark times (ms)	More options (see >> help erpimage, >> help erpimopt)

Cancel Help OK

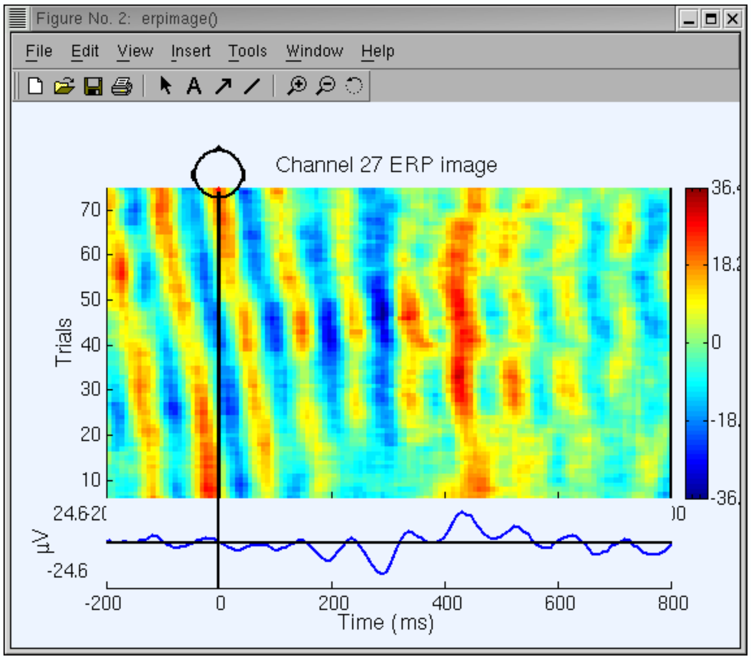
The ERP image figure (below) will be created. Here, the straight vertical line at time 0 indicates the moment of the subject response, and the curving vertical line, the time at which the stimulus was presented in each trial. Compare this figure with the previous non−aligned, RT−sorted ERP image (next−to−last figure above).



I.8.4. Plotting ERP images with spectral options

Next, we will experiment with sorting trials by their EEG phase value in a specified time/frequency window. Though **rt** values can be shown in phase−sorted ERP−image figures, we will omit them for simplicity.

To do this, return to the pop_erpimage() window from the menu. Clear the contents of the **Sorting Field**, **Event type** and **Align** inputs. Then, in the **Sort trials by phase** section, enter **10** (Hz) under **Frequency** and **0** (ms) under **Center time**. Enter **−200 800** next to "**Time limits (ms)**" to "zoom in" on the period near stimulus onset. We then obtain the ERP−image figure below.

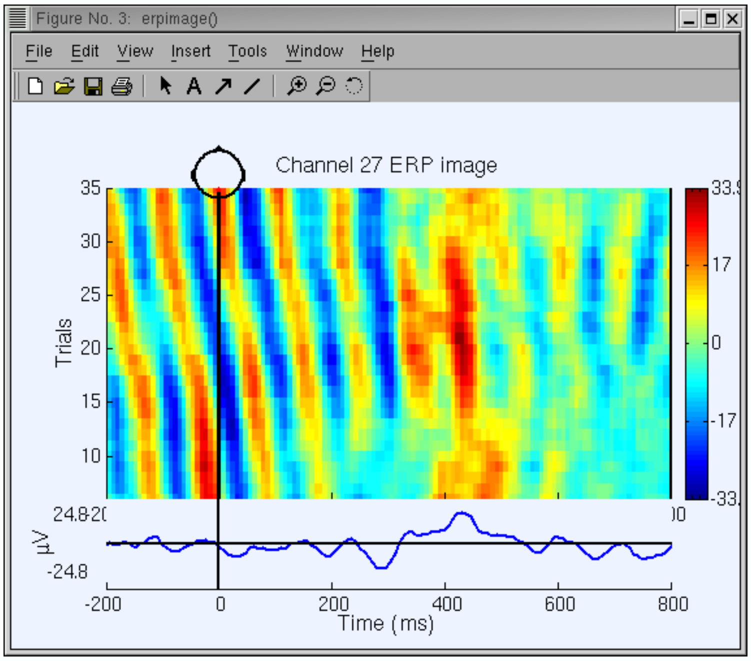


Note just before the stimulus onset the red oblique stripe: this is due to phase sorting: the phase (i.e. the latency of the wave peak) varies continuously across resorted trials.

In this computation, a 3−cycle 10 Hz wavelet was applied to a window in each trial centered at time 0. The width of the wavelet was 300 ms (i.e., three 10−Hz cycles of 100 ms). Therefore, it extended from −150 ms to 150 ms. After the wavelet was applied to each trial, the function sorted the trials in order of the phase values (−pi to pi) and displayed an ERP image of the trials in this (bottom−to−top) order. The dominance of circa 10−Hz activity in the trials, together with the 10−trial smoothing we applied makes the phase coherence between adjacent trials obvious in this view.

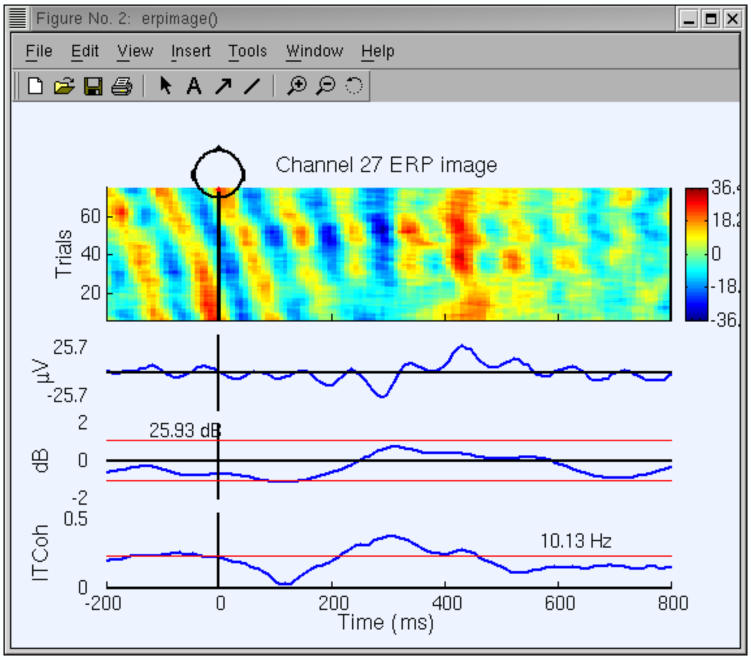
Note that we could have applied phase−sorting of trials using any time/frequency window. The results would depend on the strength of the selected frequency in the data, particularly on its degree of momentum (i.e., did the data exhibit long bursts at this frequency), and its phase−locking (or not) to experimental events. Phase−sorted ERP images using different time and frequency windows represent different paths to "fly through" complex (single−channel) EEG data.

To see the phase sorting more clearly, keep the same settings, but this time enter "**50**" under "**% of trials to ignore**". Here, the 50% of trials with smallest 10−Hz (alpha) power in the selected time window will be rejected; only the (40) others (larger−alpha 50%) will be imaged. Here (below), we can better see how the alpha wave seems to resynchronize following the stimulus. Before time 0, alpha phase is more or less random (uniformly distributed) and there is little activity in the average ERP. At about 200 ms, alpha activity seems to (partially) synchronize with the stimulus and an N300 and P400 ERP appears.



Our interpretation (above) of these trials as representing phase synchronization need not be based on visual impression alone. To statistically assess whether alpha activity is partially resynchronized by (i.e., is partly phase−reset by) the stimuli, we need to plot the phase coherence (or phase−locking factor) between the stimulus sequence and the post−stimulus data. This measure, the Inter−Trial Coherence (ITC) in our terminology, takes values between 0 and 1. A value of 1 for the time frequency window of interest indicates that alpha phase (in this latency window) is constant in every trial. A value of 0 occurs when the phase values in all trials are uniformly distributed around the unit circle. In practice, values somewhat larger than 0 are expected for any finite number of randomly phase−distributed trials.

To plot the ITC in our ERP−image figure, we choose to enter the following parameters in the **pop_erpimage()** window: we omit the "**% of Trials to ignore**" value and enter "**9 11**" under phase sorting and "**9 11**" under "**Coher freq.**". Note that these two entries must be equal (the window actually prevents the user from entering different values). Entering a frequency range instead of one frequency (e.g., **10** as before) tells **erpimage()** to find the data frequency with maximum power in the input data (here between 9 and 11 Hz). Finally enter "**0.01**" under "**Signif. level**" and press **OK**.



Two additional plot panels appear below the ERP panel. The middle panel shows mean changes in power across the epochs in dB. Red lines indicate 1% confidence limits according to surrogate data drawn from random windows in the baseline. Here, power at the selected frequency (10.13 Hz) shows no significant variations across the epoch. The number "**25.93 dB**" in the baseline of this panel indicates the absolute baseline power level. **Note:** To compare results, it is sometimes useful to set this value manually in the main ERP−image pop−window.

The bottom plot panel shows the event−related Inter−Trial Coherence (ITC), which indexes the degree of phase synchronization of trials relative to stimulus presentation. The value "**10.13 Hz**" here indicates the analysis frequency selected. Phase synchronization becomes stronger than our specified p=0.01 significance cutoff at about 300 ms. **Note:** The ITC significance level is typically lower when based on more trials. Moreover, ITC is usually not related to power changes.

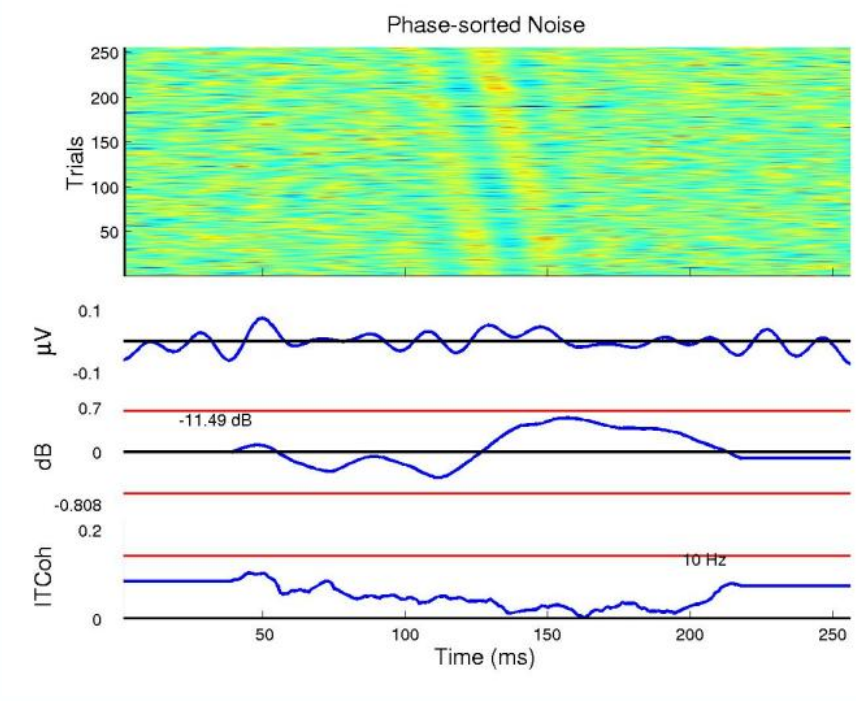
Does the ERP here arise through partial phase synchronization or reset following stimulus onset? **In a 'pure' case of (partial) phase synchronization:**

- EEG power (at the relevant frequencies) remains constant in the post−stimulus interval.
- The ITC value is significant during the ERP, but less than 1 (complete phase locking).

In our case, the figure (above) shows a significant post−stimulus increase in alpha ITC accompanied by a small (though non−significant) increase in alpha power. In general, an ERP could arise from partial phase synchronization of ongoing activity combined with a stimulus−related increase (or decrease) in EEG power.

It is important not to overinterpret the results of phase sorting in ERP image plots. For example, the following calls from the Matlab commandline simulate 256 1−s data epochs using Gaussian white noise, lowpass filters this below (simulated) 12 Hz, and draw the following 10−Hz phase sorted ERP image plot of the resulting data. The figure appears to identify temporally coherent 10−Hz activity in the (actual) noise. The (middle) amplitude panel below the ERP image shows, however, that amplitude at (simulated) 10 Hz does not change significantly through the (simulated) epochs, and the lowest panel shows that inter−trial coherence is also nowhere significant (as confirmed visually by the straight diagonal 10−Hz wave fronts in the center of the ERP image).

```
% Simulate 256 1−s epochs with Gaussian noise
% at 256−Hz sampling rate; lowpass < 12 Hz
>> data = eegfilt(randn(1,256*256),256,0,15);

% Plot ERP image, phase sorted at 10 Hz
>> erpimage(data,zeros(1,256),1:256,'Phase−sorted Noise',1,1,...
        'phasesort',[128 0 10],'srate',256,...
        'coher',[10 10 .01], 'erp','caxis',0.9);
```



Taking epochs of white noise (as above) and adding a strictly time−locked 'ERP−like' transient to each trial will give a phase−sorted ERP−image plot showing a sigmoidal, not a straight diagonal wavefront signature. How can we differentiate between the two interpretations of the same data (random EEG plus ERP versus partially phase reset EEG)? For simulated one−channel data, there is no way to do so, since **both** are equally valid ways of looking at the same (simulated) data − no matter how it was created. After all, the simulated data themselves do not retain any "impression" of how they were created − even if such an impression remains in the mind of the experimenter!

For real data, we must use convergent evidence to bias our interpretation towards one or the other (or both) interpretations. The "partial phase resetting" model begins with the concept that the physical sources of the EEG (partial synchronized local fields) may ALSO be the sources of or contributors to average−ERP features. This supposition may be strengthened or weakened by examination of the spatial scalp distributions of the ERP features and of the EEG activity. However, here again, a simple test may not suffice since many cortical sources are likely to contribute to both EEG and averaged ERPs recorded at a single electrode (pair). An ERP feature may result from partial phase resetting of only **one** of the EEG sources, or it may have many contributions including truly 'ERP−like' excursions with fixed latency and polarity across trials, monopolar 'ERP−like' excursions whose latency varies across trials, and/or partial phase resetting of **many** EEG processes. Detailed spatiotemporal modeling of the collection of single−trial data is required to parcel out these possibilities. For further discussion of the question in the context of an actual data set, see Makeig et al. (2002). In that paper, phase resetting at alpha and theta frequencies was indicated to be the predominant cause of the recorded ERP (at least at the indicated scalp site, POz). How does the ERP in the figure above differ?

The Makeig et al. paper dealt with non−target stimuli, whereas for the sample EEGLAB dataset we used epochs time locked to target stimuli from one subject (same experiment). The phase synchronization might be different for the two types of stimuli. Also, the analysis in the paper was carried out over 15 subjects and thousands of trials, whereas here we analyze only 80 trials from one subject. (The sample data we show here are used for tutorial purposes. We are now preparing a full report on the target responses in these experiments.)

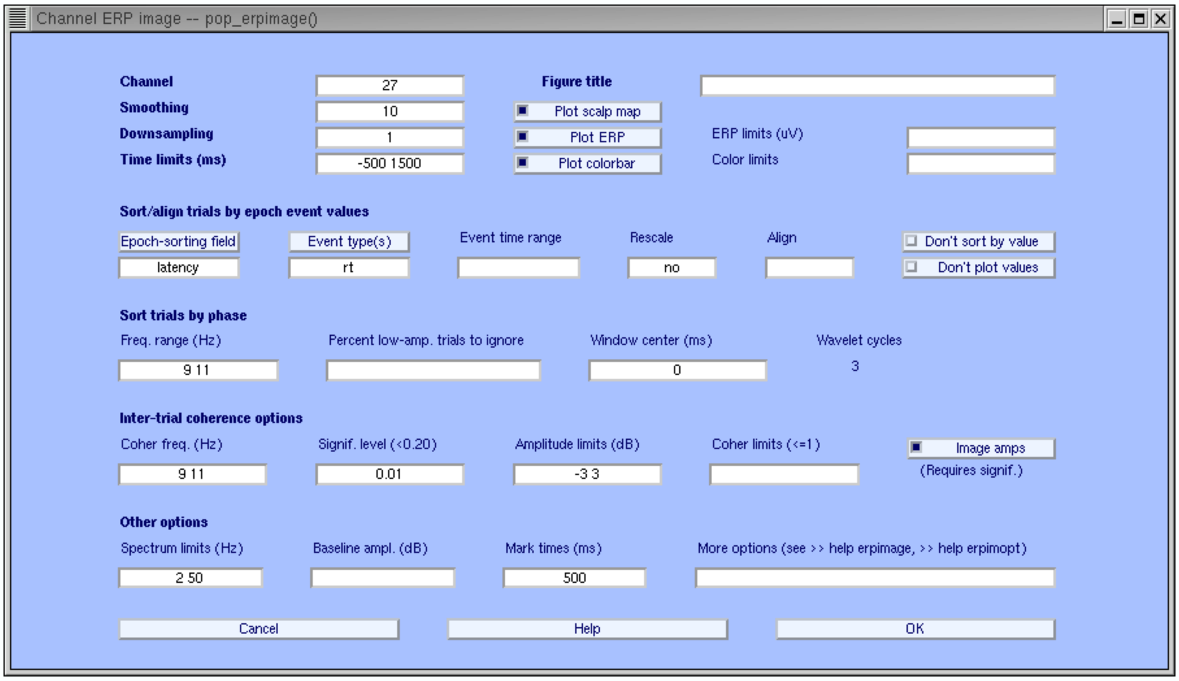
Note: Altogether, there are five trial sorting methods available in **erpimage()**:

- Sort by the sorting variable (default) − Sorts input data trials (epochs) by the '**sortvar**,' sorting variable (for example, RT) input for each epoch of the input data.
- Sort by value ('**valsort'**)− Here, trials are sorted in order of their mean value in a given time window. Use this option to sort by ERP size (option not available yet in the interactive window).
- Sort by amplitude (**'ampsort'**)−− Trials are sorted in order of spectral amplitude or power at a specified frequency and time window. Use this option to display, for example, P300 responses sorted by alpha amplitude (option not available yet in the interactive window).
- Sort by phase (**'phasesort'**)−− Trials are sorted in order of spectral phase in a specified time/frequency window.
- Do not sort (**'nosort'**)−− Display input trials in the same order they are input.

I.8.5. Plotting spectral amplitude in single trials and additional options

There are several other **erpimage()** options that we will briefly illustrate in the following example. The "**plotamp**" entry on the **pop_erpimage()** window allows us to image amplitude of the signal (at the frequency of interest) in the single trials, instead of the raw signals themselves. Check this box. The "**Spectrum plot limits**" entry adds a small power spectrum plot to the top right of the figure. Enter "**2 50**" to specify the frequency limits for this graph.

Change the "**Sorting field**" box back to "**latency**" and "**Event type** " back to "**rt**". Then enter "**500**" under "**Mark times**" to plot a vertical mark at 500 ms (here for illustrative purpose only). Finally enter "**−500 1500** " under "**Time limits**" to zoom in on a specific time window, and "**−3 3**" under " **Amplitude limits (dB)**".



Channel ERP image -- pop_erpimage()

Channel	27	**Figure title**	
Smoothing	10	■ Plot scalp map	
Downsampling	1	■ Plot ERP	ERP limits (uV)
Time limits (ms)	-500 1500	■ Plot colorbar	Color limits

Sort/align trials by epoch event values

Epoch-sorting field	Event type(s)	Event time range	Rescale	Align	☐ Don't sort by value
latency	rt		no		☐ Don't plot values

Sort trials by phase

Freq. range (Hz)	Percent low-amp. trials to ignore	Window center (ms)	Wavelet cycles
9 11		0	3

Inter-trial coherence options

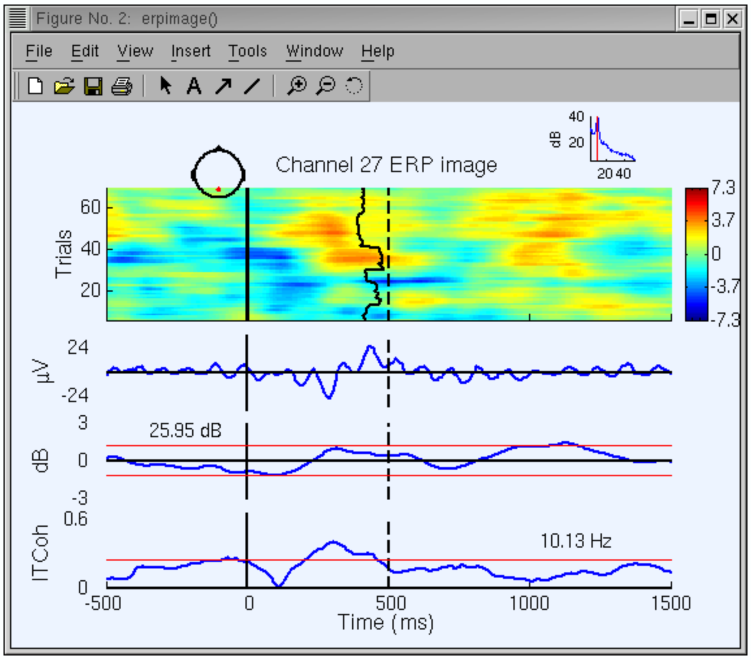
Coher freq. (Hz)	Signif. level (<0.20)	Amplitude limits (dB)	Coher limits (<=1)	■ Image amps
9 11	0.01	-3 3		(Requires signif.)

Other options

Spectrum limits (Hz)	Baseline ampl. (dB)	Mark times (ms)	More options (see >> help erpimage, >> help erpimopt)
2 50		500	

Cancel | Help | OK

The erpimage() figure below appears.

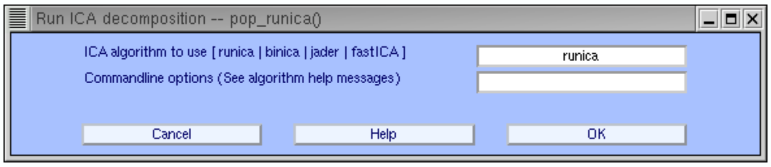


In the next tutorial, we show how to use EEGLAB to perform and evaluate ICA decomposition of EEG datasets.

I.9. Performing Independent Component Analysis of EEG data

I.9.1. Running ICA decompositions

To compute ICA components of a dataset of EEG epochs (or of a continuous EEGLAB dataset), select **Tools > Run ICA**. This calls the function pop_runica(). To test this function, simply press **OK**.



ICA algorithm to use [runica | binica | jader | fastICA] runica
Commandline options (See algorithm help messages)

ICA Algorithms: Note (above) that EEGLAB allows users to try different ICA decomposition algorithms. Only "**runica**", which calls runica() and "**jader**" which calls the function jader() (from Jean−Francois Cardoso) are a part of the default EEGLAB distribution. To use the "**fastica**" algorithm (Hyvarinen et al.), one must install the fastica toolbox and include it in the Matlab path. Details of how these ICA algorithms work can be found in the scientific papers of the teams that developed them.

In general, the physiological significance of any differences in the results or different algorithms (or of different parameter choices in the various algorithms) have not been tested −− neither by us nor, as far as we know, by anyone else. Applied to simulated, relatively low dimensional data sets for which all the assumptions of ICA are exactly fulfilled, all three algorithms return near−equivalent components. We are satisfied that Infomax ICA (runica/binica) gives stable decompositions with up to hundreds of channels (assuming enough training data are given, see below), and therefore we can recommend its use, particularly in its faster binary form (binica()). Note about "**jader**": this algorithm uses 4th−order moments (whereas Infomax uses (implicitly) a combination of higher−order moments) but the storage required for all the 4th−order moments become impractical for datasets with more than ~50 channels. Note about "**fastica**": Using default parameters, this algorithm quickly computes individual components (one by one). However, the order of the components it finds cannot be known in advance, and performing a complete decomposition is not necessarily faster than Infomax. Thus for practical purposes its name for it should not be taken literally. Also, in our experience it may be less stable than Infomax for high−dimensional data sets.

Very important note: We usually run ICA using many more trials that the sample decomposition presented here. As a general rule, finding **N** stable components (from N−channel data) may require **more than 3*N^2** (according to Tony Bell, inventor of ICA Infomax) data sample points (at each channel). In this example with 32 channels, we have 30800 data points and we needed more than 3*32^2=3072 points. Most persons that failed to find reliable component using ICA did not follow this rule.

Supported Systems for binica: To use the optional (and much faster) "**binica**", which calls binica() , the faster C translation of "runica()"), you must make the location of the executable ICA file known to Matlab and executable on your system (**Note:** Edit the EEGLAB icadefs() Matlab script file to specify the location of the **binica()** executable). The EEGLAB toolbox includes three versions of the binary executable Informax ica routine, for **linux** (compiled under Redhat 2.4), **freebsd** (3.0) and **freebsd** (4.0) (these are named, respectively **ica_linux2.4** , **ica_bsd3.0** and **ica_bsd4.0**). Note that the executable file must also be accessible through the Unix user path variable otherwise **binica()** won't work. Windows and sun version (older version of binary ICA executable) are available **here** (copy them to the EEGLAB directory). Please contact us to obtain the latest source code to compile it on your own system.

Running "**runica**" produces the following text on the Matlab command line:

```
Input data size [32,1540] = 32 channels, 1540 frames.
Finding 32 ICA components using logistic ICA.
Initial learning rate will be 0.001, block size 36.
Learning rate will be multiplied by 0.9 whenever angledelta >= 60 deg.
Training will end when wchange < 1e−06 or after 512 steps.
Online bias adjustment will be used.
Removing mean of each channel ...
Final training data range: −145.3 to 309.344
Computing the sphering matrix...
Starting weights are the identity matrix ...
Sphering the data ...
Beginning ICA training ...
step 1 − lrate 0.001000, wchange 1.105647
step 2 − lrate 0.001000, wchange 0.670896
step 3 − lrate 0.001000, wchange 0.385967, angledelta 66.5 deg
step 4 − lrate 0.000900, wchange 0.352572, angledelta 92.0 deg
step 5 − lrate 0.000810, wchange 0.253948, angledelta 95.8 deg
step 6 − lrate 0.000729, wchange 0.239778, angledelta 96.8 deg
...
step 55 − lrate 0.000005, wchange 0.000001, angledelta 65.4 deg
step 56 − lrate 0.000004, wchange 0.000001, angledelta 63.1 deg
Inverting negative activations: 1 −2 −3 4 −5 6 7 8 9 10 −11 −12 −13 −14 −15 −16 17 −18 −19 −20 −21 −22
−23 24 −25 −26 −27 −28 −29 −30 31 −32
Sorting components in descending order of mean projected variance ...
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
```

Note that the "**runica**" Infomax algorithm can only detect supergaussian (i.e. peaky) distribution of activity. If there is strong line noise in data, it is preferable to enter the option " **'extended', 1** " in the command line option box, so the algorithm can also detect subgaussian sources of activity, such as line current and/or slow activity. Another option we often use is the stop option: try " **'stop', 1E−7** " to lower the criterion for stopping learning, thereby lengthening ICA training but possibly returning cleaner decompositions, particularly of high−density array data. We also recommend the use of collections of short epochs that have been carefully pruned of noisy epochs (see **Rejecting artifacts** with EEGLAB).

Important note: the "**runica**" Infomax function returns two matrices, a data sphering matrix (which is used as a linear preprocessing to ICA) and the ICA weight matrix. For more information, refer to ICA help pages (i.e. http://www.sccn.ucsd.edu/~arno/indexica.html). If you wish, the resulting decomposition (i.e., ICA weights and sphere matrices) can then be applied to longer epochs drawn from the same data, e.g. for time−frequency decompositions for which epochs of 3−sec or more may be desirable.

The component order returned by **runica/binica** is in decreasing order of the EEG variance accounted for by each component. In other words, the lower the order of a component, the more data (neural and/or artifactual) it accounts for. In contrast to PCA, for which the first component may account for 50% of the data, the second 25%, etc..., ICA component contributions are much more homogeneous, ranging from roughly 5% down to ~0%. This is because PCA specifically makes each successive component account for **as much as possible** of the remaining activity not accounted for by previously determined components −− while ICA seeks **maximally independent** sources of activity.

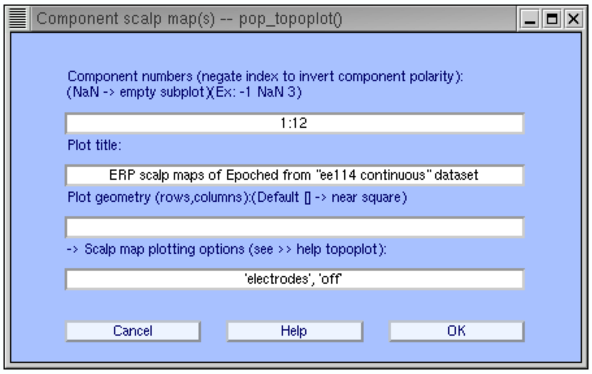
PCA components are temporally or spatially orthogonal − smaller component projections to scalp EEG data typically looking like checker boards − while ICA components of EEG data are maximally temporally independent, but spatially unconstrained −− and therefore able to find maps representing the projection of a partially synchronized domain / island / patch / region of cortex, no matter how much it may overlap the projections of other (relatively independent) EEG sources. This is useful since, apart from ideally (radially) oriented dipoles on the cortical surface (i.e., on cortical gyri, not in sulci), simple biophysics shows that the volume projection of each cortical domain must project appreciably to much of the scalp.

Note: Run twice on the same data, ICA decompositions under **runica/binica** will differ slightly. That is, the ordering, scalp topography and activity time courses of best−matching components may appear slightly different. This is because ICA decomposition starts with a random weight matrix (and randomly shuffles the data order in each training step), so the convergence is slightly different every time. Is this a problem? At the least, features of the decomposition that do not remain stable across decompositions of the same data should not be interpreted except as irresolvable ICA uncertainty.

Differences between decompositions trained on somewhat different data subsets may have several causes. We have not yet performed such repeated decompositions and assessed their common features − though this would seem a sound approach. Instead, in our recent work we have looked for commonalities between components resulting from decompositions from different subjects.

I.9.2. Plotting 2−D Component Scalp Maps

To plot 2−D scalp component maps, select **Plot > Component maps > In 2D**. The interactive window (below) is then produced by function **pop_topoplot()** . It is similar to the window we used for plotting ERP scalp maps. Simply press **OK** to plot all components. **Note:** This may take several figures, depending on number of channels and the **Plot geometry** field parameter. An alternative is to call this functions several times for smaller groups of channels (e.g., **1:30** , **31:60** , etc.). Below we ask for the first 12 components (**1:12**) only.



Component scalp map(s) -- pop_topoplot()

Component numbers (negate index to invert component polarity):
(NaN -> empty subplot)(Ex: -1 NaN 3)

1:12

Plot title:

ERP scalp maps of Epoched from "ee114 continuous" dataset

Plot geometry (rows,columns):(Default [] -> near square)

-> Scalp map plotting options (see >> help topoplot):

'electrodes', 'off'

The following **topoplot()** window appears, showing the scalp map projection of the selected components. Note that the scale in the following plot uses arbitrary units. The scale of the component's activity time course also uses arbitrary units. However, the component's scalpmap values multiplied by the component activity time course is in the same unit as the data.

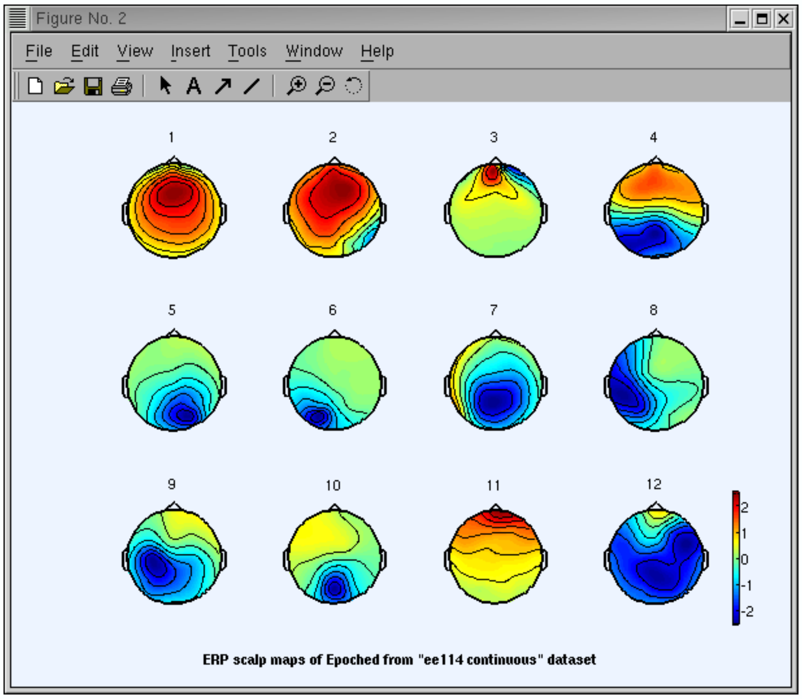
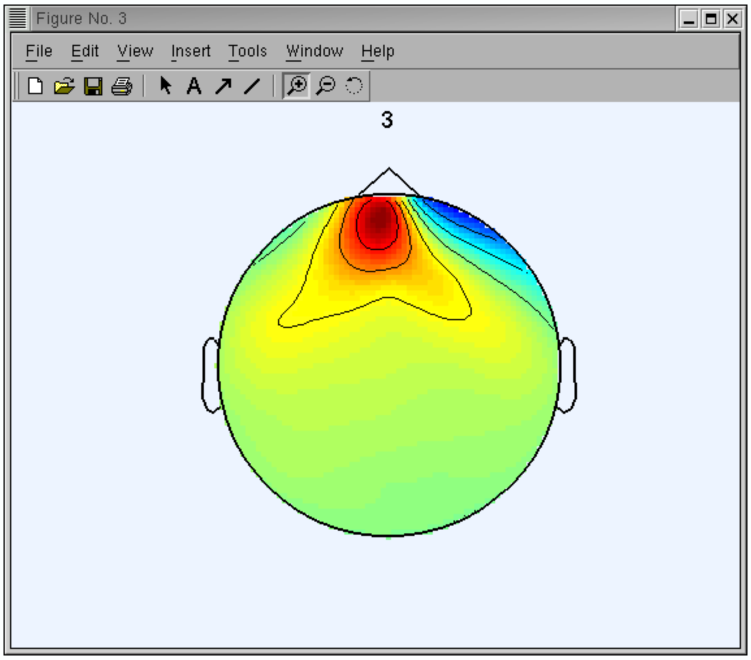


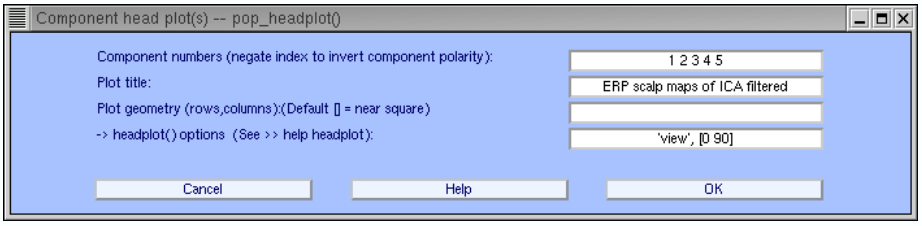
Figure No. 2
File Edit View Insert Tools Window Help
1
2
3
4
5
6
7
8
9
10
11
12
2
1
0
-1
-2
ERP scalp maps of Epoched from "ee114 continuous" dataset

Learning to recognize types of independent components may require experience. The main criteria to determine if a component is 1) cognitively related 2) a muscle artifact or 3) some other type of artifact are, first, the scalp map (as shown above), next the component time course, next the component activity power spectrum and, finally (given a dataset of event−related data epochs), the **erpimage()**. For example an expert eye would spot component 3 (above) as an eye artifact component (see also component activity by calling menu **Plot > Component activations (scroll)**). In the window above, click on scalp map number 3 to pop up a window showing it alone.



I.9.3. Plotting component headplots

Using EEGLAB, you may also plot a 3−D head plot of a component topography by selecting **Plot > Component maps > In 3D**. This calls pop_headplot(). Select one ore more components (below) and press **OK**.


Component head plot(s) -- pop_headplot()

Component numbers (negate index to invert component polarity): 1 2 3 4 5
Plot title: ERP scalp maps of ICA filtered
Plot geometry (rows,columns):(Default [] = near square)
-> headplot() options (See >> help headplot): 'view', [0 90]

The headplot() window below appears. You may use the Matlab rotate−3D option to rotate these headplots with the mouse. Else, enter a different '**view**' angle in the window above.

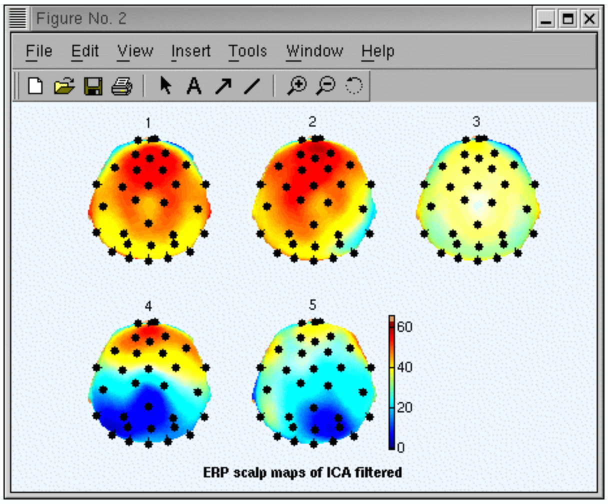
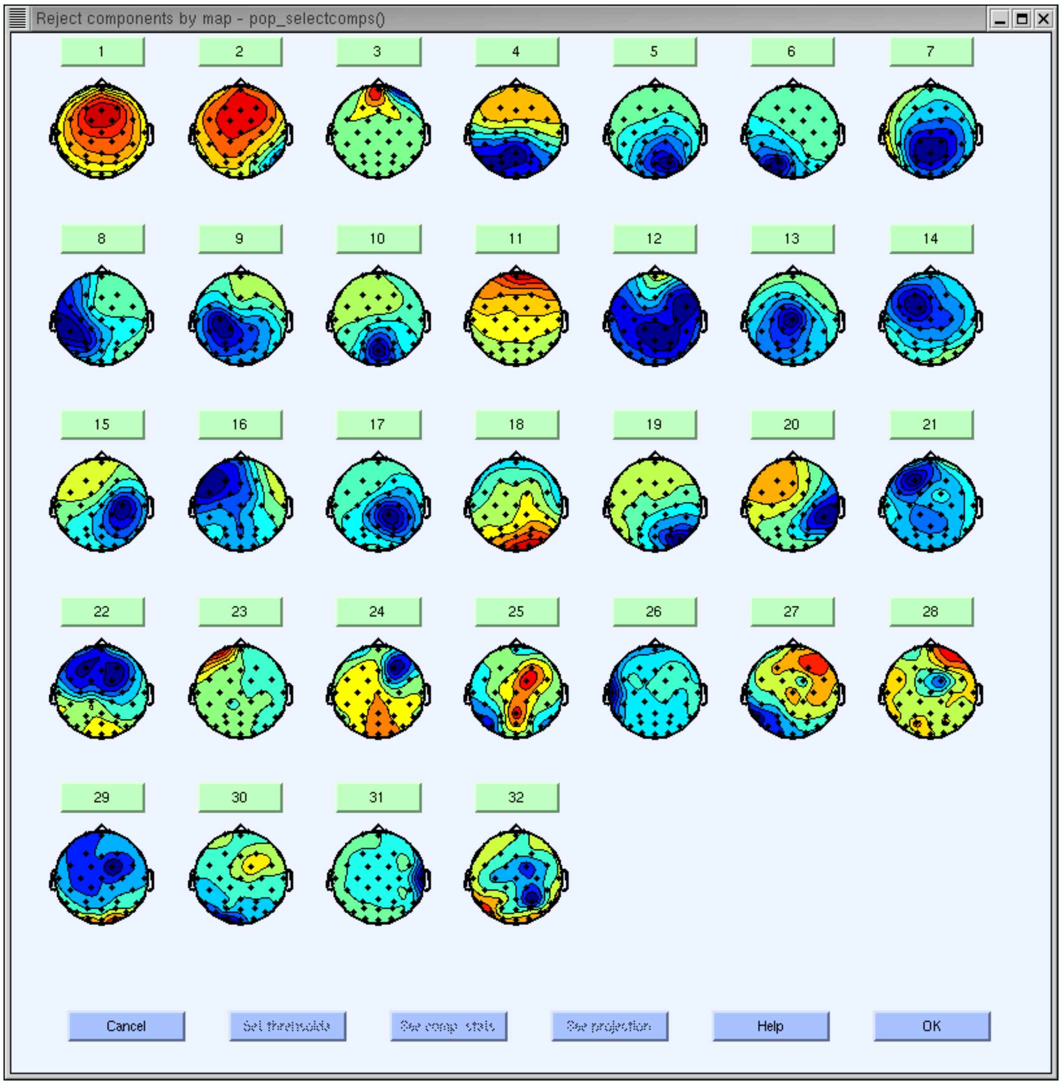


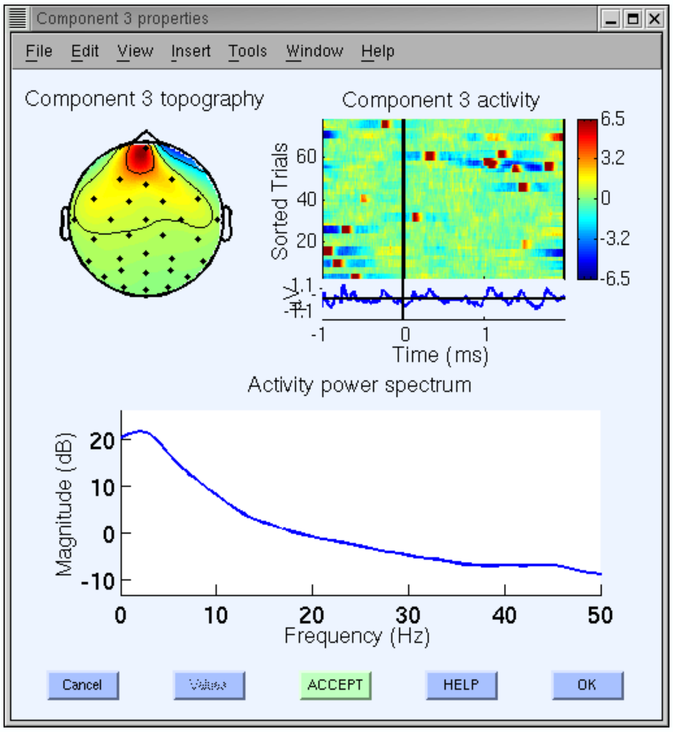
Figure No. 2
File Edit View Insert Tools Window Help
1
2
3
4
5
60
40
20
0
ERP scalp maps of ICA filtered

I.9.4. Studying and removing ICA components

To study component properties and label components for rejection (i.e. to identify components to subtract from the data), select **Tools > Reject using ICA > Reject components by map** . The difference between the resulting figure(s) and the previous 2−D scalp map plots is that one can here plot the properties of each component by clicking on the rectangular button above each component scalp map.

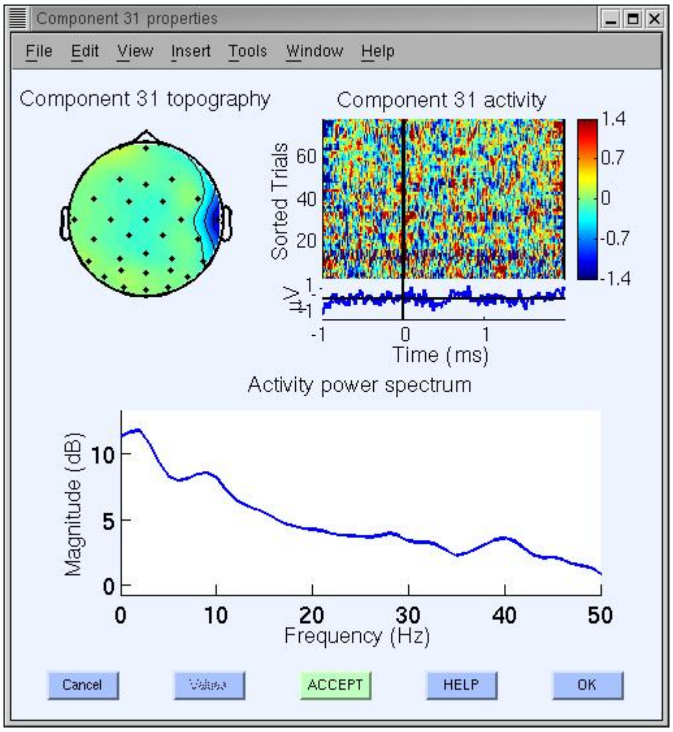


For example, click on the button labeled "**3**". This component can be identified as an eye artifact for three reasons: 1) The smoothly decreasing EEG spectrum (bottom panel) is typical of an eye artifact; 2) The scalp map shows a strong far−frontal projection typical of eye artifacts; And 3) it is possible to see individual eye movements in the component **erpimage()** (top−right panel). Eye artifacts are (nearly) always present in EEG datasets. They are usually in leading positions in the component array (because they tend to be big) and their scalp topographies (if accounting for lateral eye movements) look like component 1 or perhaps (if accounting for eye blinks) like that of component 6 (above). Component property figures can also be accessed directly by selecting **Plot > Component properties**. (There is an equivalent menu item for channels, **Plot > Channel properties**). Artifactual components are also relatively easy to identify by visual inspection of component time course (menu **Plot > Component activations (scroll)** (not show here)).

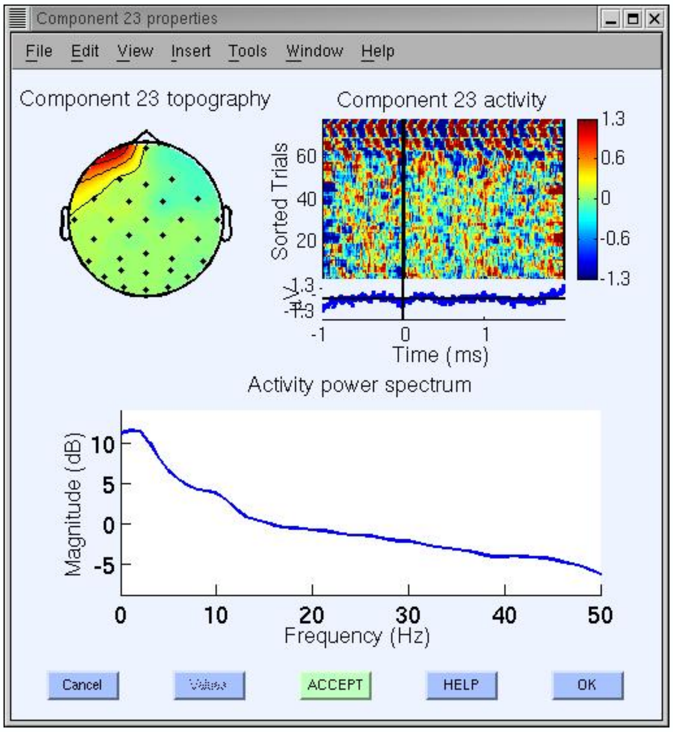


Since this component accounts for eye activity, we may wish to subtract it from the data before further analysis and plotting. If so, click on the bottom green "**Accept**" button (above) to toggle it into a red "**Reject**" button (note that at this point components are only marked for rejection; to subtract marked components, see next chapter I.9.5. Subtracting ICA components from data). Now press **OK** to go back to the main component property window.

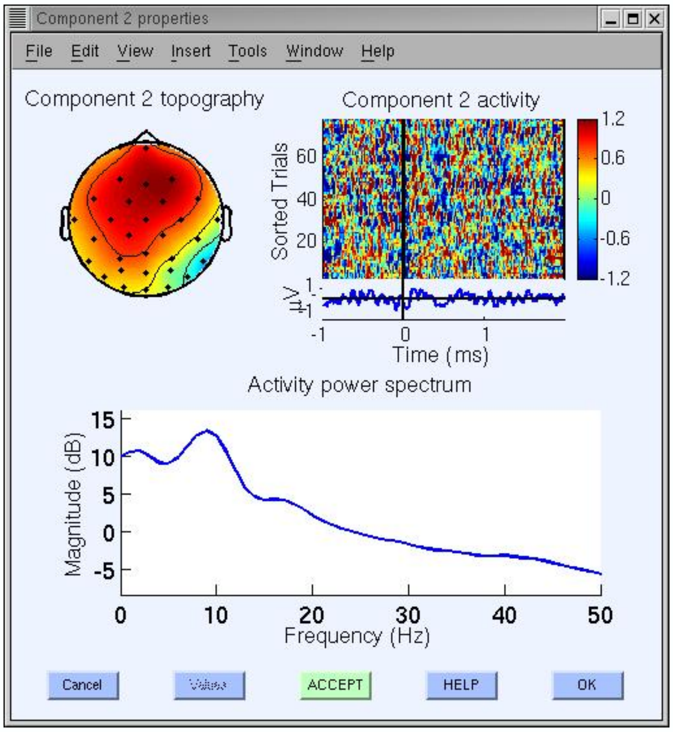
An other artifact example in our decomposition is component 31, which appears to be typical muscle artifact component. This components is spatially localized and show high power at high frequencies (20−50 Hz and above) as shown below.



Artifactual components often encountered (but not present in this decomposition) are single−channel (channel−pop) artifacts in which a single channel goes 'off,' or line−noise artifacts such as 23 (the ERP image plot below shows that it picked up some noise line at 60 Hz especially in trials 65 and on).



Many other components appear to be brain−related (**Note:** Our sample decomposition used in this tutorial is based on clean EEG data, and may have fewer artifactual components than decompositions of some other datasets). The main criteria for recognizing brain−related components are that they have 1) dipole−like scalp maps, 2) spectral peaks at typical EEG frequence is (i.e., 'EEG−like' spectra) and, 3) regular ERP−image plots (meaning that the component does not account for activity occurring in only a few trials). The component below has a strong alpha band peak at about 10 Hz and a scalp map distribution compatible with a left occipital cortex brain source. When we localize ICA sources using single−dipole or dipole−pair source localization many of the 'EEG−like' components can be fit with very low residual variance (e.g., under 5%). See our **BESA example** for details. **Note:** If you have a copy of BESA software version 2.2 available, ask us for BESA localization and plotting routines under EEGLAB.



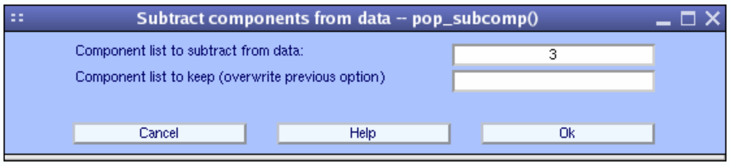
What if a component looks to be "half artifact, half brain−related"? In this case, we may ignore it, or may try running ICA decomposition again on a cleaner data subset or using other ICA training parameters. As a rule of thumb, we have learned that removing artifactual epochs containing one−of−a−kind artifacts is very useful for obtaining 'clean' ICA components.

Important note: we believe an optimal strategy is to, 1) run ICA 2) reject bad epochs (see the functions we developed to detect artifactual epochs (and channels, if any) in the **tutorial on artifact rejection**). In some cases, we do not hesitate to remove more than 10% of the trials, even from 'relatively clean' EEG datasets. We have learned that it is often better to run this first ICA composition on very short time windows. 3) Then run ICA a second time on the 'pruned' dataset. 4) Apply the resulting ICA weights to the same dataset or to longer epochs drawn from the same original (continuous or epoched) dataset (for instance to copy ICA weights and sphere information from dataset 1 to 2. First, call menu **Edit > Dataset info** of dataset 2. Then enter

"**ALLEEG(1).icaweights**" in the "**ICA weight array ...**" edit box, "**ALLEEG(1).icasphere**" in the "**ICA sphere array ...**" edit box, and press "**OK**").

I.9.5. Subtracting ICA components from data

We don't always subtract whole components from our datasets because we study individual components' activity. However, when we want to remove components, we use menu **Tools > Remove components**, which calls the pop_subcomp() function. The component numbers present by default in the resulting window (below) are those marked for rejection in the previous **Tools > Reject using ICA > Reject components by map** component rejection window (using the "**Accept**/**Reject**" buttons). Enter the component numbers you wish to reject and press **OK**.


Subtract components from data -- pop_subcomp()
Component list to subtract from data:
3
Component list to keep (overwrite previous option)
Cancel
Help
Ok

A window will pop up, plotting channel ERP before (in blue) and after (in red) component(s) subtraction and asking you if you agree with the new data ERP.

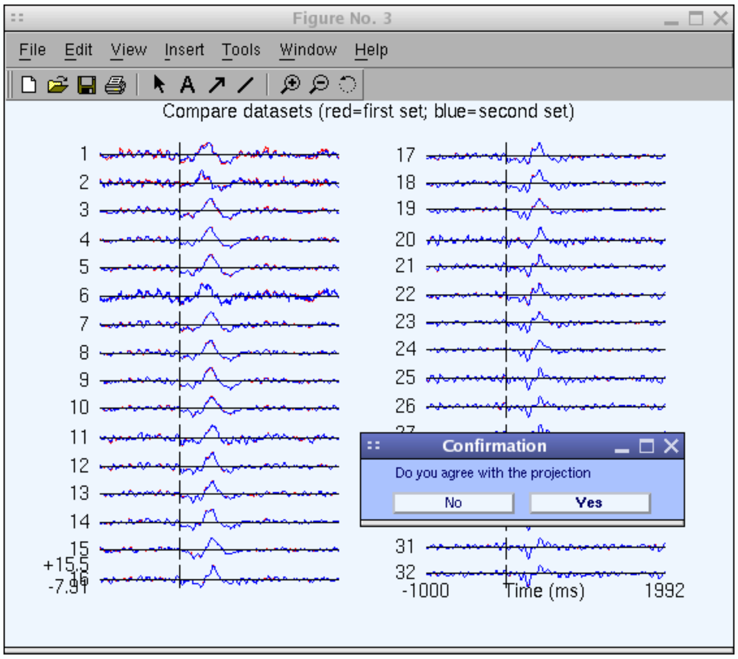
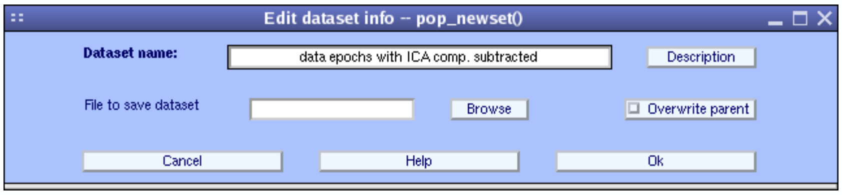


Figure No. 3
File Edit View Insert Tools Window Help
Compare datasets (red=first set; blue=second set)
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
31
32
+15.5
-7.91
-1000
Time (ms)
1992
Confirmation
Do you agree with the projection
No
Yes

Press **Yes**. A last window will pop up asking you if you want to rename the new data set. Give it a name and again press **OK**.



Dataset name: data epochs with ICA comp. subtracted

File to save dataset

☐ Overwrite parent

Note that storing the new dataset in Matlab memory does not automatically store it permanently on disk. To do this, select **File > Save current dataset**. Note that we will pursue with all components, so you can appreciate the contribution of artifactual components to the ERP. Go back to the previous dataset using the **Dataset** top menu.

Note: If you try to run ICA on this new dataset, the number of dimensions of the data will have been reduced by the number of components subtracted. We suggest again 'baseline−zeroing' the data (if it is epoched when some components have been removed, data epoch−baseline means may change). To run ICA on the reduced dataset, use the **'pca'** option under the **Tools > Run ICA** pop−up window to reduce the data dimensions to the number of remaining components before running ICA (see **runica()**). If the amount of data has not changed, ICA will typically return **the same** (remaining) independent components −− which were, after all, already found to be maximally independent for these data.

I.9.6. Retaining multiple ICA weights in a dataset

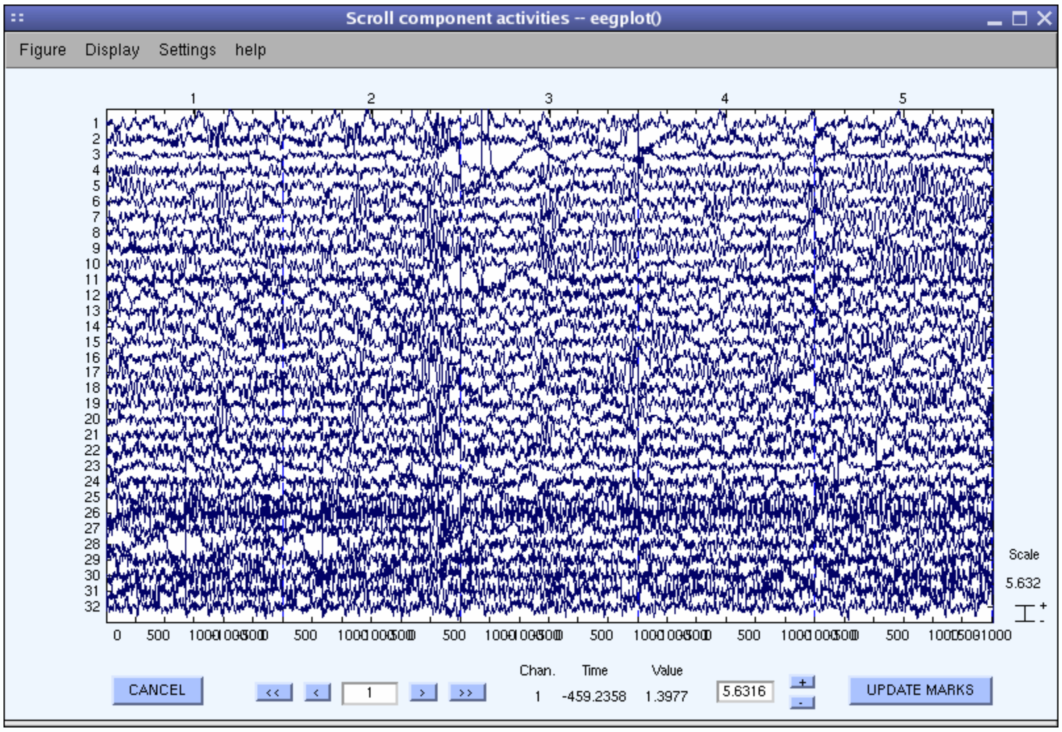
To retain multiple copies of ICA weights (e.g. EEG.weights and EEG.sphere), use the extendibility property of Matlab structures. On the Matlab command line, simply define new weight and sphere variables to retain previous decomposition weights. For example,

```
>> EEG.icaweights2 =  EEG.icaweights;% Store existing ICA weights matrix
>> EEG.icasphere2 =  EEG.icasphere;     % Store existing ICA sphere matrix
>> [ALLEEG EEG] = eeg_store(ALLEEG, EEG, CURRENTSET); % copy to
EEGLAB memory
>> EEG = pop_runica(EEG);        % Compute ICA weights and sphere again using
                                          % binica() or runica(). Overwrites new
weights/sphere matrices
                                       % into EEG.icaweights, EEG.icasphere
>> [ALLEEG EEG] = eeg_store(ALLEEG, EEG, CURRENTSET); % copy to
EEGLAB memory
```

Both sets of weights will then be saved when the dataset is saved, and reloaded when it is reloaded. See the **script tutorial** for more information about writing Matlab scripts for EEGLAB.

I.9.7. Scrolling through component activations

To scroll through component activations (time courses), select **Plot > Component activations (scroll)**. Scrolling through the ICA activations, one may easily spot components accounting for characteristic artifacts. For example, in the scrolling **eegplot()** below, component 1 appears to account primarily for blinks, components 7 and 31 for lateral eye movements, and component 25 possibly for muscle activity. Check these classifications using the complementary visualization produced by **Plot > Component properties** .

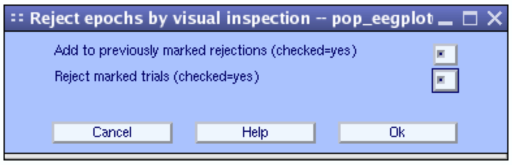


In the next tutorial, we show more ways to use EEGLAB to study ICA components of the data.

I.10. Working with ICA components

I.10.1. Rejecting data epochs by inspection using ICA

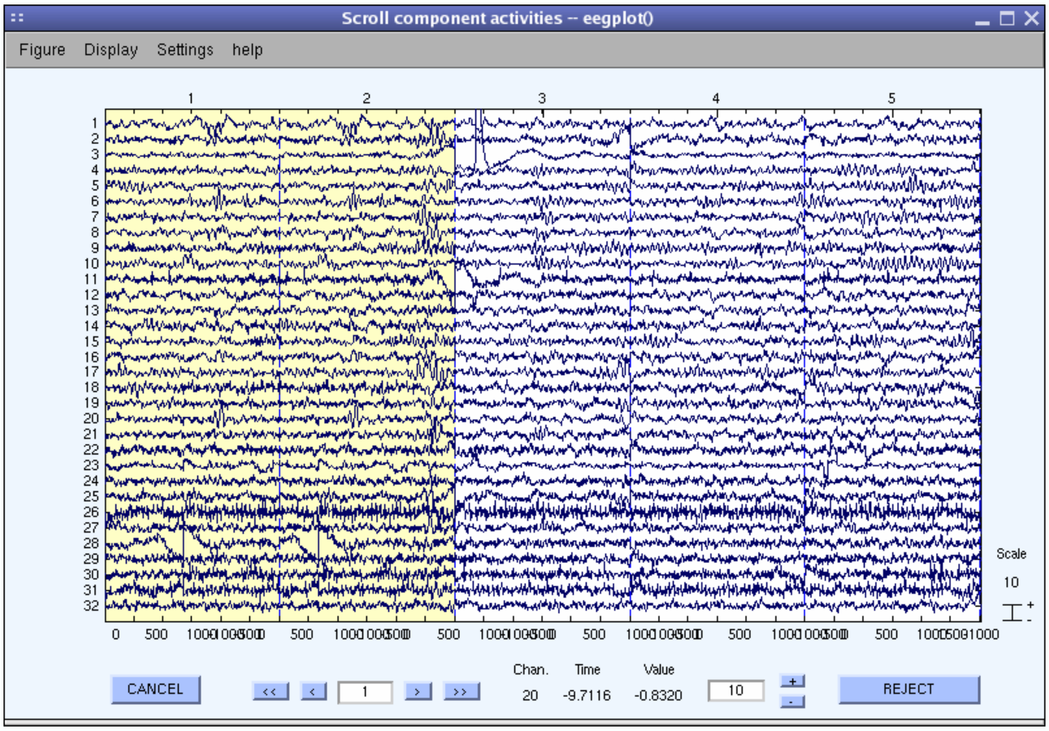
To reject data by visual inspection of its ICA component activations, select **Tools > Reject data using ICA > Reject by inspection** (calling pop_eegplot()). Note that marked epochs can be kept in memory before being actually rejected (if the option "**Reject marked trials (checked=yes)**" below is not set). Thus it is possible to mark trials for rejection and then to come back and update the marked trials. The first option ("**Add to previously marked rejections (checked=yes)**") allows us to include previous trial markings in the scrolling data window or to ignore/discard them. Since it is the first time we have used this function on this dataset, the option to plot previously marked trials won't have any effect. Check the checkbox to reject labeled trials immediately when the scrolling window is closed. Press **OK**.


Reject epochs by visual inspection -- pop_eegplot

Add to previously marked rejections (checked=yes)

Reject marked trials (checked=yes)

First, adjust the scale by entering "**10**" in the scale text edit box (lower right). Now, click on the data epochs you want to mark for rejection. For this exercise, highlight two epochs. You can then deselect the rejected epochs by clicking again on them. Press the "**Reject**" button when finished and enter a name for the new dataset (the same dataset minus the rejected epochs), or press the "**Cancel**" button to cancel the operation.



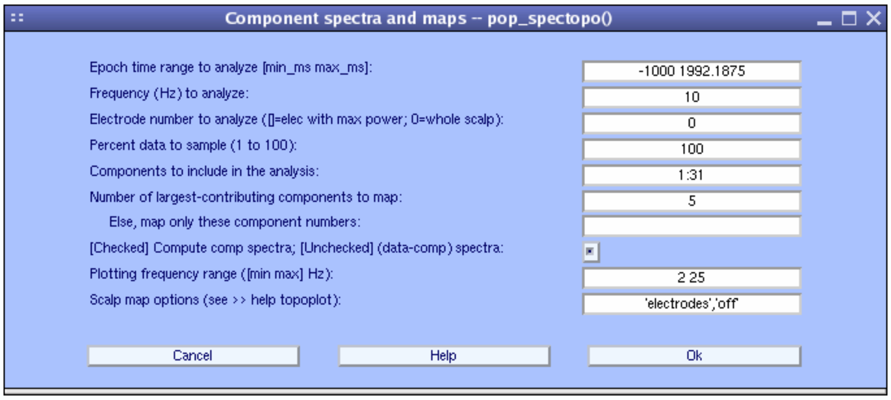
At this point, you may spend some time trying out the advanced rejection functions we developed to select and mark artifactual epochs based on ICA component maps and activations. For directions, see the **Data rejection tutorial**. Then, after rejecting 'bad' epochs, run ICA decomposition again. Hopefully, by doing this you may obtain a 'cleaner' decomposition.

Important note: what do we mean by a cleaner ICA decomposition? ICA takes all its training data into consideration. When too many types (i.e., scalp distributions) of 'noise' − complex movement artifacts, electrode 'pops', etc −− are left in the training data, these unique/irreplicable data features will 'draw the attention' of ICA, producing a set of component maps including many single−channel or 'noisy'−appearing components. The number of components (degrees of freedom) devoted to decomposition of brain EEG alone will be correspondingly reduced. Therefore, presenting ICA with as much 'clean' EEG data as possible is the best strategy (note that blink and other stereotyped EEG artifacts do not necessarily have to be removed since they are likely to be isolated as single ICA components).

For this tutorial, we decide to accept our initial ICA decomposition of our data and proceed to study the nature and behavior(s) of its independent components. First, we review a series of functions whose purpose is to help us determine which components to study and how to study them.

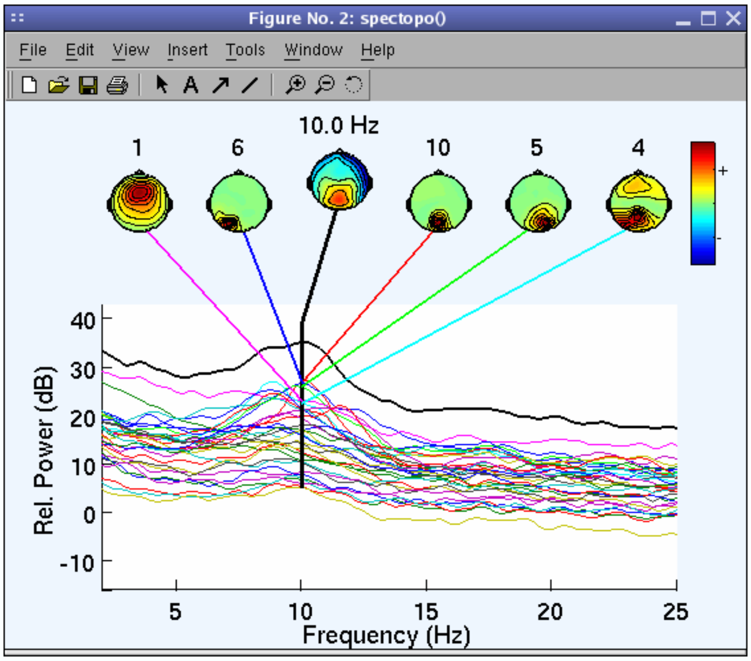
I.10.2. Plotting component spectra and maps

It is of interest to see which components contribute most strongly to which frequencies in the data. To do so, select **Plot > Component spectra and maps**. This calls **pop_spectopo()**. Its first input is the epoch time range to consider, the second the percentage of data to sample at random (smaller percentages speeding the computation, larger percentages being more definitive). Since our EEG dataset is fairly small, we choose to change this value to "**100**" (= all of the data). We will then visualize which components contribute the most at 10 Hz, entering "**10**" in the "**Scalp map frequency** " text box. We simply scan all components, the default in "**Components to consider**". Press **OK**.

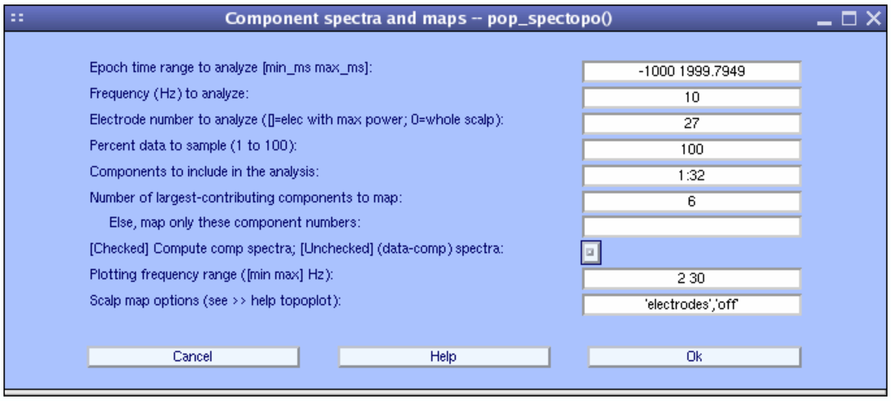


Epoch time range to analyze [min_ms max_ms]:	-1000 1992.1875
Frequency (Hz) to analyze:	10
Electrode number to analyze ([]=elec with max power; 0=whole scalp):	0
Percent data to sample (1 to 100):	100
Components to include in the analysis:	1:31
Number of largest-contributing components to map:	5
Else, map only these component numbers:	
[Checked] Compute comp spectra; [Unchecked] (data-comp) spectra:	☒
Plotting frequency range ([min max] Hz):	2 25
Scalp map options (see >> help topoplot):	'electrodes','off'

The **spectopo()** window (below) appears.

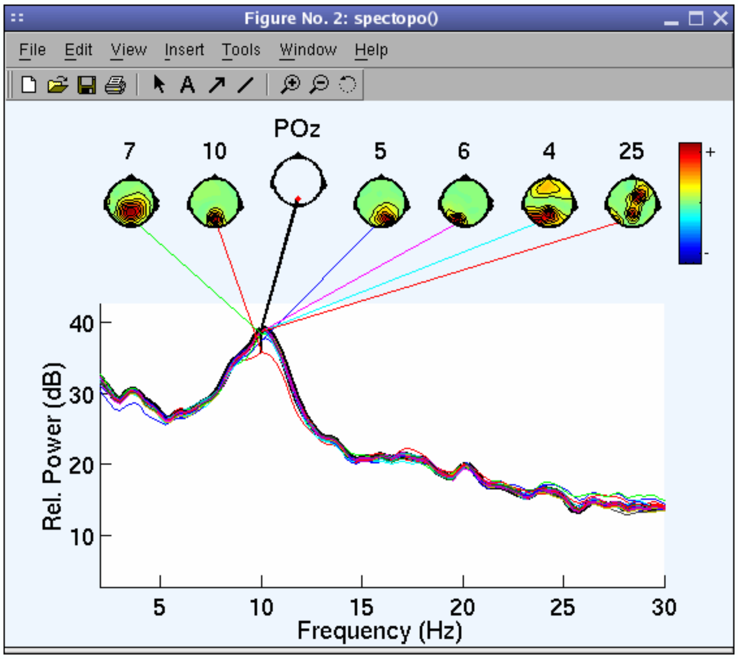


In the previous window, we plotted the spectra of each component. A more accurate strategy (for technical reasons) is to plot the data signal minus the component activity and estimate the decrease in power in comparison to the original signal at one channel (it is also possible to do it at all channel but it requires to compute the spectrum of the projection of each component at each channel which is computationally intensive). To do so, go back to the previous interactive window, choose explicitly to plot component's contribution at channel **27** (POz) where power appears to be maximum at **10** Hz using the "**Plotting channel (number):**" field, uncheck the checkbox "**[checked] compute component spectra...**". Set percent to "**100**" as before. Finally we will display **6** component maps instead of 5 (default) (note that all component spectra will be shown) and we will set the maximum frequency to be plotted at **30** Hz using the "**Plotting frequency range**" option in the bottom panel (below). Press **OK** when done.


Component spectra and maps -- pop_spectopo()

Epoch time range to analyze [min_ms max_ms]:	-1000 1999.7949
Frequency (Hz) to analyze:	10
Electrode number to analyze ([]=elec with max power; 0=whole scalp):	27
Percent data to sample (1 to 100):	100
Components to include in the analysis:	1:32
Number of largest-contributing components to map:	6
Else, map only these component numbers:	
[Checked] Compute comp spectra; [Unchecked] (data-comp) spectra:	☐
Plotting frequency range ([min max] Hz):	2 30
Scalp map options (see >> help topoplot):	'electrodes','off'

The spectopo() figure appears (as below).



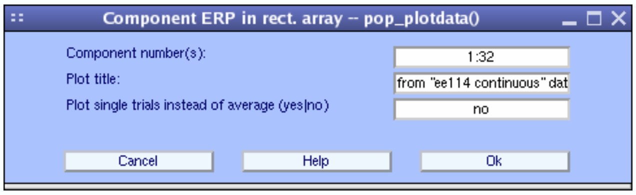
The following text is displayed

```
Component 1 percentage variance accounted for:  3.07
Component 2 percentage variance accounted for:  3.60
Component 3 percentage variance accounted for: −0.05
Component 4 percentage variance accounted for:  5.97
Component 5 percentage variance accounted for: 28.24
Component 6 percentage variance accounted for:  6.15
Component 7 percentage variance accounted for: 12.68
Component 8 percentage variance accounted for: −0.03
Component 9 percentage variance accounted for:  5.04
Component 10 percentage variance accounted for: 52.08
Component 11 percentage variance accounted for:  0.79
....
```

According to variance accounted for and component order, component 10 appears to account for more than 50% of power at 10 Hz for channel POz (note that a channel number has to be entered otherwise component contributions are not computed).

I.10.3. Plotting component ERPs

After seeing which components contribute to frequency bands of interest, it is interesting to look at which components contribute the most to the ERP. A first step is to view the component ERPs. To Plot component ERPs, select **Plot > Component ERPs > In rectangular array**, which calls function **pop_plotdata()**. Then press **OK.**



Component ERP in rect. array -- pop_plotdata()

Component number(s):	1:32
Plot title:	from "ee114 continuous" dat
Plot single trials instead of average (yes|no)	no

The **plotdata()** window below pops up, showing the average ERP for all 31 components.

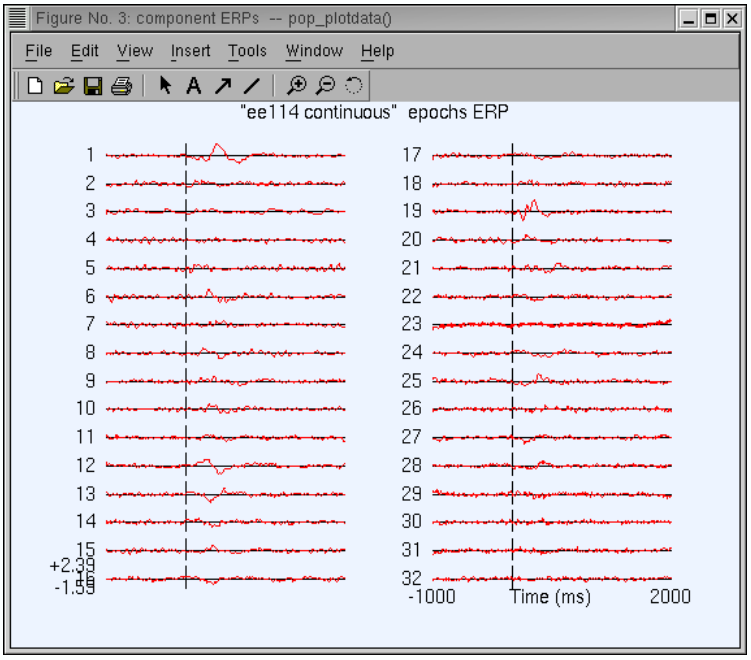
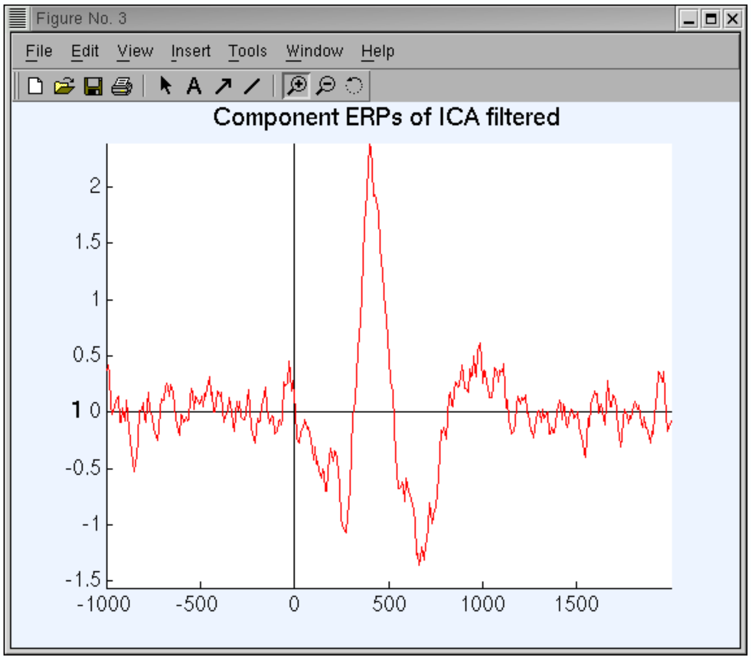


Figure No. 3: component ERPs -- pop_plotdata()
File Edit View Insert Tools Window Help
"ee114 continuous" epochs ERP
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
+2.39
-1.59
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
-1000
Time (ms)
2000

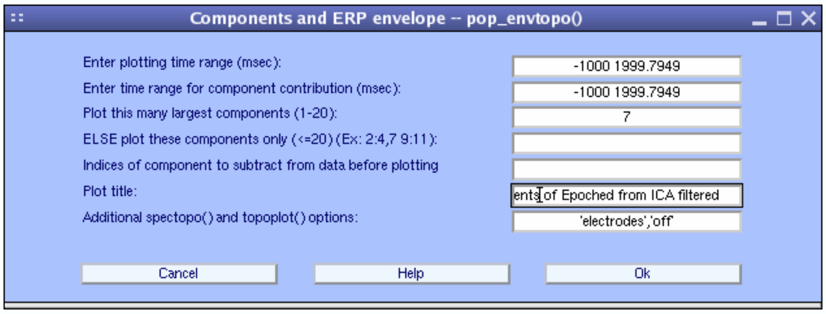
Click on the component−1 trace (above) to plot this trace in new window (as below).



As for electrodes, use menu **Plot > Plot/Compare comp. ERPs** to plot component ERP differences accross multiple datasets.

I.10.4. Plotting component ERP contributions

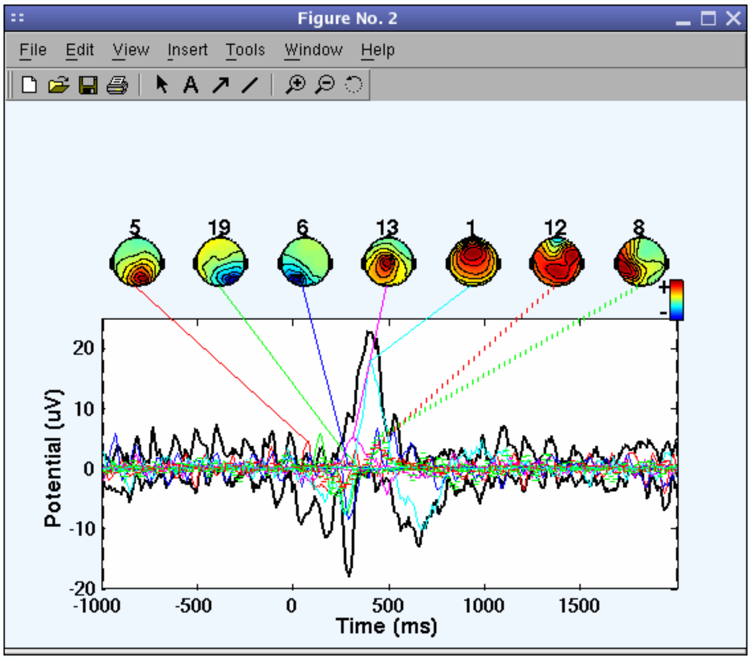
To plot the contribution of component ERPs to the data ERP, select **Plot > Component ERPs > On same axis (with maps)**, which calls pop_envtopo(). Simply press **OK** to plot the 7 components that contribute the most to the average ERP of the dataset. Note artifactual components can be subtracted from the data prior to plot the ERP using the "**Indices of component to subtract ...**" edit box.



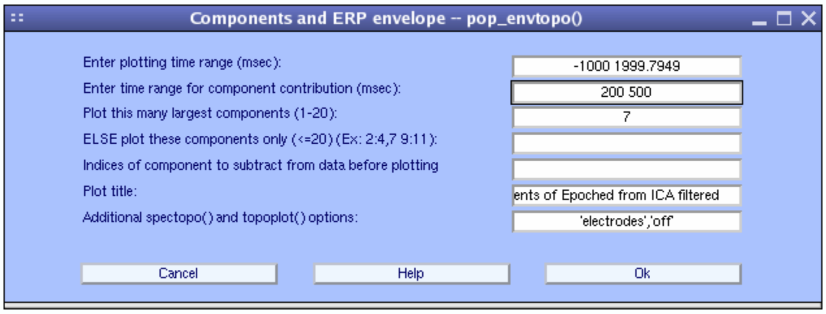
Components and ERP envelope -- pop_envtopo()

Enter plotting time range (msec):	-1000 1999.7949
Enter time range for component contribution (msec):	-1000 1999.7949
Plot this many largest components (1-20):	7
ELSE plot these components only (<=20) (Ex: 2:4,7 9:11):	
Indices of component to subtract from data before plotting	
Plot title:	ents of Epoched from ICA filtered
Additional spectopo() and topoplot() options:	'electrodes','off'

In the [envtopo()]() plot (below), the black thick line indicates the data envelope (i.e. minimum and maximum of all channel at every time point) and the colored show the component ERPs.

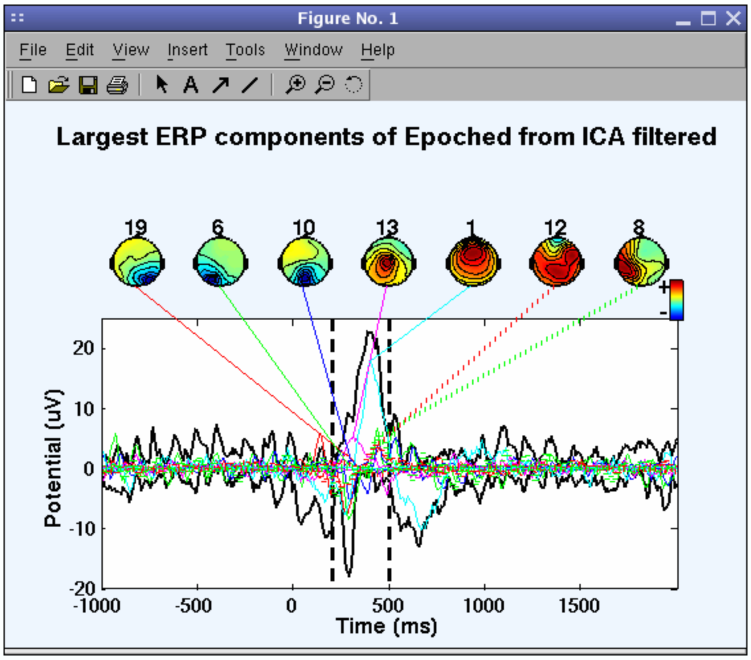


The picture above looks messy, so again call the [pop_envtopo()]() window and zoom in on time range from **200** ms to **500** ms post−stimulus, as indicated below.



Enter plotting time range (msec): -1000 1999.7949
Enter time range for component contribution (msec): 200 500
Plot this many largest components (1-20): 7
ELSE plot these components only (<=20) (Ex: 2:4,7 9:11):
Indices of component to subtract from data before plotting
Plot title: ents of Epoched from ICA filtered
Additional spectopo() and topoplot() options: 'electrodes','off'

We can see (below) that near 400 ms component 1 contributes most strongly to the ERP.

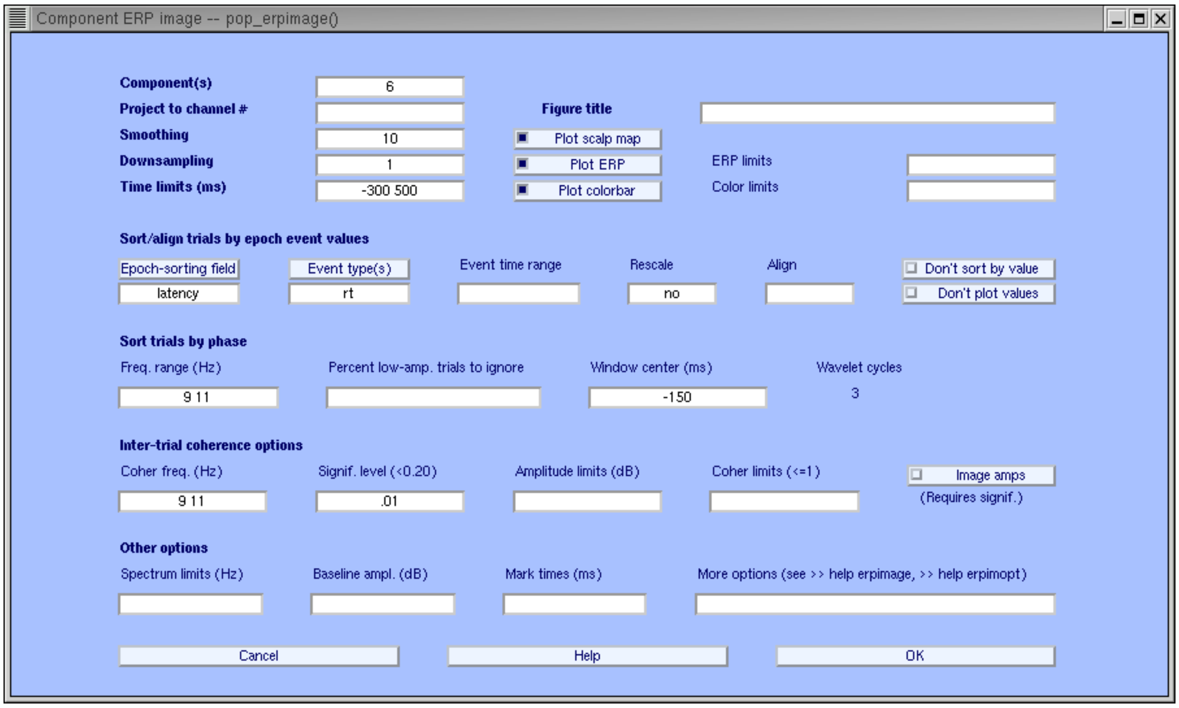


On the command line, the function also returns the percentage of variance accounted for by each component:

```
Component 1 percentage variance accounted for: 65.90
Component 2 percentage variance accounted for: −6.59
Component 3 percentage variance accounted for: 1.95
Component 4 percentage variance accounted for: 0.65
Component 5 percentage variance accounted for: 4.59
Component 6 percentage variance accounted for: 9.19
Component 7 percentage variance accounted for: −7.11
Component 8 percentage variance accounted for: 9.14
Component 9 percentage variance accounted for: −1.14
Component 10 percentage variance accounted for: 5.35
Component 11 percentage variance accounted for: −0.13
Component 12 percentage variance accounted for: 22.17
...
```

I.10.5. Component ERP−image plotting

To plot ERP−image figures for component activations, select **Plot > Component ERP image** (calling **pop_erpimage()**). This function works exactly as the one we used for plotting channel ERP images, but instead of visualizing activity at one electrode, the function here plots the activation of one component. Enter the following parameters in the interactive window to sort trials by phase at 10 Hz and 0 ms, to image reaction time, power and Inter−Trial Coherence (see the **ERP−image** tutorial for more information).


Component ERP image -- pop_erpimage()

Component(s) 6
Project to channel #
Figure title
Smoothing 10 — ■ Plot scalp map
Downsampling 1 — ■ Plot ERP — ERP limits
Time limits (ms) -300 500 — ■ Plot colorbar — Color limits

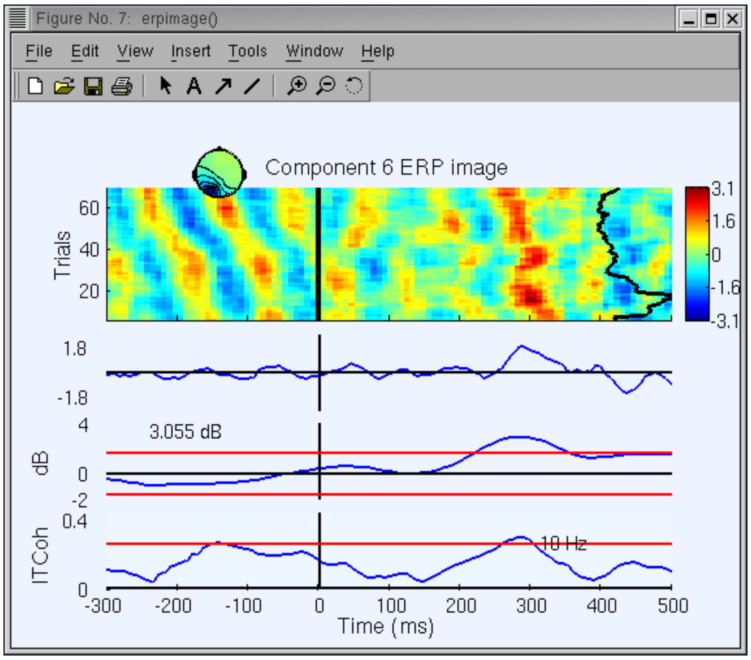
Sort/align trials by epoch event values
Epoch-sorting field: latency — Event type(s): rt — Event time range — Rescale: no — Align — ☐ Don't sort by value — ☐ Don't plot values

Sort trials by phase
Freq. range (Hz): 9 11 — Percent low-amp. trials to ignore — Window center (ms): -150 — Wavelet cycles: 3

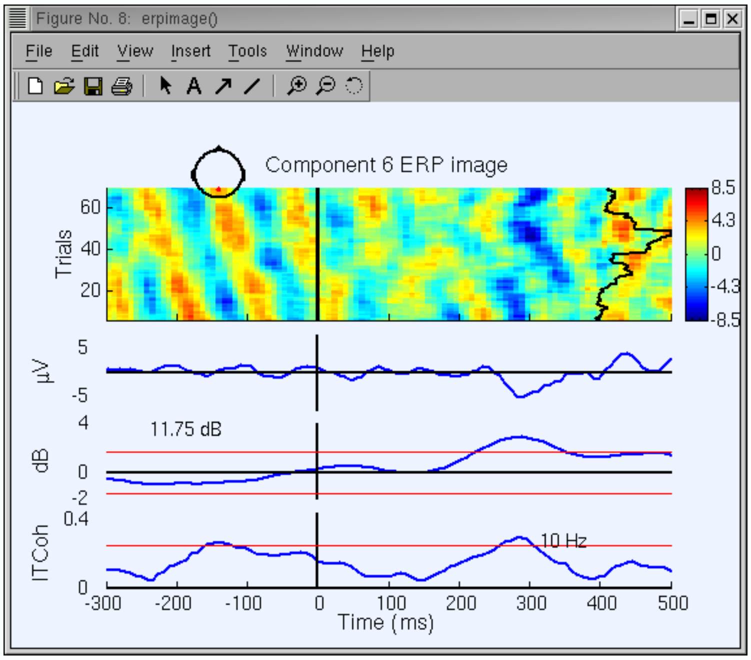
Inter-trial coherence options
Coher freq. (Hz): 9 11 — Signif. level (<0.20): .01 — Amplitude limits (dB) — Coher limits (<=1) — ☐ Image amps (Requires signif.)

Other options
Spectrum limits (Hz) — Baseline ampl. (dB) — Mark times (ms) — More options (see >> help erpimage, >> help erpimopt)

For component 6 (below) we observe in the **erpimage()** figure that phase at the analysis frequency (9.1 Hz) is evenly distributed in the time window −300 to 0 ms (as indicated by the bottom trace showing the inter−trial coherence (ITC) or phase−locking factor). This component accounts for much of the EEG power at 10 Hz, but for little if any of the average ERP. Overall, mean power at the analysis frequency does not change across the epoch (middle blue trace) and phase at the analysis frequency is not reset by the stimulus (bottom blue trace). Here again, the red lines show the bootstrap significance limits (for this number of trials).



Note: As scale and polarity information is distributed in the ICA decomposition *(not* lost!) between the projection weights (column of the inverse weight matrix, **EEG.icawinv**) and rows of the component activations matrix (**EEG.icaact**), the absolute amplitude and polarity of component activations are meaningless and the activations have no unit of measure (through they are **proportional to** microvolt). To recover the absolute value and polarity of activity accounted for by one or more components at an electrode, image the back−projection of the component activation(s) at that channel. Go back to the previous ERP−image window, use the same parameters and set "**Project to channel #**" to 27. Note that the ERP is reversed in polarity and that absolute unit for power has changed.

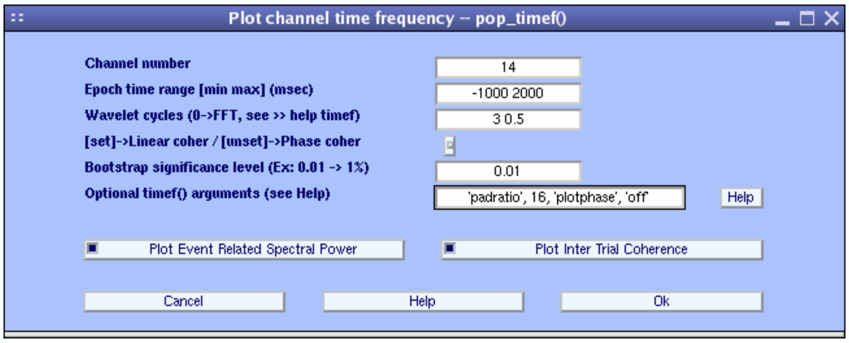


In the next tutorial, we show how to use EEGLAB to perform and visualize time/frequency decompositions of channel activities or independent component activations.

I.11. Time/frequency decomposition

I.11.1. Decomposing channel data

To detect transient event−related spectral perturbation (ERSP) (power shifts) and inter−trial coherence (ITC) (phase−locking) events time locked to experimental events in epoched EEGLAB datasets, select **Plot > Time frequency transforms > Channel time−frequency** (calling **pop_timef()**). Below, we enter "**14**" (Cz) for the "**Channel number**", "**.01**" for the "**Bootstrap significance level**", and set the optional parameter **" 'padratio' "** to "**16**" as below (a very high over−sampling factor for frequencies, giving a smooth looking plot at some computational cost). We let the other defaults remain and press **OK**.


Plot channel time frequency -- pop_timef()
Plot Event Related Spectral Power
Plot Inter Trial Coherence
Help
Cancel
Help
Ok

Channel number	14
Epoch time range [min max] (msec)	-1000 2000
Wavelet cycles (0->FFT, see >> help timef)	3 0.5
[set]->Linear coher / [unset]->Phase coher	
Bootstrap significance level (Ex: 0.01 -> 1%)	0.01
Optional timef() arguments (see Help)	'padratio', 16, 'plotphase', 'off'

The timef() window below appears. The *top image* shows mean event−related changes in spectral power (from pre−stimulus baseline) at each time during the epoch and at each frequency (< 50 Hz). To inspect these changes more closely, click on the color image. A new window will pop up. Enabling Matlab zoom allows zooming in to any desired time/frequency window. The ERSP image shows a brief but significant decrease in power at about 370 ms at 8 Hz (click on the image to zoom in and determine the exact frequency), a power increase centered at 13.5 Hz and starting at 300 ms, followed by a power increase at 20−28 Hz and 30−34 Hz. Note that the method used to asses significance is based on random data shuffling, so the exact significance limits of these features may appear slightly different.

The *upper left panel* shows the baseline mean power spectrum, and the lower part of the upper panel, the ERSP envelope (low and high mean dB values, relative to baseline, at each time in the epoch).

The *lower image* shows is the Inter−Trial coherence (ITC) at all frequencies. We previously encountered the ITC when we explained the ERPimage function. A significant ITC indicates that the EEG activity at a given time and frequency in single trials becomes phase−locked (not phase−random with respect to the time−locking experimental event). Here, a significant ITC appears briefly at about 10 Hz (at about the same time as the power increase in the *upper panel*), but this effect might well become insignificant using a more rigorous significance threshold. Note that the time/frequency points showing significant ITC and ERSP are not necessarily identical. In this plot, ITC hardly reaches significance and cannot be interpreted. The help message for the timef() function contains information about all its parameters, the images and curve plotted, and their meanings.

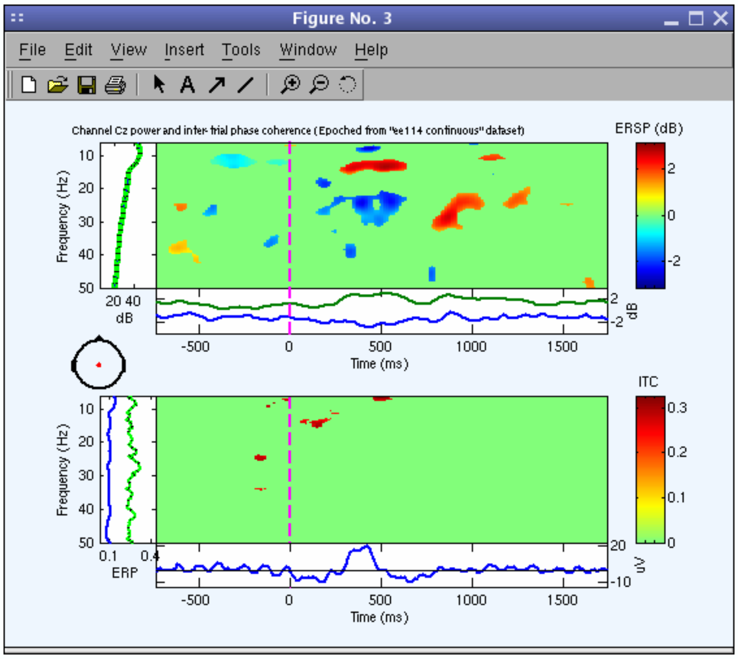
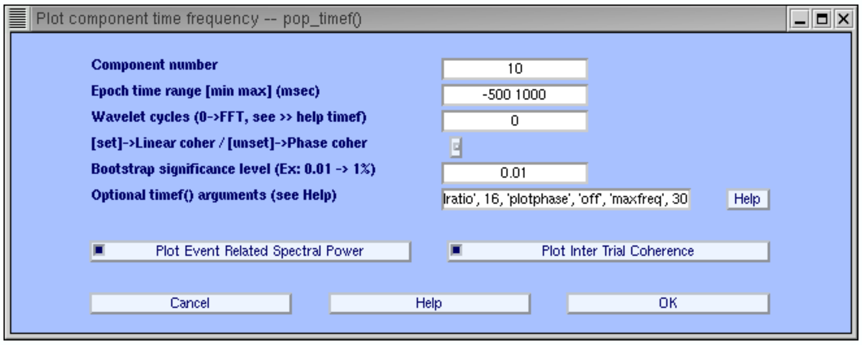


Figure No. 3
File Edit View Insert Tools Window Help
Channel Cz power and inter-trial phase coherence (Epoched from "ee114 continuous" dataset)
ERSP (dB)
Frequency (Hz)
dB
Time (ms)
ITC
ERP
uV

I.11.2. Computing component time/frequency transforms

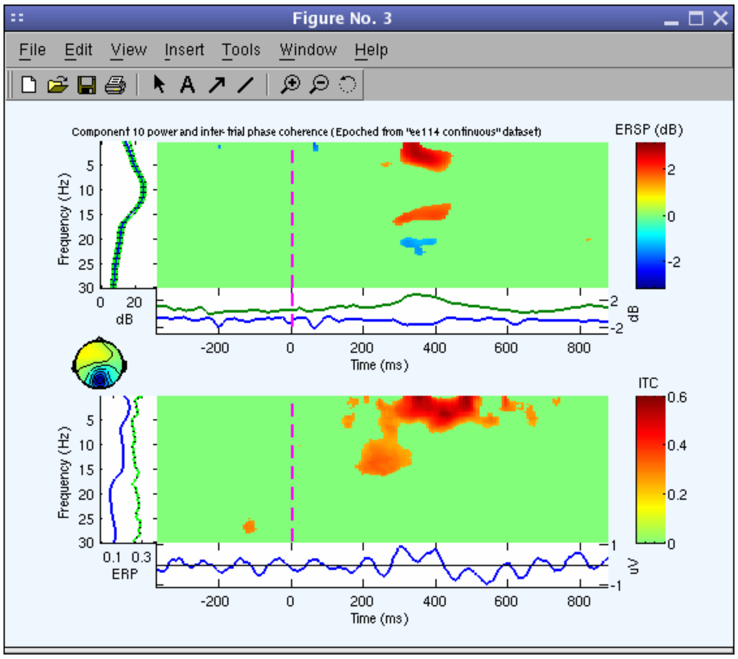
It is more interesting to look at time−frequency decompositions of component activations than of separate channel activities, since independent components may directly index the activity of one brain EEG source, whereas channel activities sum potentials volume−conducted from different parts of the brain.

To plot a component time−frequency transform, we select **Plot** > **Time/frequency transforms > Component time−frequency** (calling pop_timef()). Enter "**10**" for the "**Component number**" to plot, "**[−500 1000]**" for the "**Epoch time range**", "**0**" (FFT) for "**Wavelet cycles**", and "**.01**" for the **Bootstrap significance level**. We change **'padratio'** to **16** and add the optional argument " '**maxfreq', '30'** " to visualize only frequencies up to 30 Hz. Again, we press **OK**. **Note:** timef() decompositions using FFTs allow computation of lower frequencies than wavelets, since they compute as low as one cycle per window, whereas the wavelet method uses a fixed number of cycles (default 3) for each frequency.


Plot component time frequency -- pop_timef()

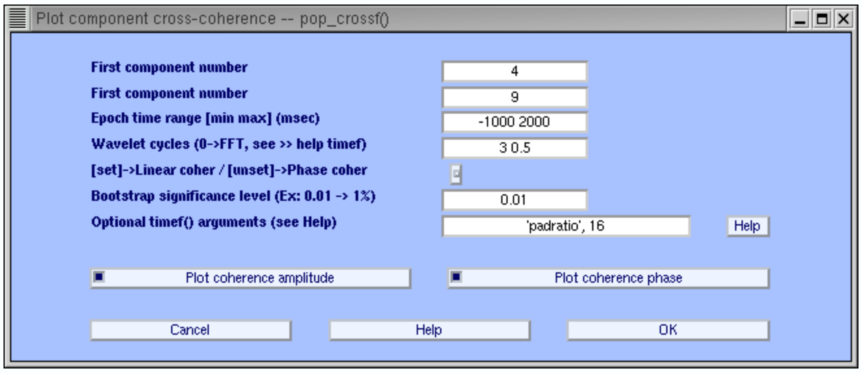
Component number	10
Epoch time range [min max] (msec)	-500 1000
Wavelet cycles (0->FFT, see >> help timef)	0
[set]->Linear coher / [unset]->Phase coher	
Bootstrap significance level (Ex: 0.01 -> 1%)	0.01
Optional timef() arguments (see Help)	ratio', 16, 'plotphase', 'off', 'maxfreq', 30

The following timef() window appears. The ITC image (*lower panel*) shows strong synchronization between the component activity and stimulus appearance, first near 15 Hz then near 4 Hz. The ERSP image (*upper panel*) shows that the 15−Hz phase−locking is followed by a 15−Hz power increase, and that the 4−Hz phase−locking event is accompanied by, but outlasts, a 4−Hz power increase. Note that the appearance of oscillatory activity in the ERP (*trace under bottom ITC image*) before and after the stimulus is **not** significant according to ITC.



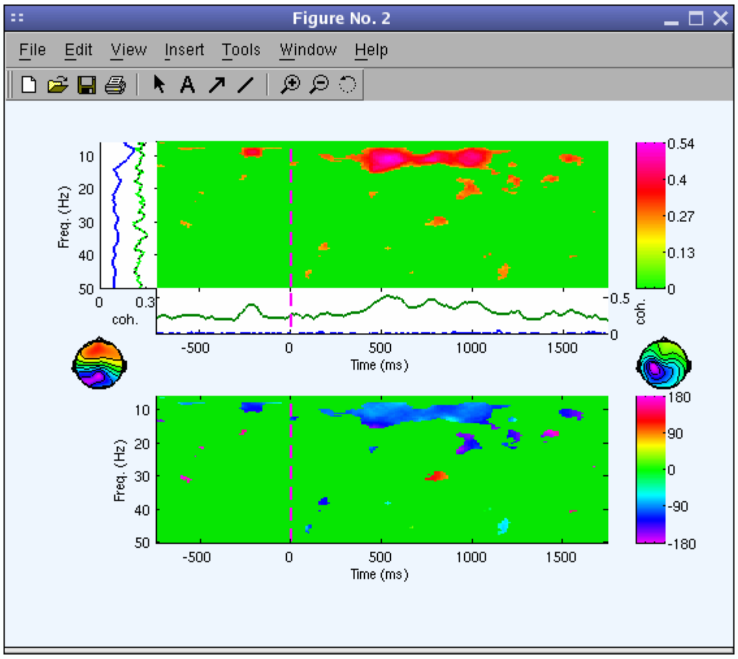
I.11.3. Computing component cross−coherences

To determine the degree of synchronization between the activations of two components, we may plot their event−related cross−coherence (a concept first demonstrated for EEG analysis by Rappelsberger). Even though independent components are (maximally) independent over the whole time range of the training data, they may become transiently (partially) synchronized in specific frequency bands. To plot component cross−coherence, select **Plot > Time−frequency transforms > Component cross−coherence**, which calls pop_crossf(). Below, we enter components "**4**" and "**9**" (Use any components in your decomposition), "**Bootstrap significance level**" to "**0.01**", set "'**padratio**'" to "**16**". We again press **OK**.



First component number	4
First component number	9
Epoch time range [min max] (msec)	-1000 2000
Wavelet cycles (0->FFT, see >> help timef)	3 0.5
[set]->Linear coher / [unset]->Phase coher	
Bootstrap significance level (Ex: 0.01 -> 1%)	0.01
Optional timef() arguments (see Help)	'padratio', 16

In the crossf() window below, the two components become synchronized (*top panel*) around 11.5 Hz (click on the image to zoom in). The *upper panel* shows the coherence magnitude (between 0 and 1, 1 representing two perfectly synchronized signals). The *lower panel* indicates the phase difference between the two signals at time/frequency points where cross−coherence magnitude (in the *top panel*) is significant. In this example, the two components are synchronized with a phase offset of about −120 degrees (this phase difference can also be plotted as latency delay in ms, using the minimum−phase assumption. See crossf() help for more details about the function parameters and the plotted variables).



One can also use **Plot > Time−frequency transforms > Channel cross−coherence** to plot event−related cross−coherence between a pair of scalp channels, but here relatively distant electrode channels may appear synchronized only because a large EEG source projects to both of them. Other source confounds may also affect channel coherences in unintuitive ways. Computing cross−coherences on independent data components may index transient synchronization between particular cortical domains.

I.11.4. Plotting ERSP time course and topography

Recall that spectral perturbations at a single−analysis frequency and channel or component in the single epochs (sorted by some relevant value) can be imaged using erpimage() or by selecting **Plot > Component|Channel ERP image**.

Called from the command line (see EEGLAB script writing), the timef() and crossf() routines can return the data for each part of their figures as a Matlab variable. Accumulating the ERSP and ITC images (plus the ERSP baseline and significance−level information) for all channels (e.g., via an EEGLAB script) allows use of another toolbox routine, tftopo() (currently not available from the EEGLAB menu).

Tutorial concluding remark

This tutorial only gives a rough idea of the utility of EEGLAB for processing single−trial and averaged EEG, The analyses of the sample dataset we presented here are by no way definitive. One should examine the activity and scalp map of each independent component carefully, noting its dynamics and its relationship to behavior, to begin to understand its function. Further questions may be more important: How are the activations of pairs of maximally independent components inter−related? How do their relationships evolve across time or change across conditions? Are independent components of different subjects related, and if so, how? We believe that EEGLAB is a suitable environment for exploring these and other questions about EEG dynamics.

II. Importing/exporting data and event or epoch information into EEGLAB

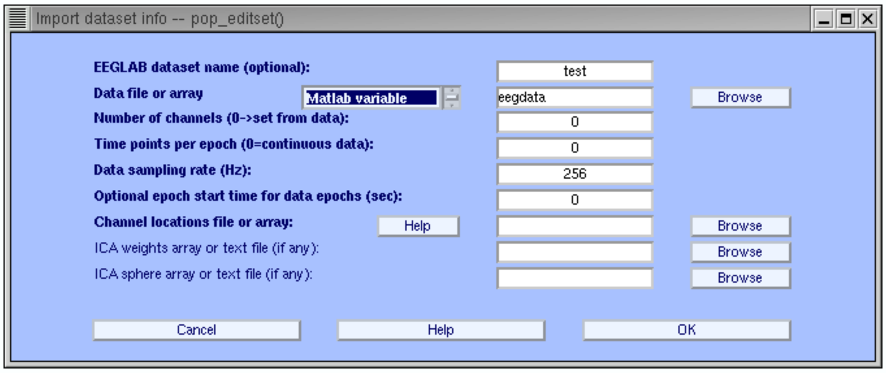
II.1. Importing continuous data

II.1.1. Importing a Matlab array

We first construct a 2−D Matlab array 'eegdata' containing simulated EEG data in which rows are channels and columns are data points:

```
>> eegdata = rand(32, 256*100);
% build a matrix of random test data (32 channels, 100 seconds at 256 Hz)
```

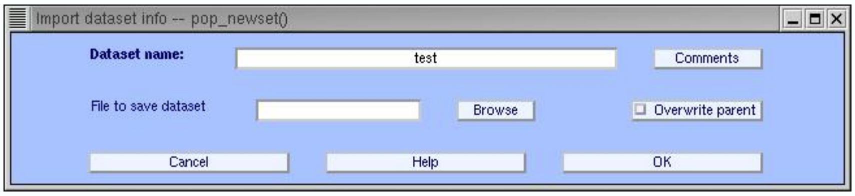
To import these data, select the menu item **File > Import data > Read ascii/float data file or array**. Set the sampling frequency to 256 Hz, press "**OK**". Other dataset parameters will be automatically adjusted.



EEGLAB dataset name (optional):	test	
Data file or array Matlab variable	eegdata	Browse
Number of channels (0->set from data):	0	
Time points per epoch (0=continuous data):	0	
Data sampling rate (Hz):	256	
Optional epoch start time for data epochs (sec):	0	
Channel locations file or array: Help		Browse
ICA weights array or text file (if any):		Browse
ICA sphere array or text file (if any):		Browse

Note on importing data from other file formats: To import continuous data from a **Matlab .mat file** instead of from a Matlab array, scroll the list of choices in the box above that shows " **Matlab array** ". **Note:** When reading a Matlab *.mat* file, EEGLAB assumes it contains only one Matlab variable. For reading a **(32−bit) binary float−format data file**, two choices are available: '**float le**' ("little−endian") and '**float be**' ("big−endian") The correct choice here depends on operating system. In case the bit ordering is unknown, try each of them. Note that the toolbox command line function **shortread()** can also be used to read data from a **(16−bit) short−integer file**. The resulting Matlab array may then be imported into EEGLAB as shown above.

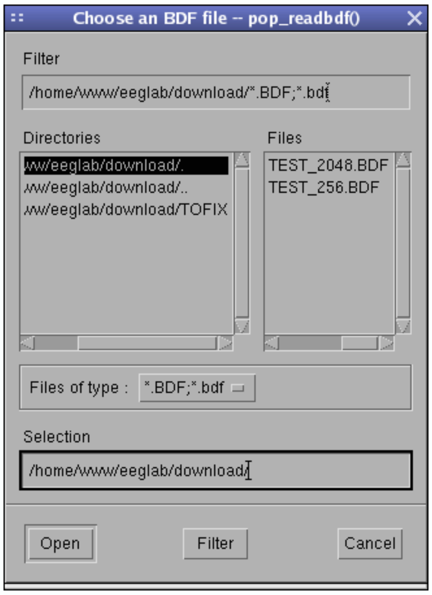
Now is a good time to add some "**Comments**" about the dataset. A window will pop up containing a text box for this purpose. Enter as much information about the dataset as possible. The information you enter will be saved with the dataset for future reference by you or by others who may load the data. You can add to or revise these comments later by selecting menu item "**Edit > Dataset info**". It is also possible to save the newly created data into a new dataset, either retaining or overwriting (in Matlab process memory) the old dataset. To also save the new dataset (with all its accompanying information fields) to disk, enter a filename in the lower edit field. Press "**OK**" to accept.



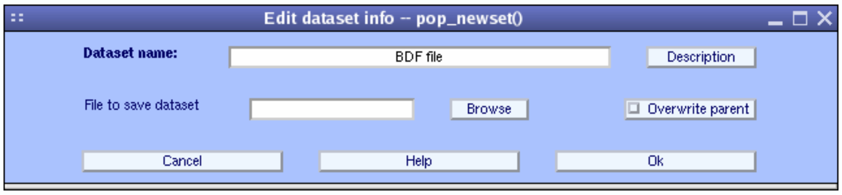
Then use menu item **Plot > EEG data (scroll)** to visualize the imported data.

II.1.2. Importing Biosemi .BDF files

Biosemi has extended the 16−bit European standard "EDF" (European Data Format) to their 24−bit data format, BDF (Biosemi Data Format). Select menu item **File > Import data > Read Biosemi .BDF file** (calling function **pop_readbdf()**). A window will pop up to ask for a file name.



Press **OPEN** to import a file.



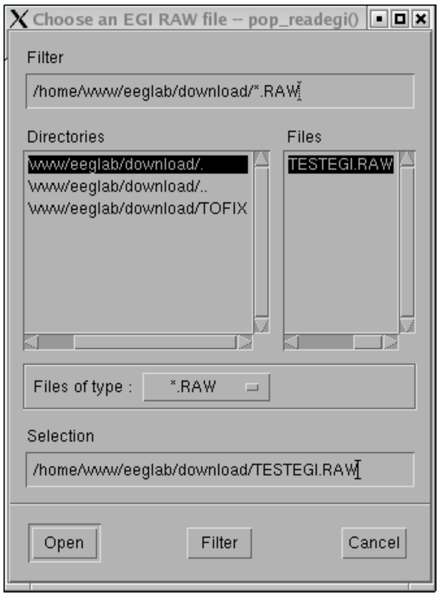
Press **OK**, then select menu item **Plot > EEG data (scroll)** to inspect the data. A sample .BDF file is available −− **TEST_256.BDF** (use "save link as" in your browser). (More sample files and reading functions are also available from the Biosemi ftp site). To extract event records from .BDF data, select menu item **File > Import event info > From data channel** as explained elsewhere in this tutorial.

II.1.3. Importing European data format .EDF files

To import data collected in the joint European 16−bit data format (EDF), select menu item **File > Import data > Read European data format .EDF file** (calling function pop_readedf()). A window will pop up to ask for a file name. Then select menu item **Plot > EEG data (scroll)** to inspect the data. A sample .EDF file is available −− TEST.EDF (use "save link as" in your browser). To extract event records from .EDF data, select menu item **File > Import event info > From data channel** as explained elsewhere in this tutorial.

II.1.4. Importing EGI .RAW continuous files

To read EGI (Electrical Geodesics Incorporated) .RAW data files, select menu item **File > Import data > Read EGI .RAW file**. The following window will appear


Choose an EGI RAW file -- pop_readegi()

Filter
/home/www/eeglab/download/*.RAW

Directories
www/eeglab/download/.
www/eeglab/download/..
www/eeglab/download/TOFIX

Files
TESTEGI.RAW

Files of type : *.RAW

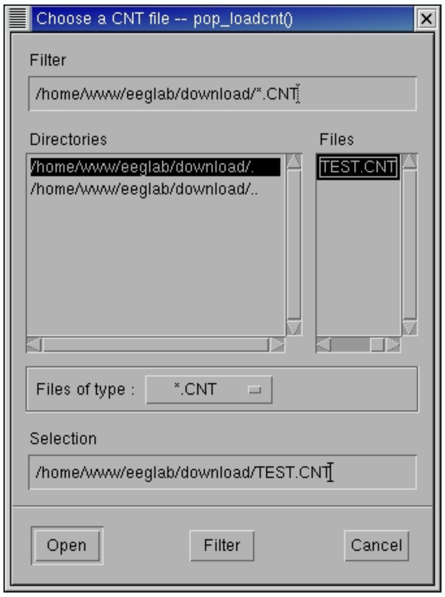
Selection
/home/www/eeglab/download/TESTEGI.RAW

The function pop_readegi() should be able to read EGI Version 2 and Version 3 data files. The presence of events in the EGI format is recorded in an additional EGI data channel. Information from this channel is automatically imported into the EEGLAB event table and this channel is then removed by EEGLAB from the EEGLAB data. (If, for any reason this fails to occur, extract events using menu item **File > Import event info > From data channel** as explained elsewhere in this tutorial.)

II.1.5. Importing Neuroscan .CNT continuous files

Note: In our experience, importing Neuroscan files is not an easy matter and no universal read function may exist. It seems that the EEGLAB function works properly in most cases, even for recent (2002) data files. For more about how to read Neuroscan files under Matlab, see a helper page. A stand−alone Matlab function for reading this format is also available on the function index page as **loadcnt()** (simply save the source code). You may import the EEG test file **TEST.CNT** and use it to test the following tutorial steps.

Start by selecting the menu item **File > Import data > Read .CNT data file**, which calls the **pop_loadcnt()** function. The following window will appear:


Choose a CNT file -- pop_loadcnt()

Filter: /home/www/eeglab/download/*.CNT

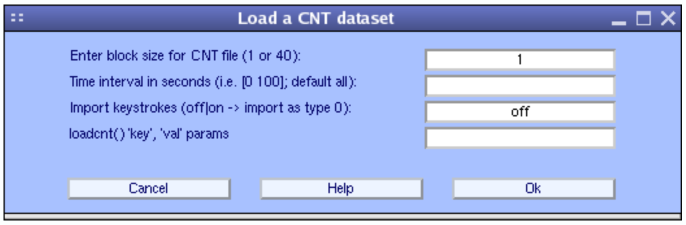
Directories:
/home/www/eeglab/download/.
/home/www/eeglab/download/..

Files: TEST.CNT

Files of type : *.CNT

Selection: /home/www/eeglab/download/TEST.CNT

Select the file to input (after changing the filter to the correct directory if necessary), and press "**OPEN**". The following window will pop up:

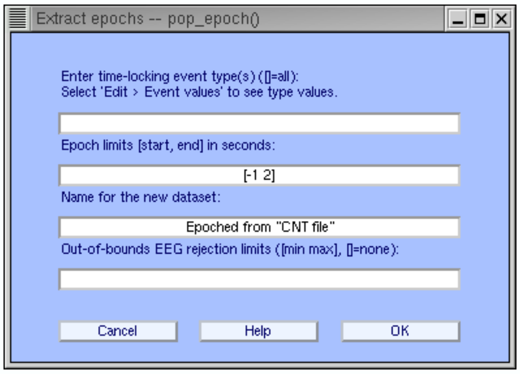

Load a CNT dataset

Enter block size for CNT file (1 or 40):	1
Time interval in seconds (i.e. [0 100]; default all):	
Import keystrokes (off|on -> import as type 0):	off
loadcnt() 'key', 'val' params	

The first input field concerns the structure of the .CNT file. If the imported data don't look like continuous EEG, try changing this number. Most often it should be 1 or 40, but other values may work. Now press "**OK**".

Next, use menu item **Plot > EEG data (scroll)** to inspect the input data and menu item **Edit > Event values** to inspect the input event field latencies.

We will now illustrate how to import additional epoch event information contained in an accompanying Neuroscan .DAT file into the EEGLAB dataset. Before importing the .DAT file, you must first epoch the loaded data (i.e., you must separate it into data epochs). To epoch the data, select **Tools > Extract epochs**


Extract epochs -- pop_epoch()

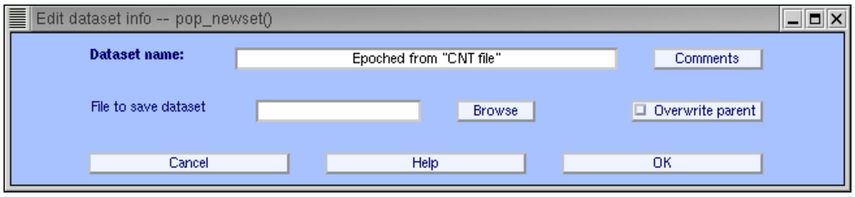
Enter time-locking event type(s) ([]=all):
Select 'Edit > Event values' to see type values.

Epoch limits [start, end] in seconds:
[-1 2]

Name for the new dataset:
Epoched from "CNT file"

Out-of-bounds EEG rejection limits ([min max], []=none):

Simply press "**OK**" to epoch the data based on the events contained in the .CNT file (here using a time window extending from −1 s before to 2 s after events). The following window will appear:


Edit dataset info -- pop_newset()

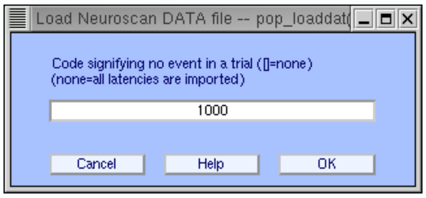
Dataset name: Epoched from "CNT file"

File to save dataset

Use this window to enter comments and save the new dataset. For computers with limited memory (RAM), try overwriting the parent dataset (here, the continuous dataset we have just imported) by checking the "**Overwrite parent**" box in this window. One may also save the dataset to disk. Press "**OK**" when done. Data epochs have now been extracted from the EEG data.

II.1.6. Importing Neuroscan .DAT information files

To import the .DAT file linked to a previously loaded .CNT file, select menu item **File > Import epoch info > Import .DAT info file** (calling function pop_loaddat()). The sample .DAT file associated with the continuous .CNT file we used above is available for download −− TEST.DAT Select the file to import in the resulting window. A second window will then appear:


Load Neuroscan DATA file -- pop_loaddat(

Code signifying no event in a trial ([]=none)
(none=all latencies are imported)
1000

In .DAT files, there must be a reaction time (in milliseconds) for each epoch. However, depending on experiment design there may be no reaction time in a given epoch. Then one has to use a code value for reaction time latencies in these epochs. For instance, you might decide that a value of "1000" (ms) would indicate that the subject did not respond. (If all the epochs of the experiment already have a reaction time, do not enter anything here.)

II.1.7. Importing Snapmaster .SMA files

Select menu item **File > Import data > Read Snapmaster .SMA file** (calling function **pop_snapread()**). A window will pop up to ask for a file name. Then select menu item **Plot > EEG data (scroll)** to inspect the data and item **Edit > Event values** to inspect event latencies and types. A sample .SMA data file is available −− **TEST.SMA** (use "save link as" in your browser).

II.1.8. Importing ERPSS .RAW or .RDF data files

To read ERPSS files (Event−Related Potential Software System, JS Hansen, Event−Related Potential Laboratory, University of California San Diego, La Jolla, CA, 1993), select menu item **File > Import data > Read ERPSS .RAW or .RDF file** (calling function **pop_read_erpss()**). A window will pop up to ask for a file name. Then select menu item **Plot > EEG data (scroll)** to inspect the data and item **Edit > Event values** to inspect event latencies and types. A sample .RDF data file is available −− **TEST.RDF** (use "Save link as" in your browser). A header file for the ERPSS format is also available here.

II.1.9. Importing data in other data formats

The home page of **Alois Schlolg** contains links to functions to read other EEG data formats under Matlab (see also **biosig.sf.net**). We are also willing to add other data importing functions to EEGLAB. Please send us a sample raw data file together with a Matlab function to read it and a README file describing the origin of the data and its format. We will attempt to include such functions in future releases of EEGLAB. Contributing authors will retain all rights to the functions they submit, and the authors' name(s) will be displayed in the function header. See our page on how to **contribute** to EEGLAB.

II.2. Importing event information for a continuous EEG dataset

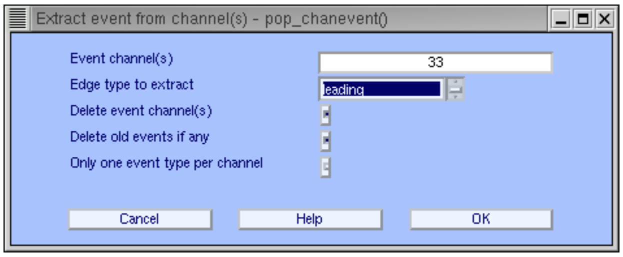
II.2.1. Importing events from a data channel

Often, information about experimental events are recorded onto one of the rows (channels) of the EEG data matrix. Once more we create simulated data to illustrate how to import events from a data channel. Assuming an EEG dataset with 33 rows (channels), out of which the first 32 are channels and the last (33) is an event channel with values 1 (stimulus onset), 2 (subject response), and 0 (other), Matlab code for generating such data follows (to test, copy and paste the code to the Matlab command line):

```
>> eegdata = rand(32, 256*100); % 32 channels of random activity (100 s sampled at 256 Hz).
>> eegdata(33,[10:256:256*100]) = 1; % simulating a stimulus onset every second
>> eegdata(33,[100:256:256*100]+round(rand*128)) = 2; % simulating reaction times about 500 ms after stimulus onsets
```

After copying the code above to Matlab and importing the array "eegdata" into EEGLAB as a test dataset (see **import a Matlab array** in this section), select menu item **File > Import event info >**

Import events from data channel to call function pop_chanevent() .


Extract event from channel(s) - pop_chanevent()

Event channel(s)	33
Edge type to extract	leading
Delete event channel(s)	
Delete old events if any	
Only one event type per channel	

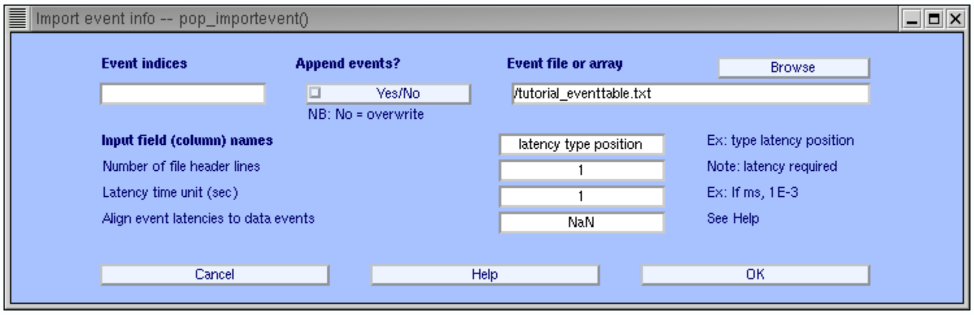
Enter "**33**" as the event channel and set the edge−extract type to "**leading**" (**Note:** place the mouse over the text " **Edge type to extract**" to see contextual help). Press "**OK**". Now, the event information will have been imported into the test EEGLAB dataset. At the same time, channel 33 will have been deleted from the test data. Select menu item **Edit > Event values** to inspect the imported event types and latencies.

II.2.2. Importing events from a Matlab array or text file

Using the random EEG dataset created above, we import event information stored in an ASCII text file, tutorial_eventtable.txt. This text file is composed of three columns, the first containing the latency of the event (in seconds), the second the type of the event, and the third a parameter describing the event (for example, the position of the stimulus). For example, the top lines of such a file might be:

Latency	Type	Position
1	target	1
2.3047	response	1
3	target	2
4.7707	response	2
5	target	1
6.5979	response	1
...		

Select menu item **File > Import event info > Import Matlab array or ASCII file**



Import event info -- pop_importevent()

Event indices	Append events?	Event file or array	Browse
	☐ Yes/No NB: No = overwrite	/tutorial_eventtable.txt	

Input field (column) names	latency type position	Ex: type latency position
Number of file header lines	1	Note: latency required
Latency time unit (sec)	1	Ex: If ms, 1E-3
Align event latencies to data events	NaN	See Help

Browse for the tutorial text file, set the number of header lines to 1 (for the first line of the file, which gives the column field names) and set the input fields (i.e., the names associated with the columns in the array) to "**latency type position**". If these field names are quoted or separated by commas, these extra characters are ignored. **NOTE:** It is **NECESSARY** to use the names "**latency**" and "**type**" for two of the fields. These two field names are used by EEGLAB to extract, sort and display events. These fields must be **lowercase** since Matlab is case sensitive.

In this interactive window the input "**Event indices**" and checkbox "**Append events?**" can be used to insert new events or replace a subset of events with new events (for instance for large EEG files which may have several event files).

Important note about aligning events

An essential input above is "**Align event latencies to data events**" which aligns the first event latency to the existing event latency and checks latency consistency. A value of **NaN** (Matlab for not−a−number) indicates that this option is ignored (as in the example above). However, for most EEG data, the EEG is recorded with basic events stored in an event channel (see Import events from a data channel above) for instance. Detailed event information is recorded separately in a text file: as a consequence the events recorded in the text file have to be aligned with the events recorded in the EEG.

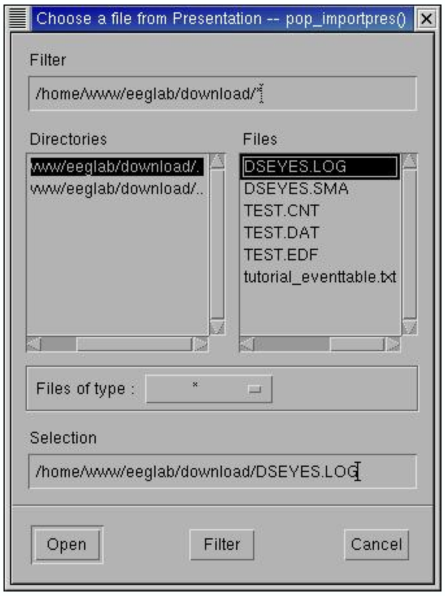
To do so, set the input for "**Align event latencies to data events** " to 0 if the first event in the text file correspond to the first event recorded in the EEG (i.e., if the offset between the two is 0). Setting this value to 1 indicates that event 1 in the event text file corresponds to event number 2 in the EEG data. Here, negative values can also be used to indicate that events in text file start before those recorded in the EEG).

When aligning events, as shown in the following section, the function displays the latencies of the two event types, so the user can check that they are aligned based on his knowledge of the experiment (for instance, there may be more events in the text file than recorded in the EEG).

II.2.3. Importing events from a Presentation file

EEGLAB can also import events saved using the Presentation software. Sample files are available for download here: TEST.SMA (a SnapMaster .SMA file) and TEST.LOG (a Presentation event log file).

To test the use of these sample files, first import the .SMA data file using menu item **File > Import data > Read .SMA data file**. Then select **File > Import event info > Import events from a Presentation file** to import events from the Presentation log file as shown below



Matlab returns

```
Reading file (lines): 7
Check alignment between pre−existing (old) and loaded event latencies:
Old event latencies (10 first):  10789  21315  31375  41902  51962  62489 ...
New event latencies (10 first): 10789  21315  31376  41902  51962  62489 ...
Done.
```

After aligning the first event latency recorded in the Presentation file to the first event latency recorded in the EEG in the SnapMaster file, check that the events recorded in the SnapMaster file have the same latencies as the ones recorded in the .LOG presentation file. Note that if the events are shifted, it is always possible to suppress events manually or to import the presentation file as a text file, as described in the previous section. Note that some Presentation files that contain comments at the end of the file may not be supported. If you are not able to import a Presentation file, try removing any comments from the end of the file. If it still does not work, try importing the Presentation file as a text file as described in the previous section.

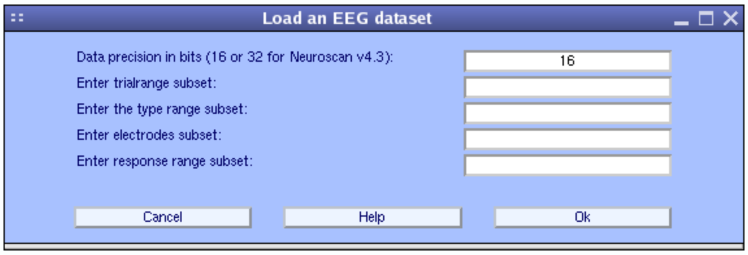
II.3. Importing sets of single−trial EEG epochs into EEGLAB

II.3.1. Importing .RAW EGI data epoch files

Select **File > Import data > Read EGI .RAW file**. A window will pop up to ask for the file name. Then select menu item **Plot > EEG data (scroll)** to inspect the imported data.

II.3.2. Importing Neuroscan .EEG data epoch files

Select **File > Import data > Read .EEG data file**, calling function pop_loadeeg(). A first window will pop up to ask for the file name and a second window (below) will query for data format (16 or 32 bits) and for data range. See the pop_loadeeg() function for more details.



Load an EEG dataset

Data precision in bits (16 or 32 for Neuroscan v4.3):	16
Enter trialrange subset:	
Enter the type range subset:	
Enter electrodes subset:	
Enter response range subset:	

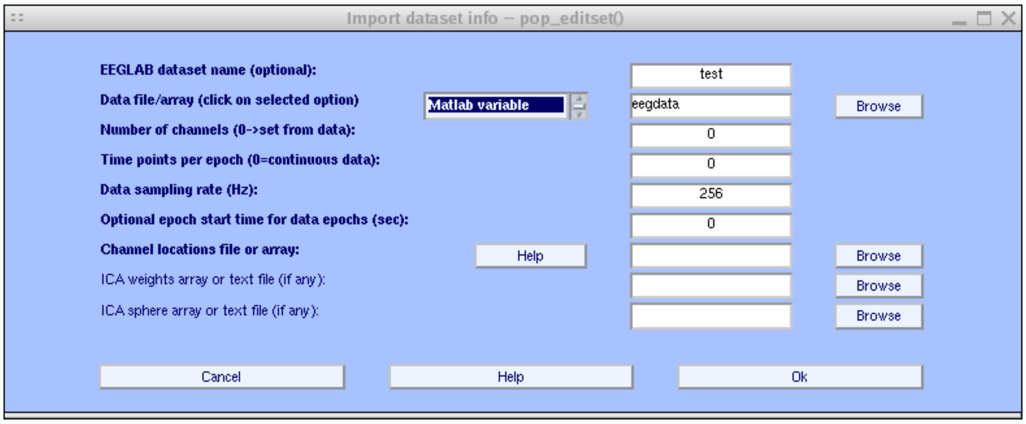
Then select menu item **Plot > EEG data (scroll)** to inspect the imported data. In this case, epoch events have also been imported from the file and can be visualized using menu item **Edit > Event values**.

II.3.3. Importing epoch info Matlab array or text file into EEGLAB

Importing epoch information means that data epochs have already been extracted from the continuous EEG data, and that the Matlab array or text epoch information file has one entry per epoch. To illustrate how to import such a file or array, we will once more create some simulated EEG data.

>> eegdata = rand(32, 256, 10); % 32 channels, 256 time points per epoch, 10 epochs

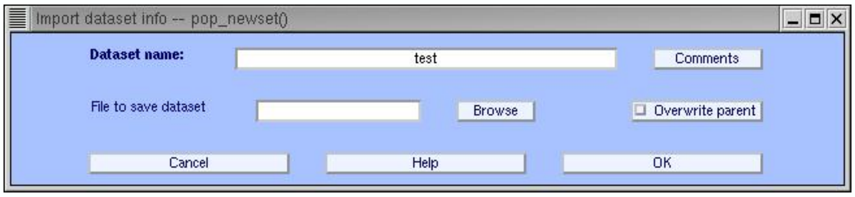
Select menu item **File > Import data > Read ascii/float data file or Matlab array**.



Import dataset info -- pop_editset()

EEGLAB dataset name (optional):		test
Data file/array (click on selected option)	Matlab variable	eegdata
Number of channels (0->set from data):		0
Time points per epoch (0=continuous data):		0
Data sampling rate (Hz):		256
Optional epoch start time for data epochs (sec):		0
Channel locations file or array:		
ICA weights array or text file (if any):		
ICA sphere array or text file (if any):		

Press "**OK**" in this window (**Note:** If you select a data importing format different from "Matlab variable", be sure to click on it to actually select the option). Then, a second window will pop up.



Dataset name: test

File to save dataset:

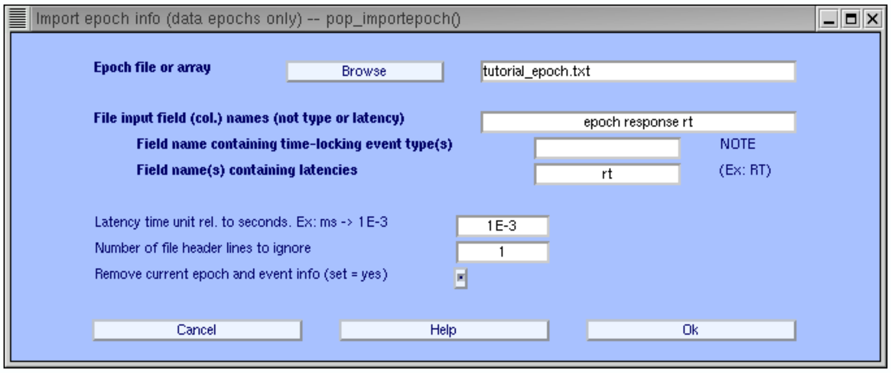
☐ Overwrite parent

Press "**OK**" in this window (see **importing a Matlab array** at the beginning of this page for more information). Note that the Matlab array, being 3−D, is automatically imported as data epochs: the first dimension is interpreted as data channels, the second as data points and the third as data epochs or trials (e.g., our sample data matrix above contains 10 epochs).

Let us imagine that our simulated EEG data came from a simple stimulus/response task, with subject responses being either "correct" or "wrong" and response latencies recorded in milliseconds. Then the epoch event file might look something like this:

Epoch	Response	Response_latency
1	Correct	502
2	Correct	477
3	Correct	553
4	Correct	612
5	Wrong	430
6	Correct	525
7	Correct	498
8	Correct	601
9	Correct	398
10	Correct	573

This file **tutorial_epoch.txt** can be downloaded or copied from the array above in a text file. Then select menu item **File > Import epoch info > Import Matlab array or ascii file**, bringing up the following window:



Epoch file or array: tutorial_epoch.txt

File input field (col.) names (not type or latency): epoch response rt

Field name containing time-locking event type(s):

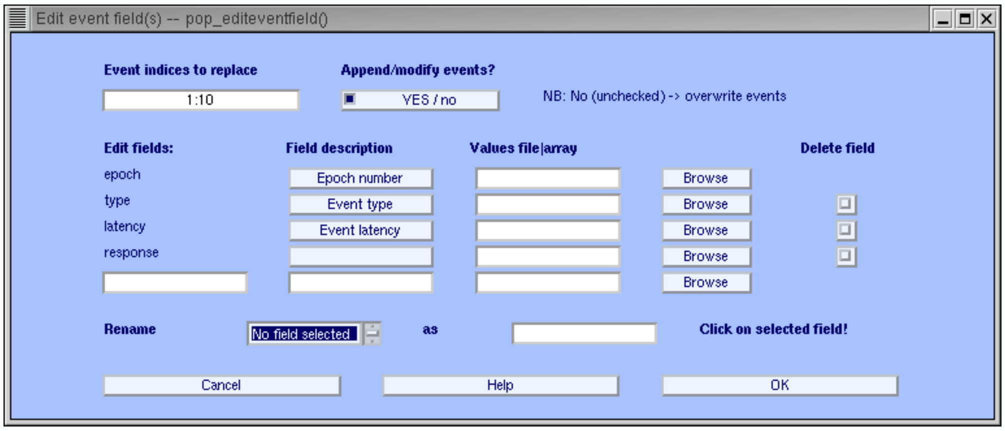
Field name(s) containing latencies: rt

Latency time unit rel. to seconds. Ex: ms -> 1E-3: 1E-3

Number of file header lines to ignore: 1

Remove current epoch and event info (set = yes): ☒

Above, browse for the "**tutorial_epoch.txt**" file, set the input fields to "**epoch response rt**" (where rt is an acronym for "reaction time"). The only one of these fields that contains latency information is "**rt**", so it is entered as input to the next query field. This file (see above) has 1 header line, as we need to specify in the "**File header line box**". Finally the reaction times are recorded in milliseconds, which we indicate as "**1E−3**" (i.e., one−thousandth of a second). Note that the last entry, "**Clean previous events**", allows the user to import other information later if it is unset. Press "**OK**" when done. Now select **Edit > Event fields**.



Edit event field(s) -- pop_editeventfield()

Event indices to replace: 1:10
Append/modify events?: YES / no
NB: No (unchecked) -> overwrite events

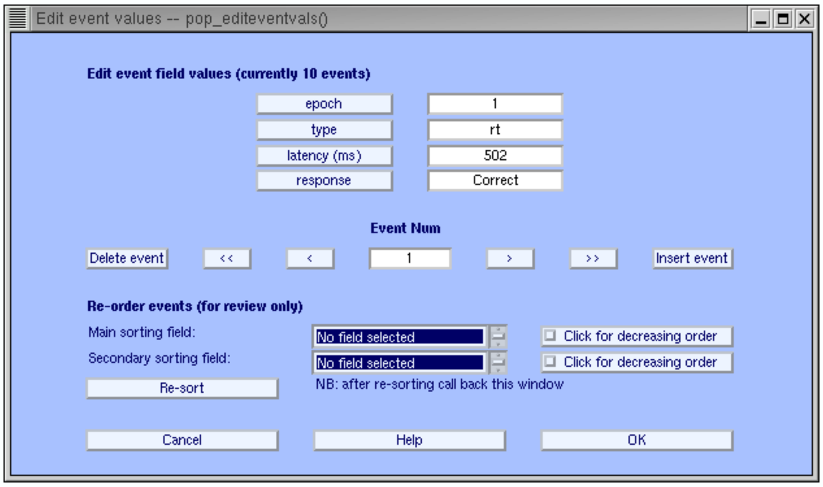
Edit fields:	Field description	Values file|array		Delete field
epoch	Epoch number		Browse	
type	Event type		Browse	☐
latency	Event latency		Browse	☐
response			Browse	☐
			Browse	

Rename No field selected **as** **Click on selected field!**

Cancel | Help | OK

It is a good idea to click each of the "**Field description**" buttons above to add detailed descriptions of the meaning of each of the fields and the coding of the field values (for example: 1 = correct, 2 = wrong, etc.). This information is stored along with the event field values when the dataset is saved (very useful for later analysis by you or others!).

Above, there are now four fields, not three as for the file data. Also note that the field "**rt**" is not present. All of this is normal because EEGLAB does not store epoch information as such, but converts it into fields in its events structure. Though this seems a bit complicated in the beginning, it helps avoid redundancy and inconsistencies between epoch and event information. It also means that new data epochs can be re−extracted from data epochs based on the stored events. Now select menu item **Edit > Event values** to inspect what happened to the reaction time information:



Edit event values -- pop_editeventvals()

Edit event field values (currently 10 events)

epoch	1
type	rt
latency (ms)	502
response	Correct

Event Num

Delete event | << | < | 1 | > | >> | Insert event

Re-order events (for review only)

Main sorting field: No field selected ☐ Click for decreasing order
Secondary sorting field: No field selected ☐ Click for decreasing order
Re-sort
NB: after re-sorting call back this window

Cancel | Help | OK

As shown above, when epoch information was imported, events with type named **rt** were created and assigned a latency. If we had had several columns containing latency information, the function would have created several types. (**Programming note:** For convenience, standard epoch information is available from the command line in the variable **EEG.epoch**. Also, event information available in **EEG.event** can be used for script or command line data processing. See the **script writing tutorial** for more information.)

II.4. Importing sets of data averages into EEGLAB

EEGLAB was made to process and visualize single−trial data. Event−related potential (ERP) averages can also be processed and visualized, but they should not be imported directly. Note that in our current practice, we perform averaging over trials **after** applying ICA and not before (see Makeig et al. Science, 2002).

However, an increasing number of ERP researchers find it of interest to apply ICA to related **sets** of ERP grand averages (see Makeig et al, J. Neuroscience, 1999 and Makeig et al., Royal Society, 1999). To import data grand−average epochs into EEGLAB, stack the different conditions in a single array as explained below.

II.4.1. Importing data into Matlab

First, the data averages for different conditions must be imported to Matlab. For example, one may export these averages in text format and then use the standard Matlab function

```
>> load −ascii filename.txt
```

Note that to import ASCII files to Matlab, all column names and row names must be removed. For Neuroscan user, we have also programmed the function **loadavg()** which can read most binary .AVG Neuroscan files.

II.4.2. Concatenating data averages

For example, from a three−condition experiment, we may derive three ERP averages with a sampling rate of 1000 Hz, covering from −100 to 600 ms with respect to stimulus onsets (Note that we always process each subject individually and then compare the results across subjects at the end of the analysis).

For instance typing **>> whos** under Matlab might return

```
  Name        Size        Bytes  Class

  avgcond1     31x600       148800  double array
  avgcond2     31x600       148800  double array
  avgcond3     31x600       148800  double array

Grand total is 55800 elements using 446400 bytes
```

Note: If necessary, transpose the arrays using

```
>>avgcond1 = avgcond1';
```

Then concatenate the arrays

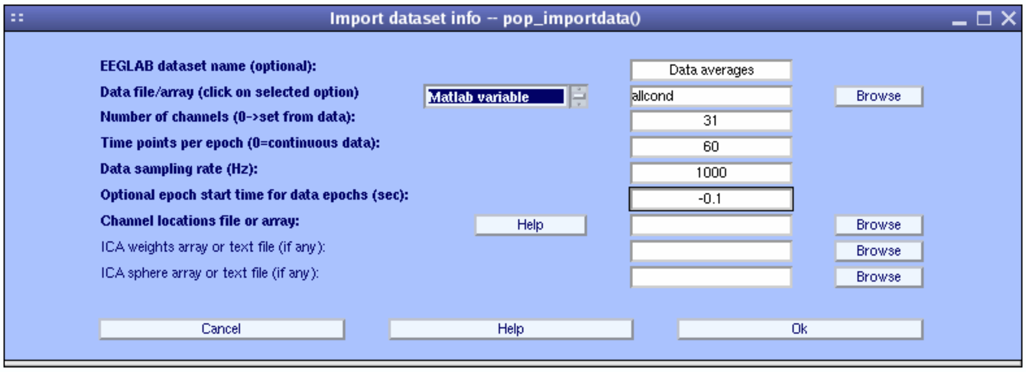
```
>> allconds = [ avgcond1 avgcond2 avgcond3 ] ;
```

II.4.3. Importing concatenated data averages in EEGLAB

Start Matlab, then start EEGLAB

```
>> eeglab
```

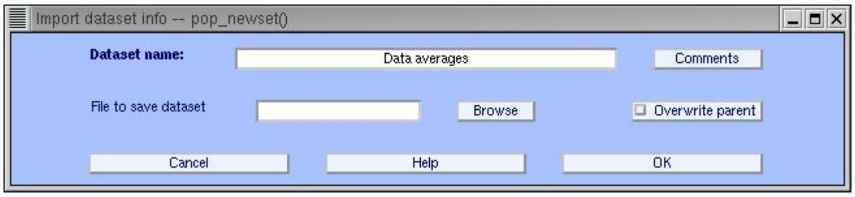
Select menu item **File > Importing data > Read ascii/float file or Matlab array**



Import dataset info -- pop_importdata()

EEGLAB dataset name (optional):		Data averages
Data file/array (click on selected option)	Matlab variable	allcond
Number of channels (0->set from data):		31
Time points per epoch (0=continuous data):		60
Data sampling rate (Hz):		1000
Optional epoch start time for data epochs (sec):		-0.1
Channel locations file or array:		
ICA weights array or text file (if any):		
ICA sphere array or text file (if any):		

Enter the information as indicated above. The following window pops up and allows you to add comments and/or save the new dataset immediately. Press **OK** to create a new dataset.

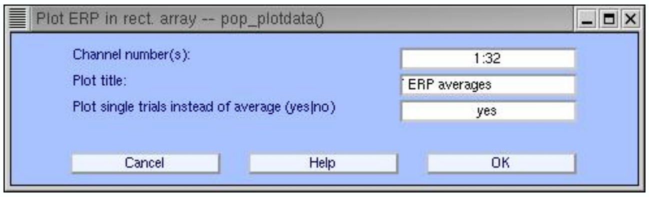


Import dataset info -- pop_newset()

Dataset name: Data averages

File to save dataset

Select **Plot > ERP plots> ERP in rect. array** and set the last option **Plot single trials** to **YES** to visualize the three condition ERPs.



Plot ERP in rect. array -- pop_plotdata()

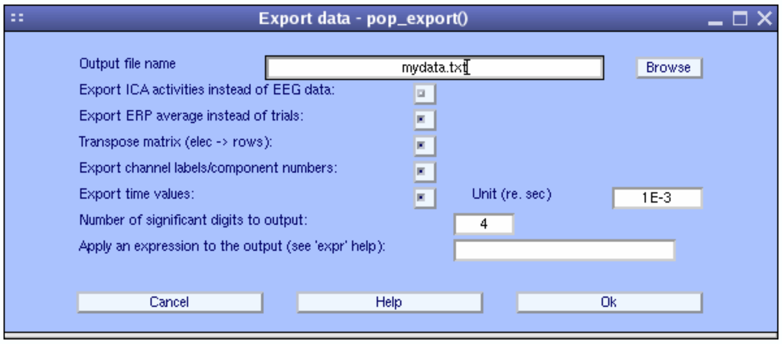
Channel number(s):	1:32
Plot title:	ERP averages
Plot single trials instead of average (yes|no)	yes

It is possible to process the three average−ERP epochs as if they were single−trial epochs (although in this case some EEGLAB functions may not be meaningful). See the **Data analysis tutorial** for more information.

II.5. Exporting data and ICA matrices

II.5.1. Exporting data to an ASCII text file

EEGLAB datasets can be exported as ASCII files using menu item **File > Exports> Data and ICA activity to text file**. Enter a file name (**mydata.txt**, for instance). Check the second checkbox to export the average ERP instead of the data epochs. By default, the electrode labels are saved for each row (4th check box) and the time values are saved for each column (5th checkbox). Time units can be specified in the edit box closest to the time values checkbox. Finally, check the third checkbox to transpose the matrix before saving.


Export data - pop_export()

Output file name	mydata.txt		Browse
Export ICA activities instead of EEG data:	☐		
Export ERP average instead of trials:	☑		
Transpose matrix (elec -> rows):	☑		
Export channel labels/component numbers:	☑		
Export time values:	☑	Unit (re. sec)	1E-3
Number of significant digits to output:	4		
Apply an expression to the output (see 'expr' help):			

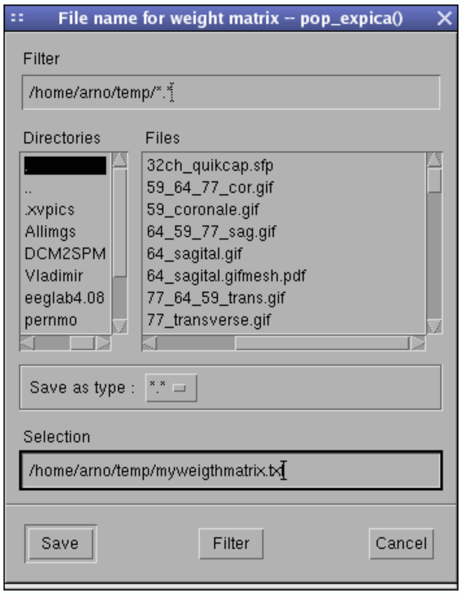
The file written to disk may look like this

```
                 FPz     EOG1    F3      Fz      F4      EOG2    FC5     FC1     ...
-1000.0000       -1.1091 2.0509  0.1600  -0.1632 -0.4848 -1.3799 -0.0254 -0.4788 ...
-992.1880        0.6599  1.7894  0.3616  0.6354  0.8656  -2.9291 -0.0486 -0.4564 ...
-984.3761        1.8912  1.3653  -0.6887 -0.0437 0.2176  -0.9767 -0.6973 -1.5856 ...
-976.5641        0.5129  -0.5399 -1.4076 -1.2616 -0.8667 -3.5189 -1.5411 -1.9671 ...
-968.7521        -0.0322 -0.4172 -0.9411 -0.6027 -0.9955 -2.3535 -1.6068 -1.0640 ...
-960.9402        0.1491  0.0898  -0.0828 0.3378  0.0312  -2.4982 -0.9694 -0.0787 ...
-953.1282        -1.9682 -1.5161 -1.2022 -0.8192 -1.1344 -3.3198 -1.6298 -0.9119 ...
-945.3162        -3.7540 -2.1106 -2.6597 -2.4203 -2.2365 -3.5267 -1.9910 -2.7470 ...
-937.5043        -2.4367 -0.1690 -0.9962 -1.7021 -2.8847 -2.1883 -0.2790 -1.5198 ...
-929.6923        -2.8487 -0.3114 -1.6495 -2.6636 -4.0906 -1.7708 -1.2317 -2.3702 ...
-921.8803        -2.8535 0.1097  -1.5272 -2.0674 -3.8161 -3.1058 -0.8083 -1.5088 ...
-914.0684        -3.9531 -0.4527 -1.8168 -2.2164 -3.4805 -2.1490 -1.0269 -1.3791 ...
-906.2564        -3.9945 -0.1054 -1.8921 -2.8029 -3.5642 -3.4692 -1.1435 -2.2091 ...
-898.4444        -4.4166 -0.8894 -3.3775 -3.8089 -3.8068 -1.7808 -2.5074 -3.5267 ...
-890.6325        -3.0948 0.5812  -2.5386 -1.7772 -1.8601 -2.8900 -2.0421 -2.0238 ...
-882.8205        -3.1907 0.7619  -3.6440 -2.1976 -1.4996 -0.6483 -3.4281 -2.7645 ...
-875.0085        -1.7072 2.5182  -3.2136 -2.4219 -1.3372 -1.5834 -2.9586 -2.8805 ...
-867.1966        -1.8022 1.7044  -2.6813 -3.2165 -2.7036 0.0279  -2.5038 -3.4211 ...
-859.3846        -3.1016 -0.1176 -3.6396 -4.3637 -3.9712 -3.5499 -3.4217 -4.5840 ...
-851.5726        -1.7027 0.7413  -3.3635 -3.8541 -3.5940 -1.3157 -2.9060 -3.8355 ...
-843.7607        -0.2382 0.5779  -1.9576 -2.6630 -1.8187 -1.1834 -1.4307 -2.4980 ...
-835.9487        0.7006  0.4125  -0.4827 -1.7712 -2.0397 0.2534  0.2594  -1.2367 ...
-828.1367        -0.2056 -0.3509 0.4829  -0.6850 -1.1222 0.0394  1.4929  0.7069  ...
-820.3248        0.3797  -0.3791 0.9267  0.2139  -0.6116 -0.7612 1.3307  1.5108  ...
-812.5128        -0.8168 -1.4683 -0.3252 -0.8263 -1.5868 -0.7416 -0.2708 -0.1987 ...
-804.7009        -0.7432 -0.3581 -0.9168 -0.8350 -1.7733 -0.4928 -0.7747 -0.6256 ...
...
```

The first column contains the time axis and the other the data for each electrode. This file might, for example, be imported into SPSS or BMDP.

II.5.2. Exporting ICA weights and inverse weight matrices

Use menu item **File > Export> Weight matrix to text file** to export the ICA unmixing matrix (weights*sphere). Simply enter a file name in the pop−up window and press **Save**.


File name for weight matrix -- pop_expica()

Filter: /home/arno/temp/*.*

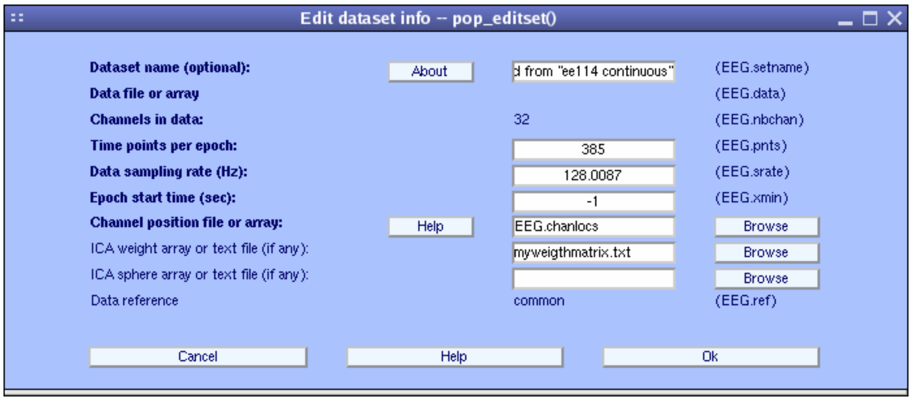
Directories: .. .xvpics Allimgs DCM2SPM Vladimir eeglab4.08 permno

Files: 32ch_quikcap.sfp 59_64_77_cor.gif 59_coronale.gif 64_59_77_sag.gif 64_sagital.gif 64_sagital.gifmesh.pdf 77_64_59_trans.gif 77_transverse.gif

Save as type : *.*

Selection: /home/arno/temp/myweigthmatrix.txt

The text file on disk then contains the weight matrix. It may be re−imported into another EEGLAB dataset using menu item **Edit > Dataset info**. As shown below, enter the filename in the **ICA weight array** edit box. Leave the sphere edit box empty, or empty it if it is not empty. See the ICA decomposition tutorial for more details on sphere and weight matrices.


Edit dataset info -- pop_editset()

Dataset name (optional):	About	d from "ee114 continuous"	(EEG.setname)
Data file or array			(EEG.data)
Channels in data:		32	(EEG.nbchan)
Time points per epoch:		385	(EEG.pnts)
Data sampling rate (Hz):		128.0087	(EEG.srate)
Epoch start time (sec):		-1	(EEG.xmin)
Channel position file or array:	Help	EEG.chanlocs	Browse
ICA weight array or text file (if any):		myweigthmatrix.txt	Browse
ICA sphere array or text file (if any):			Browse
Data reference		common	(EEG.ref)

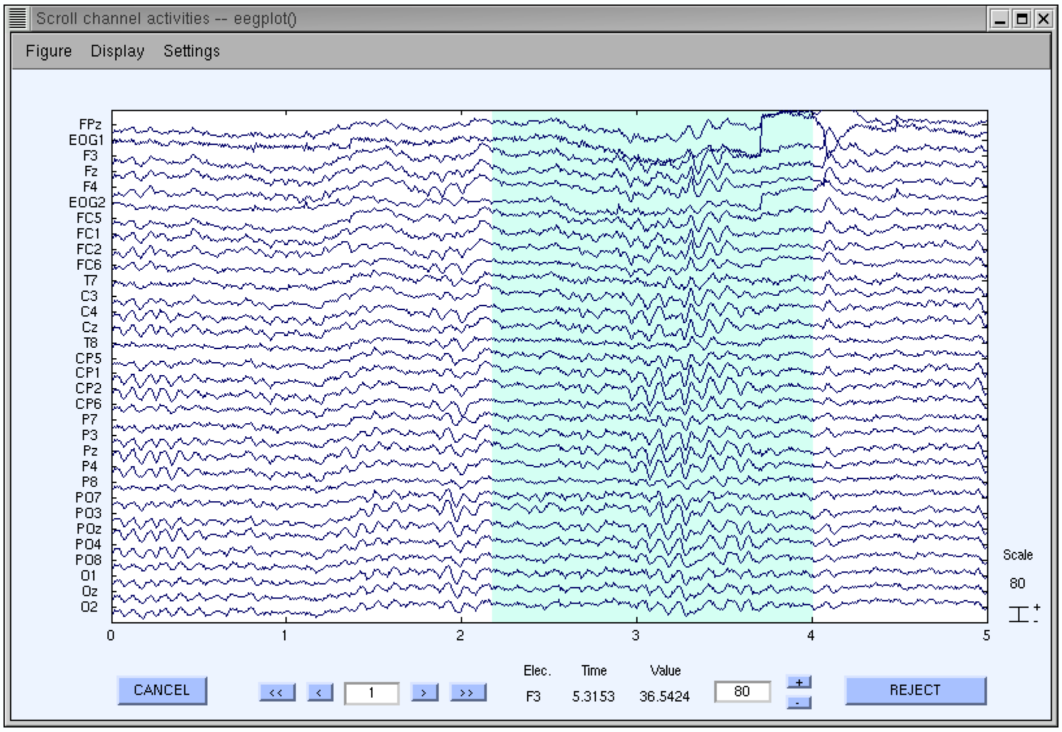
III. Rejecting artifacts in continuous and epoched data

Strategy: The approach used in EEGLAB for artifact rejection is to use **statistical** thresholding to **suggest** epochs to reject from analysis. Current computers are fast enough to allow easy confirmation and adjustment of suggested rejections by visual inspection, which our **eegplot()** tool makes convenient. We therefore favor **semi−automated rejection** coupled with visual inspection, as detailed below. In practice, we favor applying EEGLAB rejection methods to **independent components** of the data, using a seven−stage method outlined **below**.

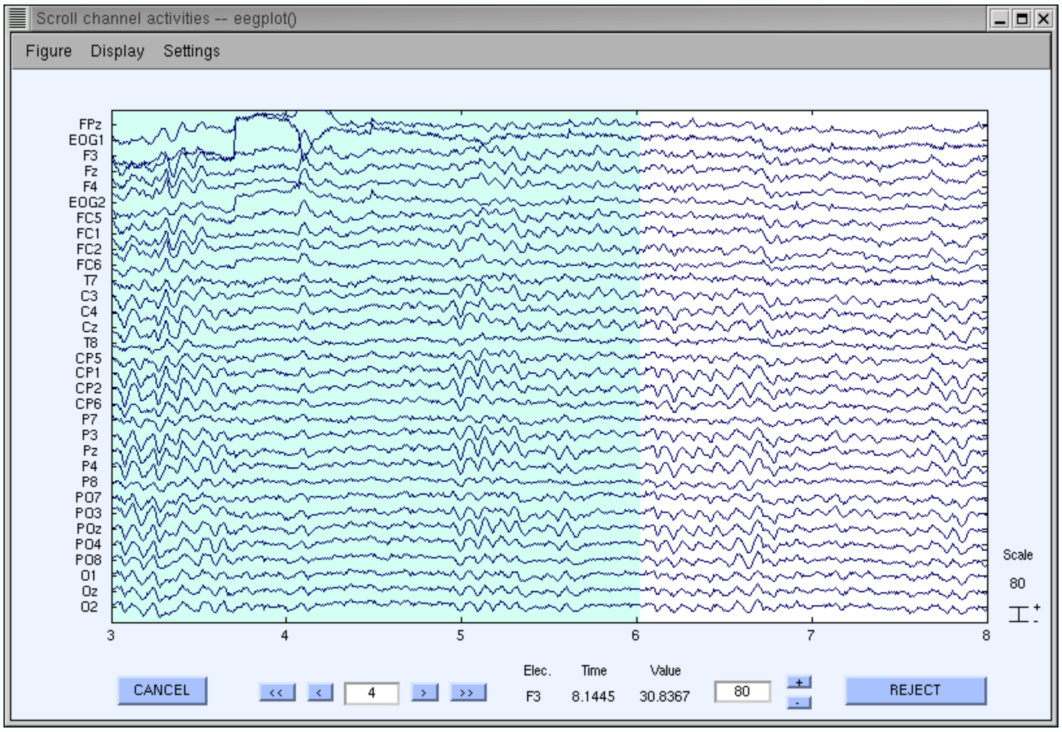
Examples: In all this section, we illustrate EEGLAB methods for artifact rejection using the same sample EEG dataset we used in the **main data analysis tutorial**. These data are not perfectly suited for illustrating artifact rejection since they have relatively few artifacts! Nevertheless, by setting low thresholds for artifact detection it is was possible to find and mark outlier trials. Though these may not necessarily be artifactual, we use them here to illustrate how the EEGLAB data rejection functions work. We encourage users to make their own determinations as to which data to reject and to analyze using the set of EEGLAB rejection tools.

III.1. Rejecting artifacts in continuous data

Rejecting portions of continuous data can only be performed visually using function **pop_eegplot()**. Select **Tools > Reject continuous data**. Select data for rejection by dragging the mouse over a data region. After marking some portions of the data for rejection, press "**REJECT**" and a new data set will be created with the rejected data omitted. EEGLAB will adjust the **EEG.event** structure fields and will insert "**rejection boundary**" events where data has been rejected. Thus, rejection on continuous data must be performed **before** separating it into data epochs. **Note:** To "de−select" a portion of the data, simply click on the selected region. This allows re−inspection of the data portions marked for rejection in two or more passes, e.g., after the user has developed a more consistent rejection strategy or threshold. See the section on **Visualizing EEG data** under the main tutorial for more information about how to use this interactive window.

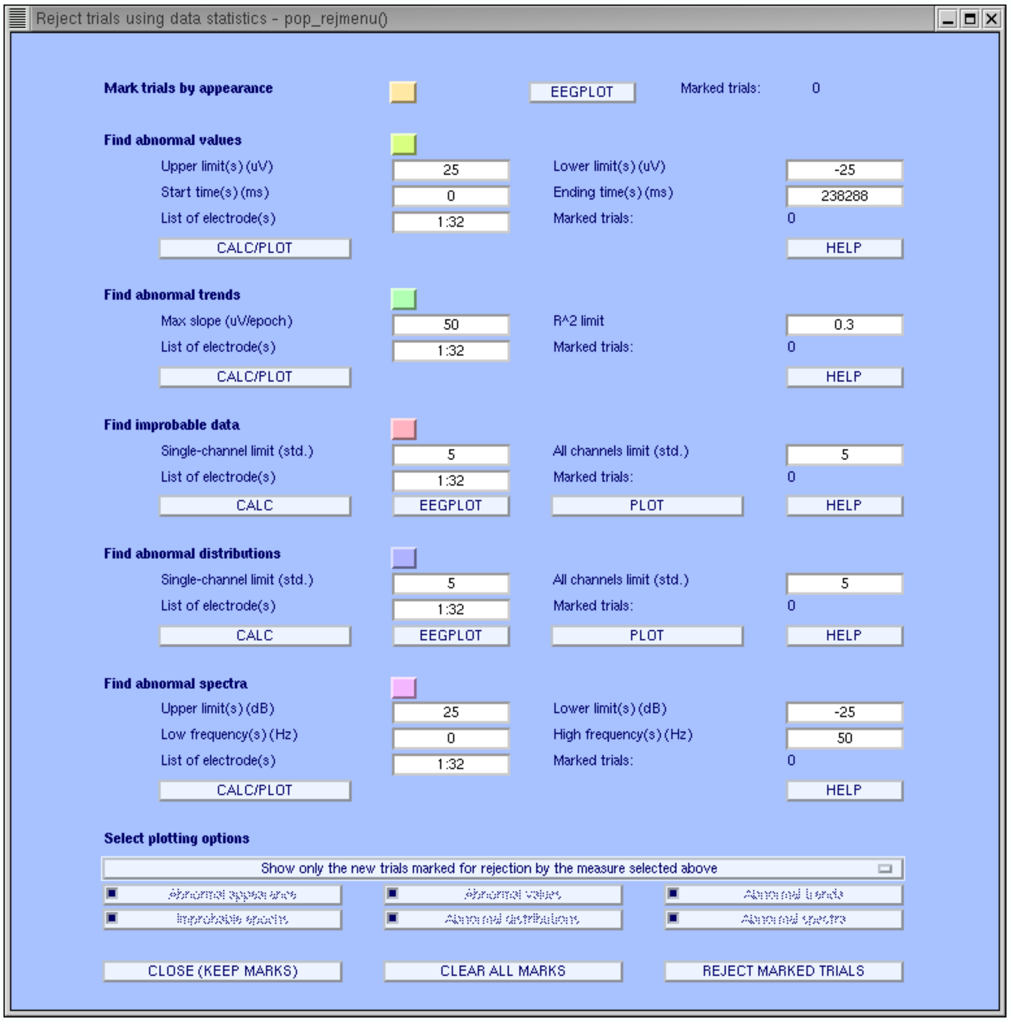


Note: to select portions of data that extend out of the plotting window, simply drag the mouse over the new region and connect it to a previously marked region. For instance, in the following plotting window which already had the time interval 2.2 seconds to 4 seconds selected (as show above), drag the mouse from 6 seconds back to 4.



III.2. Rejecting artifacts in epoched data

In contrast to continuous EEG data, we have developed several functions to find and mark for rejection those data epochs that appear to contain artifacts using their statistical distributions. To call the main window for epoched data rejection, select **Tools > Reject data epochs > Reject data (all methods)**. The window below will pop up. We will describe the fields of this window in top−to−bottom order. First, change the bottom multiple choice button reading "**Show all trials marked for rejection by the measure selected above or checked below**" to the first option, "**Show only the new trials marked for rejection by the measure selected above**". We will come back to the default option at the end.


Reject trials using data statistics - pop_rejmenu()

Mark trials by appearance | EEGPLOT | Marked trials: 0

Find abnormal values

Upper limit(s) (uV)	25	Lower limit(s) (uV)	-25
Start time(s) (ms)	0	Ending time(s) (ms)	238288
List of electrode(s)	1:32	Marked trials:	0
CALC/PLOT			HELP

Find abnormal trends

Max slope (uV/epoch)	50	R^2 limit	0.3
List of electrode(s)	1:32	Marked trials:	0
CALC/PLOT			HELP

Find improbable data

Single-channel limit (std.)	5	All channels limit (std.)	5
List of electrode(s)	1:32	Marked trials:	0
CALC	EEGPLOT	PLOT	HELP

Find abnormal distributions

Single-channel limit (std.)	5	All channels limit (std.)	5
List of electrode(s)	1:32	Marked trials:	0
CALC	EEGPLOT	PLOT	HELP

Find abnormal spectra

Upper limit(s) (dB)	25	Lower limit(s) (dB)	-25
Low frequency(s) (Hz)	0	High frequency(s) (Hz)	50
List of electrode(s)	1:32	Marked trials:	0
CALC/PLOT			HELP

Select plotting options

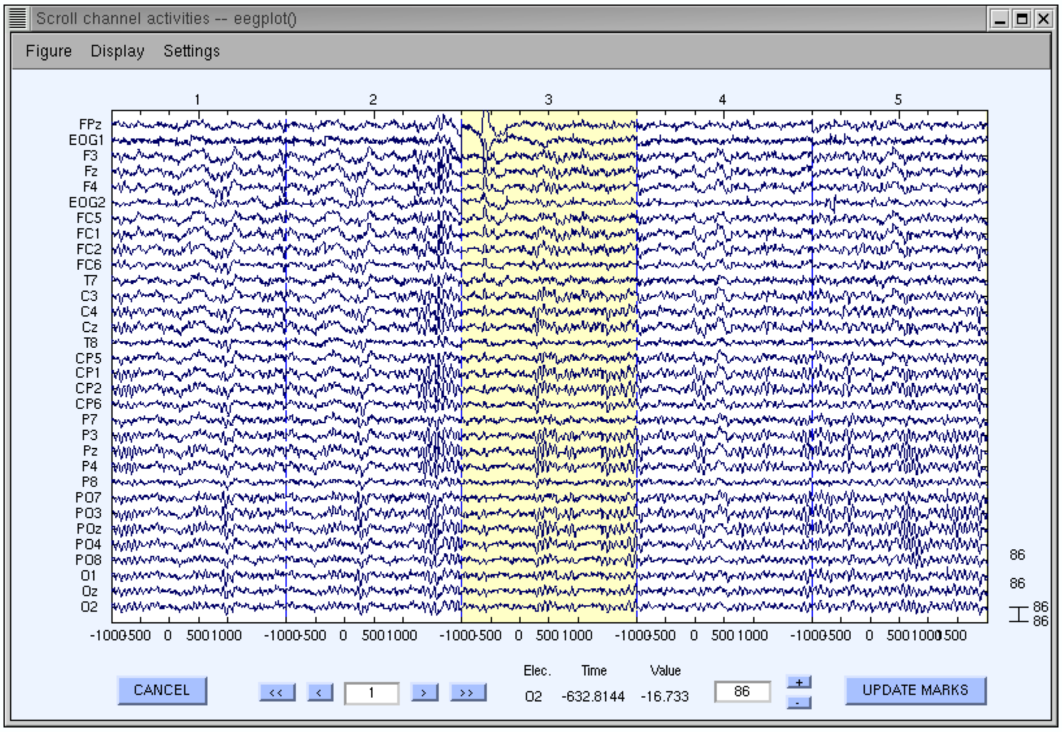
Show only the new trials marked for rejection by the measure selected above

Abnormal appearance	Abnormal values	Abnormal trends
Improbable epochs	Abnormal distributions	Abnormal spectra

CLOSE (KEEP MARKS) | CLEAR ALL MARKS | REJECT MARKED TRIALS

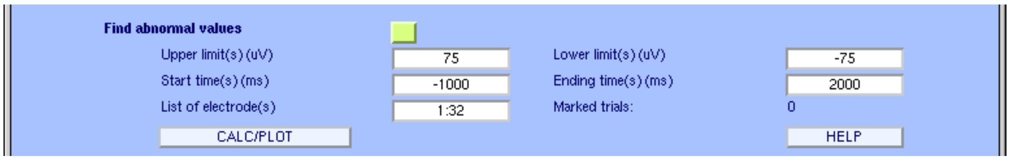
III.2.1. Rejecting epochs by visual inspection

As with continuous data it is possible to use EEGLAB to reject epoched data simply by visual inspection. To do so, press the "**EEGPLOT**" button on the top of the interactive window. A scrolling window will pop up. Epochs are shown delimited by blue dashed lines and can be selected/deselected for rejection simply by clinking on them. Rejecting parts of an epoch is not possible.


Scroll channel activities -- eegplot()
Figure Display Settings
FPz EOG1 F3 Fz F4 EOG2 FC5 FC1 FC2 FC6 T7 C3 C4 Cz T8 CP5 CP1 CP2 CP6 P7 P3 Pz P4 P8 PO7 PO3 POz PO4 PO8 O1 Oz O2
1 2 3 4 5
-1000-500 0 500 1000
CANCEL
UPDATE MARKS
Elec. Time Value
O2 -632.8144 -16.733
86

III.2.2. Rejecting extreme values

During "threshold rejection", the user sets up bounding values the data should not exceed. If the data (at the selected electrodes) exceeds the given limits during a trial, the trial is marked for rejection. Enter the following values under ***Threshold rejection*** (which calls function **pop_eegthresh()**), and press "**CALC/PLOT**". The parameters entered below tell EEGLAB that EEG values should not exceed +/−**75** µV in any of the 32 channels (**1:32**) at any time within the epoch (**−1000** to **2000** ms).

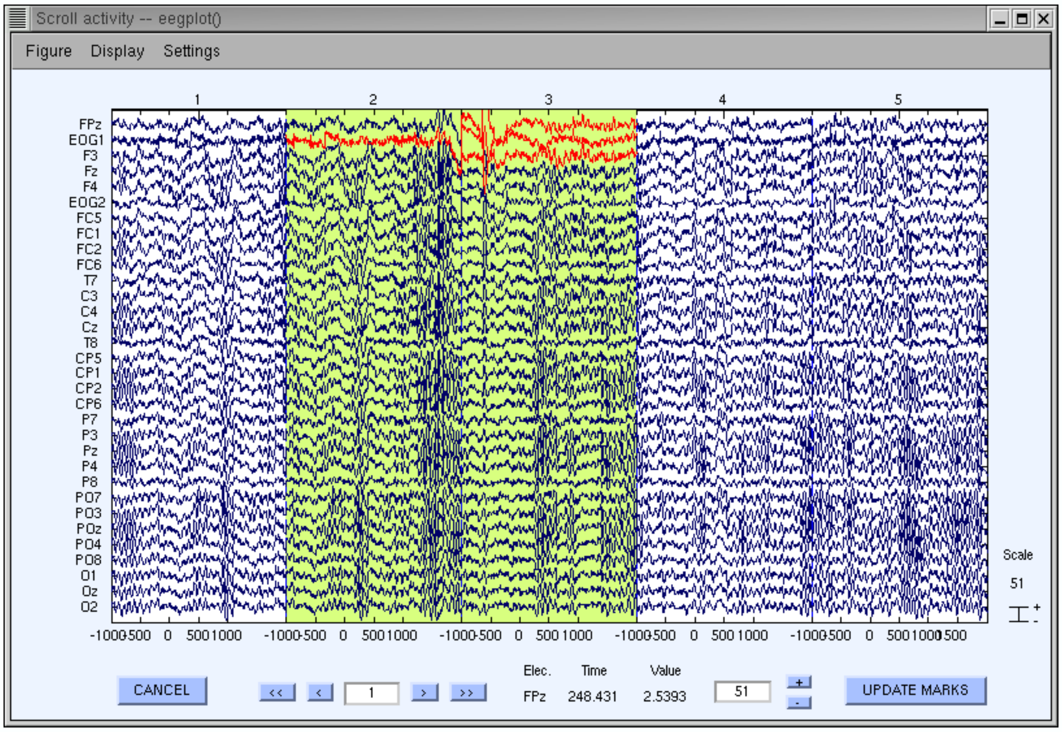


Find abnormal values

Upper limit(s) (uV)	75	Lower limit(s) (uV)	-75
Start time(s) (ms)	-1000	Ending time(s) (ms)	2000
List of electrode(s)	1:32	Marked trials:	0

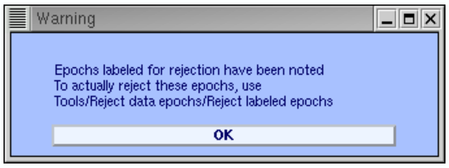
CALC/PLOT HELP

Marked trials are highlighted in the eegplot() window (below) that then pops up. Now epochs marked for rejection may be un−marked manually simply by clicking on them. Note that the activity at some electrodes is shown in red, indicating the electrodes in which the outlier values were found. Press "**Update Marks**" to confirm the epoch markings.


Scroll activity -- eegplot()
Figure
Display
Settings
FPz EOG1 F3 Fz F4 EOG2 FC5 FC1 FC2 FC6 T7 C3 C4 Cz T8 CP5 CP1 CP2 CP6 P7 P3 Pz P4 P8 PO7 PO3 POz PO4 PO8 O1 Oz O2
1 2 3 4 5
Scale
51
CANCEL
UPDATE MARKS

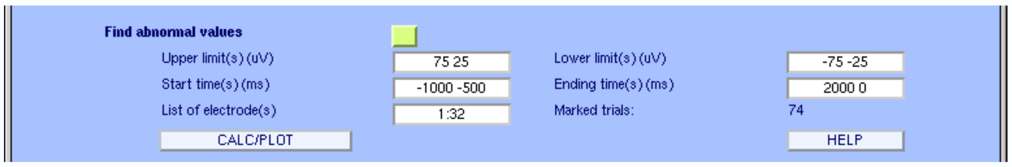
Elec. FPz Time 248.431 Value 2.5393

At this point, a warning will pop up indicating that the marked epochs have **not** yet been rejected and are simply marked for rejection. When seeing this warning, press "**OK**" to return to main rejection window.



Epochs labeled for rejection have been noted
To actually reject these epochs, use
Tools/Reject data epochs/Reject labeled epochs

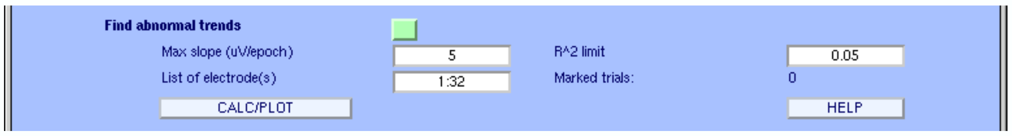
Also note that several thresholds may be entered along with separate applicable time windows. In the following case, in addition to the preceding rejection, additional epochs in which EEG potentials went above **25** µV or below **−25** µV during the **−500** to **0** ms interval will be marked for rejection.



Find abnormal values	🟩		
Upper limit(s) (uV)	75 25	Lower limit(s) (uV)	-75 -25
Start time(s) (ms)	-1000 -500	Ending time(s) (ms)	2000 0
List of electrode(s)	1:32	Marked trials:	74

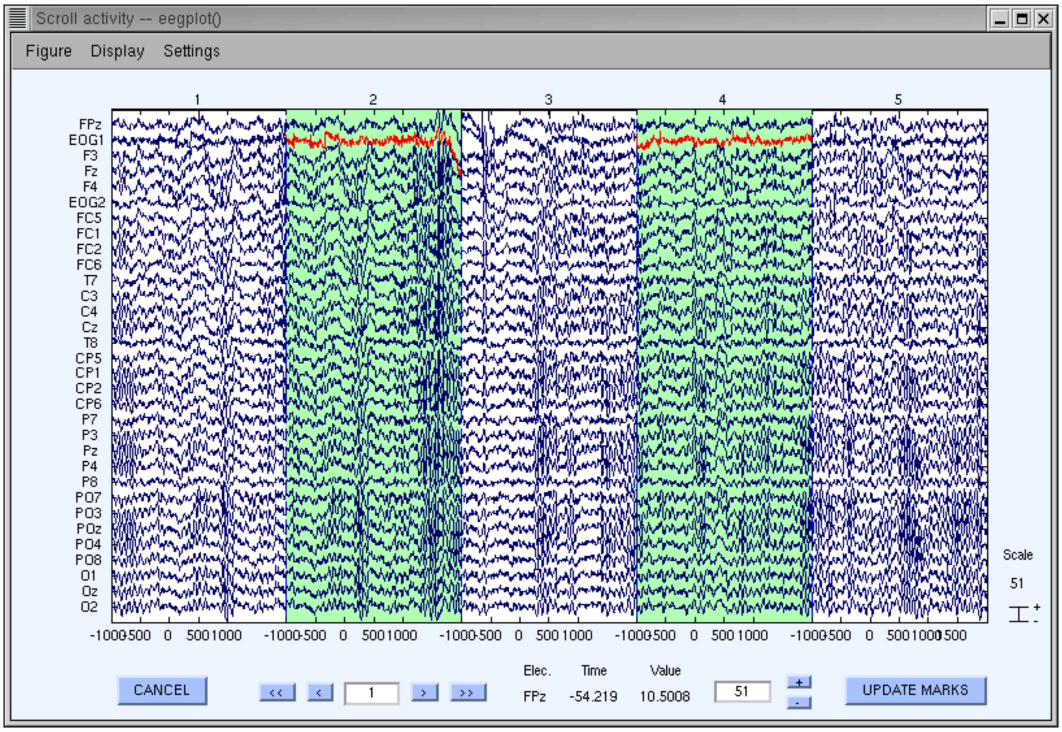
III.2.3. Rejecting abnormal trends

Artifactual currents may cause linear drift to occur at some electrodes. To detect such drifts, we designed a function that fits the data to a straight line and marks the trial for rejection if the slope exceeds a given threshold. The slope is expressed in microvolt over the whole epoch (50, for instance, would correspond to an epoch in which the straight−line fit value might be 0 µv at the beginning of the trial and 50 µ v at the end). The minimal fit between the EEG data and a line of minimal slope is determined using a standard R−square measure. To test this, in the main rejection window enter the following data under ***Rejection of stable activity*** (which calls function pop_rejtrend()), and press "**CALC/PLOT**".



Find abnormal trends	🟩		
Max slope (uV/epoch)	5	R^2 limit	0.05
List of electrode(s)	1:32	Marked trials:	0

The following window will pop up. Note that in our example the values we entered are clearly too low (otherwise we could not detect any linear drifts in in this clean EEG dataset). Electrodes triggering the rejection are again indicated in red. To deselect epochs for rejection, click on them. To close the scrolling window, press the "**Update marks**" button.

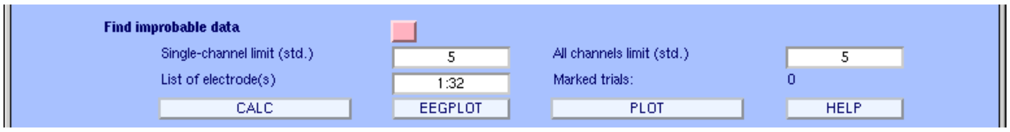


Elec. Time Value
FPz -54.219 10.5008

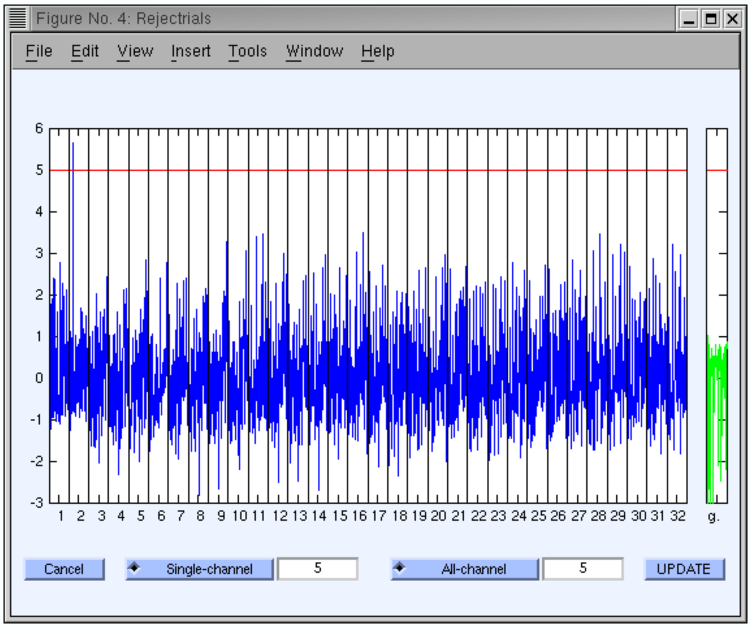
Note: Calling this function (pop_rejtrend()) either directly from the command line, or by selecting **Tools > Reject data epochs > Reject flat line data**, allows specifying additional parameters.

III.2.4. Rejecting improbable data

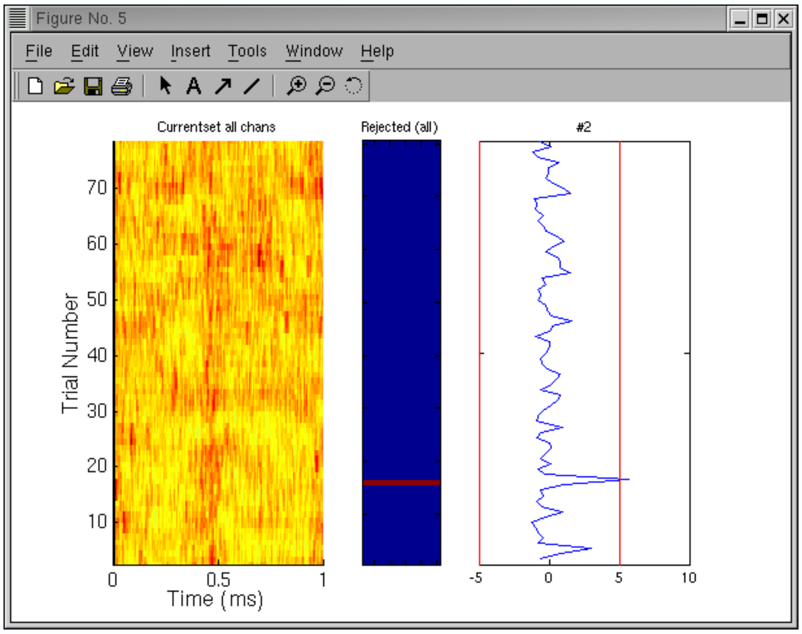
By determining the probability distribution of values across the data epochs, one can compute the probability of occurrence of each trial. Trials containing artifacts are (hopefully) improbable events and thus may be detected using a function that measures the probability of occurrence of trials. Here, thresholds are expressed in terms of standard deviations of the mean probability distribution. Note that the probability measure is applied both to single electrodes and the collection of all electrodes. In the main rejection window, enter the following parameters under ***Probability rejection*** (which calls function pop_jointprob()), then press "**PLOT**".


Find improbable data
Single-channel limit (std.) 5
All channels limit (std.) 5
List of electrode(s) 1:32
Marked trials: 0

The first time this function is run on a dataset, its computation may take some time. However the trial probability distributions will be stored for future calls. The following display shows the probability measure value for every trial and electrode (each number at the bottom shows an electrode, the blue traces the trial values). On the right, the panel containing the green lines shows the probability measure over all the electrodes. Rejection is carried out using both single− and all−channel thresholds. To disable one of these, simply raise its limit (e.g. to 15 std.). The probability limits may be changed in the window below (for example, to 3 standard deviations). The horizontal red lines shift as limit values are then modified.



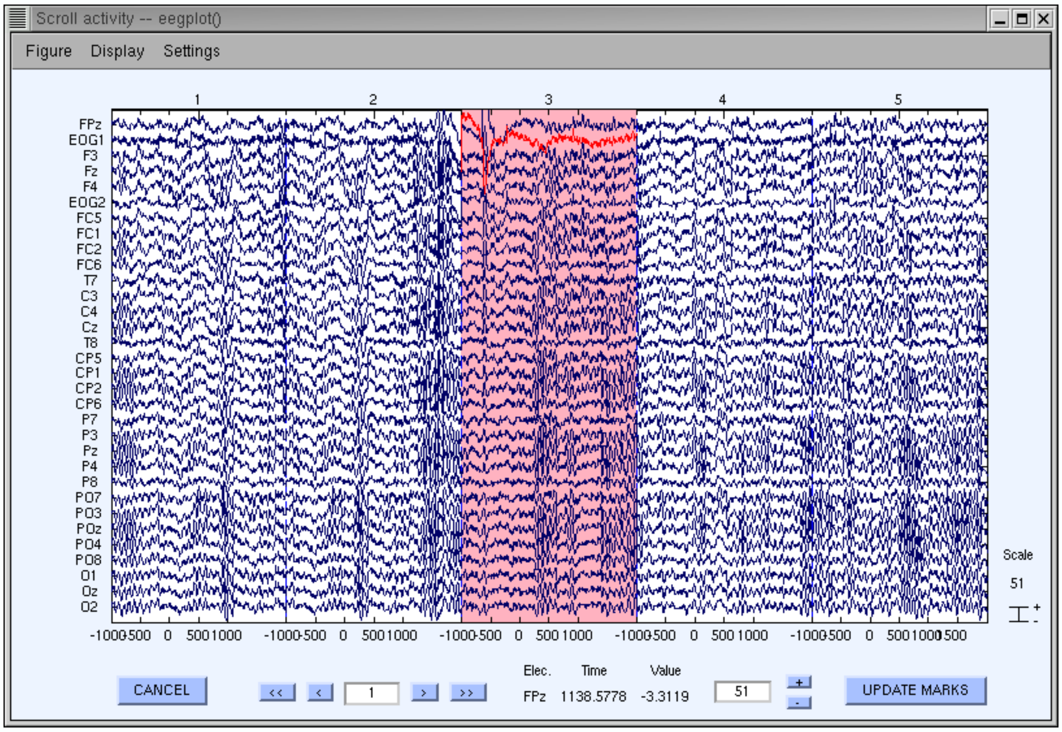
It is also possible to look in detail at the trial probability measure for each electrode by clicking on its sub−axis plot in the above window. For instance, by clicking on plot for electrode number two, the following window would pop−up (from left to right, the first plot is the **ERPimage** of the square values of EEG data for each trial, the second plot indicates which trials were labeled for rejection (all electrodes combined) and the last plot is the original probability measure plot for this electrode with limits indicated in red).



Close this window and press the "**UPDATE**" button in the probability plot window to add to the list of trials marked for rejection.

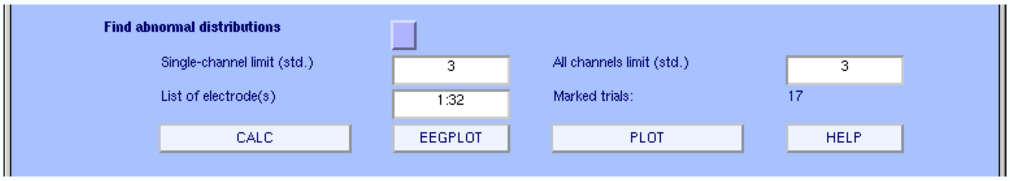
Now, back in the main rejection window, note that adjustments to the thresholds in the trial probability window have been automatically updated.

Now press the "**EEGPLOT**" button to obtain the standard scrolling data window. Inspect the trials marked for rejection, make any adjustments manually, then close the window by pressing "**UPDATE MARKS**".



III.2.5. Rejecting abnormally distributed data

Artifactual data epochs sometimes have very 'peaky' activity value distributions. For instance, if a discontinuity occurs in the activity of data epoch for a given electrode, the activity will be either close to one value or the other. To detect these distributions, a statistical measure called kurtosis is useful. Kurtosis is also known as the fourth cumulant of the distribution (the skewness, variance and mean being the first three). A high positive kurtosis value indicates an abnormally 'peaky' distribution of activity in a data epoch, while a high negative kurtosis value indicates abnormally flat activity distribution. Once more, single− and all−channel thresholds are defined in terms of standard deviations from mean kurtosis value. For example, enter the following parameters for ***kurtosis−based rejection*** (which calls function pop_rejkurt()), then press "**PLOT**".



Find abnormal distributions

Single-channel limit (std.): 3
All channels limit (std.): 3
List of electrode(s): 1:32
Marked trials: 17

As with probability rejection, these limits may be updated in the pop−up window below, and pressing "**EEGPLOT**" in the main rejection window (above) brings up the standard data scrolling window display which may be used to manually adjust the list of epochs marked for rejection.

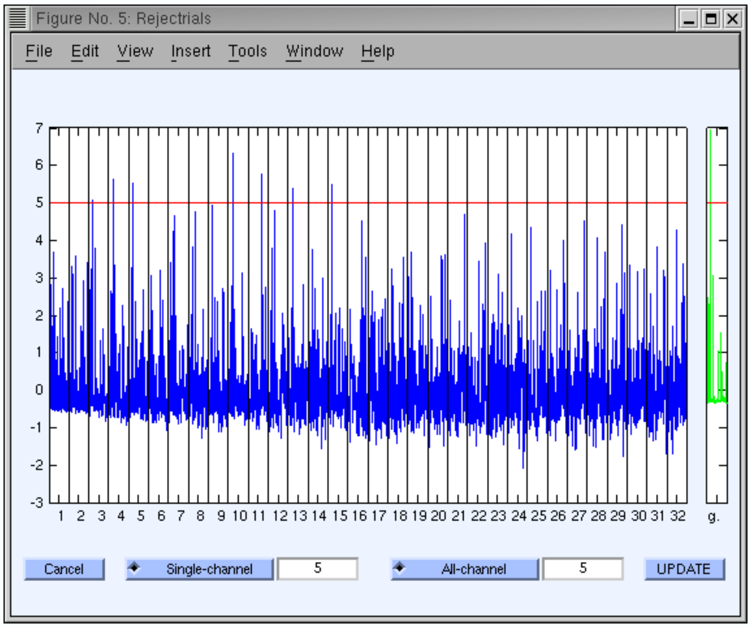
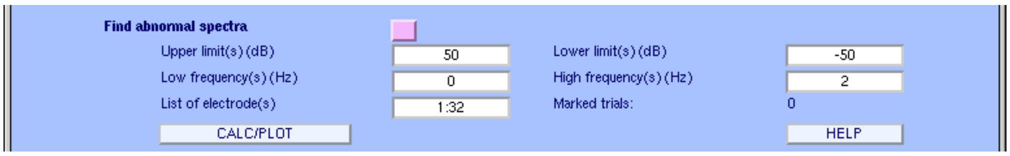


Figure No. 5: Rejectrials
File Edit View Insert Tools Window Help
Cancel
Single-channel
5
All-channel
5
UPDATE
g.

III.2.6. Rejecting abnormal spectra

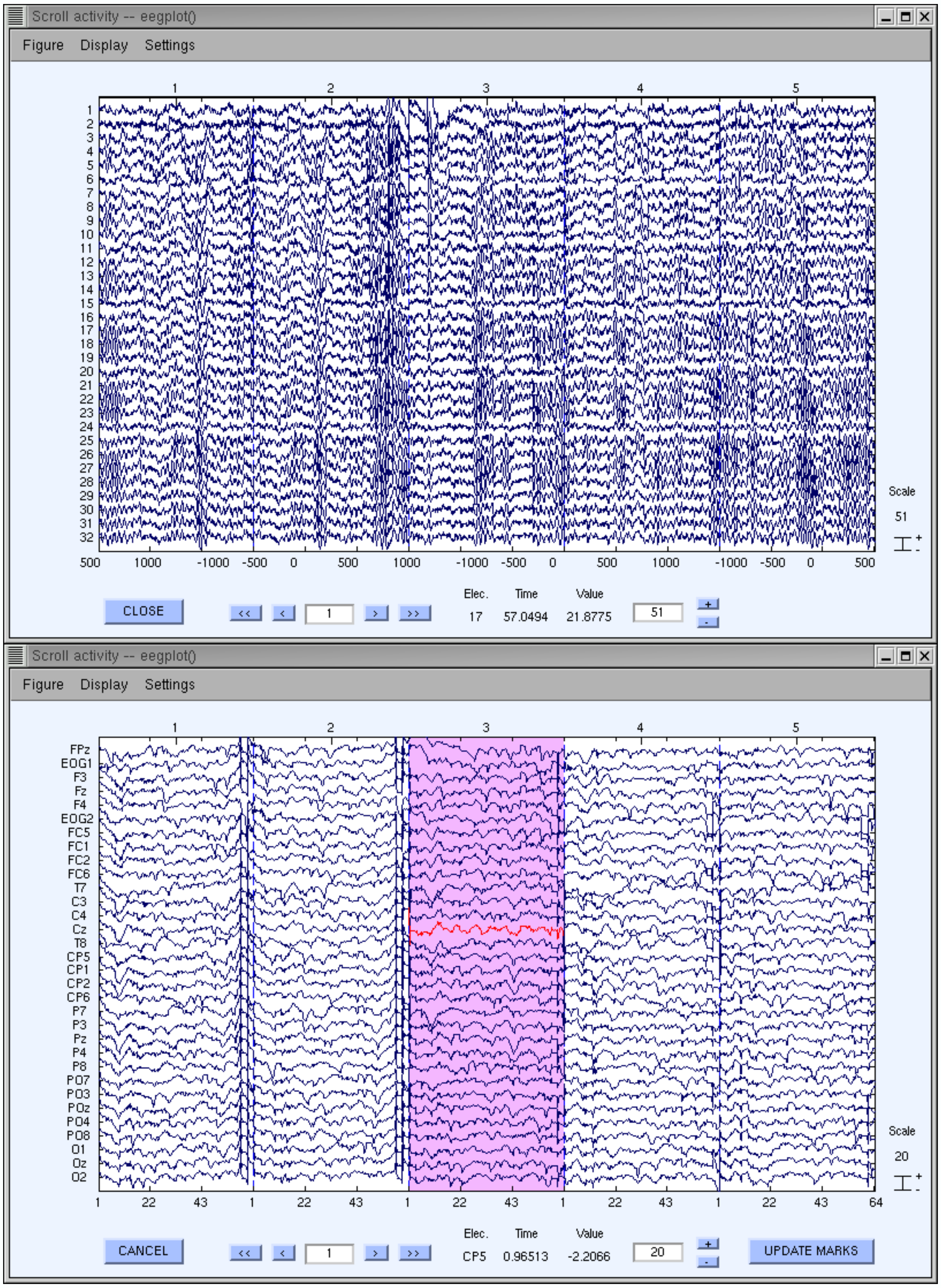
According to our analysis (Delorme, A., Makeig, S., Jung, T.P., Sejnowski, T.J. (2001), "Automatic rejection of event−related potential trials and components using independent component analysis"), rejecting data epochs using spectral estimates may be the most effective method for selecting data epochs to reject for analysis. In this case, thresholds are expressed in terms of amplitude changes relative to baseline in dB. To specify that the spectrum should not deviate from baseline by **+/−50** dB in the **0−2** Hz frequency window, enter the following parameters under ***reject abnormal spectra*** (which calls function **pop_rejspec()**), then press "**CALC/PLOT**".



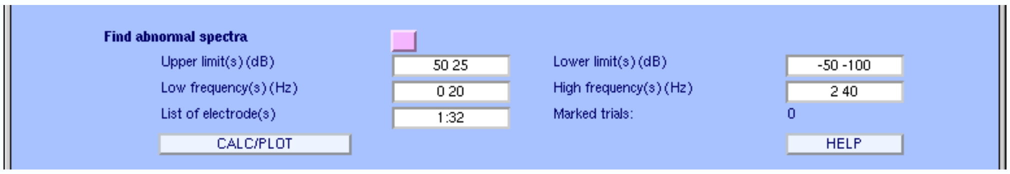
Find abnormal spectra

Upper limit(s) (dB)	50	Lower limit(s) (dB)	-50
Low frequency(s) (Hz)	0	High frequency(s) (Hz)	2
List of electrode(s)	1:32	Marked trials:	0

Computing the data spectrum for every data epoch and channel may take a while. We use a frequency decomposition based on the slepian multitaper (Matlab pmtm() function) to obtain more accurate spectra than using standard Fourier transforms. After computing the trial spectra, the function removes the average power spectrum from each trial spectrum and tests whether the remaining spectral differences exceed the selected thresholds. Here, two windows pop up (as below), the top window being slaved to the bottom. The top window shows the data epochs and the bottom window the power spectra of the same epochs. When scrolling through the trial spectra, the user may use the top window to check what caused a data trial to be labeled as a spectral outlier. After further updating the list of marked trials, close both windows by pressing "**UPDATE MARKS**" in the bottom window.


Scroll activity -- eegplot()
Figure Display Settings
Scale 51
CLOSE
Elec. Time Value
17 57.0494 21.8775
51
Scroll activity -- eegplot()
Figure Display Settings
FPz EOG1 F3 Fz F4 EOG2 FC5 FC1 FC2 FC6 T7 C3 C4 Cz T8 CP5 CP1 CP2 CP6 P7 P3 Pz P4 P8 PO7 PO3 POz PO4 PO8 O1 Oz O2
Scale 20
CANCEL
Elec. Time Value
CP5 0.96513 -2.2066
20
UPDATE MARKS

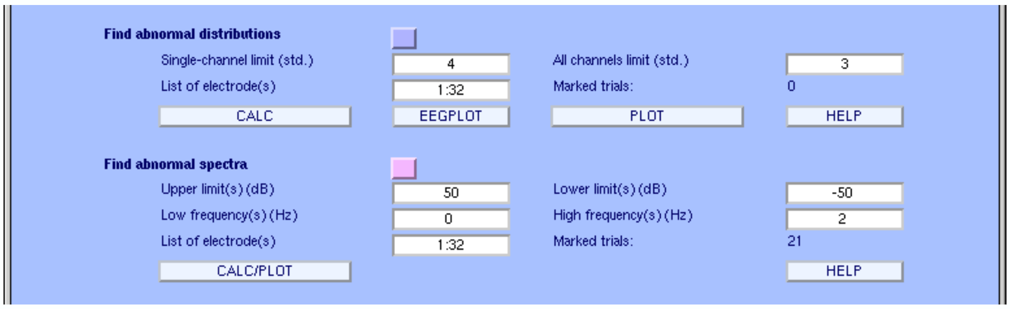
As with standard rejection of extreme values, several frequency thresholding windows may be defined. In the window below, we specify that the trial spectra should again not deviate from the mean by **+/−50** dB in the **0−2** Hz frequency window (good for catching eye movements) and should not deviate by **+25** or **−100** dB in the **20−40** Hz frequency window (useful for detecting muscle activity).


Find abnormal spectra

Upper limit(s) (dB)	50 25	Lower limit(s) (dB)	-50 -100
Low frequency(s) (Hz)	0 20	High frequency(s) (Hz)	2 40
List of electrode(s)	1:32	Marked trials:	0

III.2.7. Inspecting current versus previously proposed rejections

To compare the effects of two different rejection thresholds, the user can plot the currently and previously marked data epochs in different colors. To do this, change the option in the long rectangular tab under ***Plotting options***. Select **Show previous and new trials marked for rejection by this measure selected above**. For instance, using ***kurtosis rejection***, enter the following parameters and press "**EEGPLOT**".

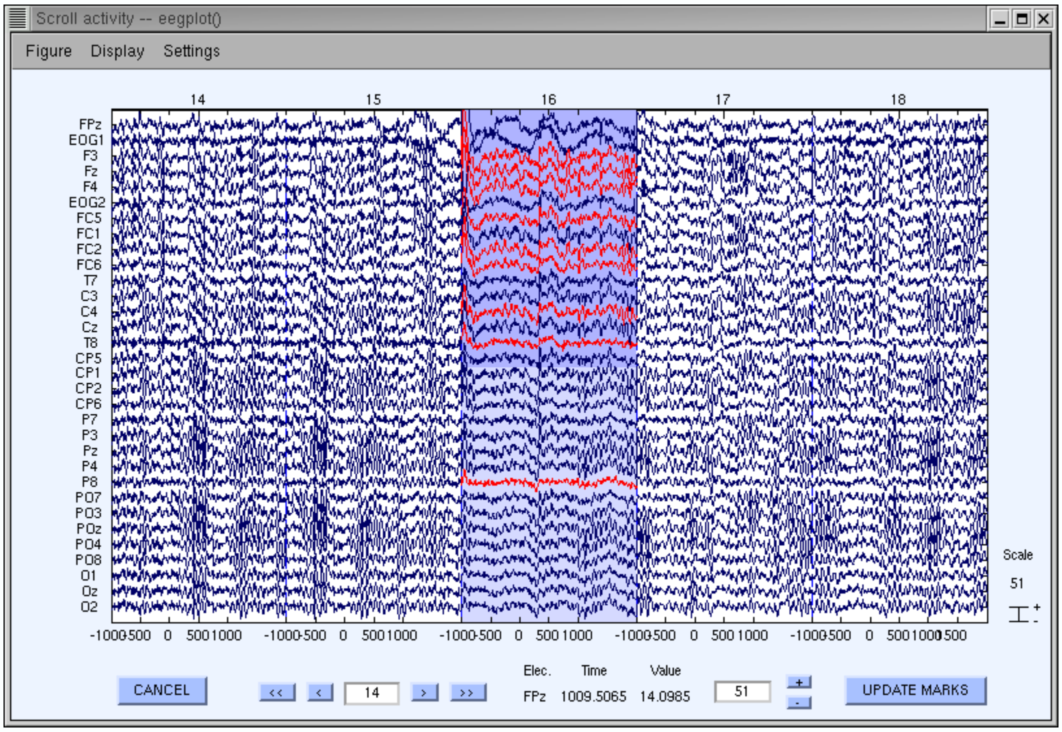

Find abnormal distributions

Single-channel limit (std.)	4	All channels limit (std.)	3
List of electrode(s)	1:32	Marked trials:	0

Find abnormal spectra

Upper limit(s) (dB)	50	Lower limit(s) (dB)	-50
Low frequency(s) (Hz)	0	High frequency(s) (Hz)	2
List of electrode(s)	1:32	Marked trials:	21

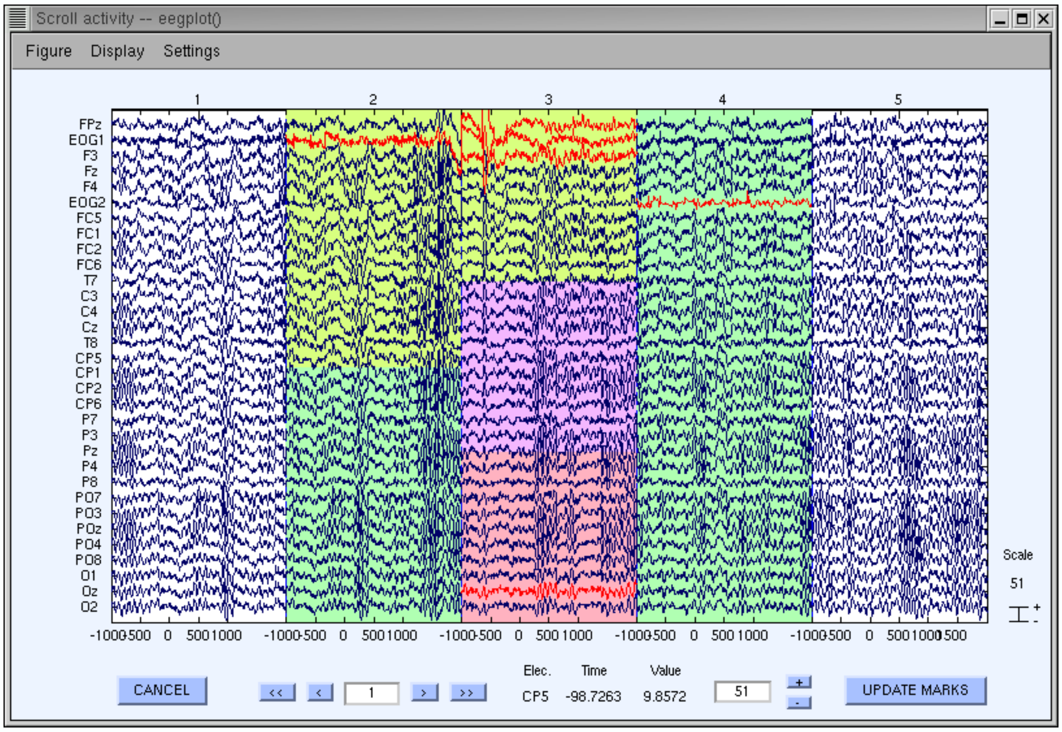
The scrolling window appears and now shows that at trial 57 the blue window is actually divided into two, with dark blue on the top and light blue on the bottom. Dark blue indicates that the trial is currently marked as rejected. The light blue indicates that the trial was already rejected using the previous rejection threshold.



Note that pressing "**UPDATE MARKS**" only updates the currently rejected trials and optional visual adjustments. Previously rejected trials are ignored and can not be removed manually in the plot window.

III.2.8. Inspecting results of all rejection measures

To visually inspect data epochs marked for rejection by different rejection measures, select **Show all trials marked for rejection measures by the measure selected above or checked below** in the long rectangular tab under ***Plotting options***. Then press "**CALC/PLOT**" under ***Threshold frequency***. Rejected windows are divided into several colored patches, each color corresponding to a specific rejection measure. Channels colored red are those marked for rejection by any method.



When visually modifying the set of trials labeled for rejection in the window above, all the **current rejection measure** are affected and are reflected in the main rejection window after you press the "**UPDATE MARKS**" button.

III.2.9. Notes and strategy

−− The approach we take to artifact rejection is to use statistical thresholding to **suggest** epochs to reject from analysis. Current computers are fast enough to allow easy confirmation and adjustment of suggested rejections by visual inspection. We therefore favor **semi−automated rejection** coupled with visual inspection.

−− All the functions presented here can also be called individually through **Plot > Reject data epochs** or from the Matlab command line.

−− After labeling trials for rejection, it is advisable to **save** the dataset before actually rejecting the marked trials (marks will be saved along with the dataset). This gives one the ability go back to the original dataset and recall which trials were rejected.

−− All these rejection measures are useful, but if one does not know how to use them they may be inefficient. Because of this, we have not provided standard rejection thresholds for all the measures. In the future we may include such information in this tutorial. The user should **begin** by visually rejecting epochs from some test data, then adjust parameters for one or more of the rejection measures, comparing the visually selected epochs with the results of the rejection measure. All the measures can capture both simulated and real data artifacts. In our experience, the most efficient measures seem to be frequency threshold and linear trend detection.

−− We typically apply semi−automated rejection to the **independent component activations** instead of to the raw channel data, as described below.

III.3. Rejection based on independent data components

We usually apply the measures described above to the activations of the **independent components** of the data. As independent components tend to concentrate artifacts, we have found that bad epochs can be more easily detected using independent component activities. The functions described above work exactly the same when applied to data components as when they are applied to the raw channel data. Select **Tools > Reject data using ICA > Reject data (all methods)**. We suggest that the analysis be done iteratively in the following **seven steps**:

1. Visually reject unsuitable (e.g. paroxysmal) portions of the continuous data.
2. Separate the data into suitable short data epochs.
3. Perform ICA on these epochs to derive their independent components.
4. Perform semi−automated and visual−inspection based rejection on the derived components.*
5. Visually inspect and select data epochs for rejection.
6. Reject the selected components and data epochs.
7. Perform ICA a second time on the pruned collection of short data epochs −− This may improve the quality of the ICA decomposition, revealing more independent components accounting for neural, as opposed to mixed artifactual activity. If desired, the ICA unmixing and sphere matrices may then be applied to (longer) data epochs from the same continuous data. Longer data epochs are useful for time/frequency analysis, and may be desirable for tracking other slow dynamic features.

***Note**: After closing the main ICA rejection window, select **Tools > Reject data using ICA > Export marks to data rejection** and then **Tools > Reject data epochs > Reject by inspection** to visualize data epochs marked for rejection using ICA component activities.

IV. Writing EEGLAB Matlab Scripts

IV.1. Why write EEGLAB Matlab scripts?

EEGLAB is a collection of Matlab functions called from a main graphic interface. Writing EEGLAB Matlab scripts simply involves calling these functions from a script file or from command line instead of causing them to be called interactively by EEGLAB. EEGLAB's history mechanism eases the transition from menu−based to script−based computing. It allows the user to perform exploratory signal processing on a sample data set, then to use the accumulated commands issued from EEGLAB windows to be written to a script file and modified using any script editor.

Writing Matlab scripts to perform EEGLAB analyses allows the user to largely automate the processing of one or more datasets. Because advanced analyses may involve many parameter choices and require fairly lengthy computations, it is often more convenient to write a custom script, particularly to process multiple data sets in the same way.

IV.2. Using EEGLAB command history to perform basic EEGLAB script writing

Every time the user calls a function from the EEGLAB menus, the function call is saved in EEGLAB history. Thus, basic script writing simply involves copying and pasting the function history called by EEGLAB. Typing:

```
>> h
```

under Matlab displays the EEGLAB history on the Matlab command line. For instance, after performing the first step of the main tutorial (opening an existing data set), typing **h** on the command line may return the following text:

```
[ALLEEG EEG CURRENTSET ALLCOM] = eeglab;
EEG = pop_loadset( 'ee114squares.set', '/home/arno/tutorial/');
[ALLEEG EEG CURRENTSET] = eeg_store(ALLEEG, EEG);
```

The first command (**eeglab**) runs EEGLAB and initializes several EEGLAB variables. The second command (**pop_loadset()**) reads the dataset into the EEG structure variable, and the last (**eeg_store()**) stores the dataset in the ALLEG structure variable. For more detailed information, read the Matlab help messages for these functions:
either (1) via the EEGLAB menu selections

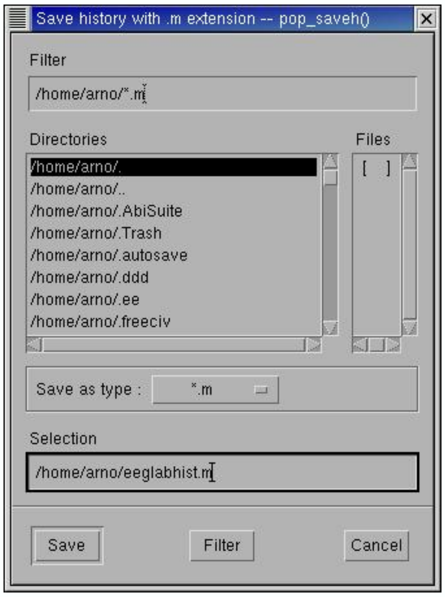
For **pop_loadset()**: via **Help > EEGLAB functions > Interactive pop_functions** or **Help > EEGLAB menus**

For **eeg_store()**: via **Help > EEGLAB advanced > Admin functions**

Or (2) using Matlab command line help

```
>> help [functionname]
```

Now use menu item **File > Save history** to save the command history into an ascii−text Matlab script file. Save the file in the current directory, or in a directory in the Matlab command path (i.e., in the list returned by **>> path**). Selecting the "**Save history**" menu will pop up the window below:



Clicking "**Save**" in the window above will cause the command history to be saved into the Matlab script file "**eeglabhist.m**" (you can choose any name for this file, as long as it ends in the standard Matlab script file extension, "**.m**"). Now try closing the current Matlab session, restart Matlab, and run the script saved above by typing

```
>> eeglabhist
```

The main EEGLAB window is created and the data set is read. Now open the script file "**eeglabhist.m**" in any text editor. **Note:** when the file was saved, an extra command, "**>> eeglab redraw** " was added at the bottom of the file to insure that the main graphic interface would be updated after the data set was (re)read.

Building and running short or long EEGLAB Matlab scripts saved by EEGLAB history can be that simple. Simply perform the EEGLAB steps desired via the EEGLAB menu, save the EEGLAB command history, and re−run the saved script file to have Matlab repeat all the steps performed manually. For instance, following the first several steps of the main tutorial, the command **>> h** would return (with Matlab−style comments in black and bold added for clarity):

```
[ALLEEG EEG CURRENTSET ALLCOM] = eeglab; % start EEGLAB under Matlab

EEG = pop_loadset( 'ee114squares.set', '/home/payton/ee114/'); % read in the
dataset
[ALLEEG EEG CURRENTSET] = eeg_store(ALLEEG, EEG); % copy it to ALLEEG

EEG = pop_editeventfield( EEG, 'indices',  '1:155', 'typeinfo',  'Type of the
event'); % edit the dataset event field
[ALLEEG EEG] = eeg_store(ALLEEG, EEG, CURRENTSET); % copy changes to
ALLEEG
```

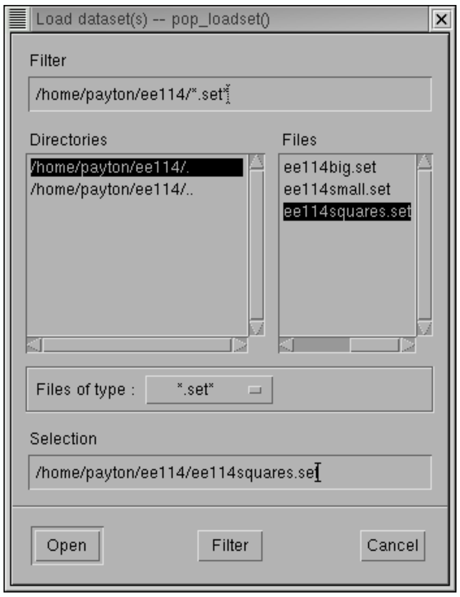
EEG.comments = pop_comments('', '', strvcat('In this experiment, stimuli can appear at 5 locations ', 'One of them is marked by a green box ', 'If a square appears in this box, the subject must respond, otherwise he must ignore the stimulus.', ' ', 'These data contain responses to (non−target) circles appearing in the attended box in the left visual field ')); *% update the dataset comments field*
[ALLEEG EEG] = eeg_store(ALLEEG, EEG, CURRENTSET); *% copy changes to ALLEEG*

pop_eegplot(EEG, 1, 0, 1); *% pop up a scrolling window showing the component activations*

EEG.chanlocs=pop_chanedit(EEG.chanlocs, { 'load', '/home/payton/ee114/chan32.locs'},{ 'convert',{ 'topo2sph', 'gui'}},{ 'convert',{ 'sph2cart', 'gui'}}); *% read the channel location file and edit the channel location information*

figure; pop_spectopo(EEG, 0, [−1000 237288.3983] , 'percent', 20, 'freq', [10], 'icacomps', [1:0],'electrodes','off');
% plot RMS power spectra of the ICA component activations; show a scalp map of total power at 10 Hz plus maps of the components contributing most power at the same frequency

Important note that functions called from the main EEGLAB interactive window display the name of the underlying pop_function in the window title bar. For instance, selecting **File > Load an existing dataset** to read an existing data set uses EEGLAB function "pop_loadset()".



Load dataset(s) -- pop_loadset()

Filter
/home/payton/ee114/*.set

Directories
/home/payton/ee114/.
/home/payton/ee114/..

Files
ee114big.set
ee114small.set
ee114squares.set

Files of type : *.set*

Selection
/home/payton/ee114/ee114squares.set

Open Filter Cancel

The next steps in learning to use EEGLAB Matlab scripts involve learning to change EEGLAB function parameters and adding loops to perform multiple analyses.

IV.3. How do EEGLAB pop_functions work?

When an EEGLAB **pop_function** is called without a minimum number of parameter arguments, either from the EEGLAB menu or from the Matlab command line, it pops up an interactive window to collect the required parameter values from the user. Otherwise, the function simply executes using the given parameters without popping up an interactive window. This allows EEGLAB−based Matlab scripts to run without requiring additional user input.

For instance open a new Matlab session and try

```
>> [ALLEEG EEG CURRENTSET ALLCOM] = eeglab;
>> EEG = pop_loadset;
```

An interactive window now pops up to ask for the dataset name, just as it would do if the **pop_loadset()** command were issued from the EEGLAB menu. If, on the other hand, the user provides two string arguments, one for the filename and one for the file path, to the **pop_loadset()** function, no interactive window appears and the data set is read automatically. **All the interactive EEGLAB pop_functions work this way!**

Also, most of the data−processing pop_functions (each named **pop_[funcname]()** call the processing function **[funcname]** . In fact the **pop_funcname()** function is a graphic−user interface function that operates on the EEG data structure using the standalone processing function **funcname()** that has no knowledge of the EEG dataset structure (described in the next paragraph). For instance, **pop_erpimage()** calls the general data plotting function **erpimage()**. To study the input parameters to these functions, either use EEGLAB help menu or Matlab help, for example:

```
>> help erpimage
>> help pop_erpimage
```

Note: only **pop_funcname()** functions and **eeg_funcname()** functions process the EEG dataset structure described in the next paragraph (in contrast to **pop_funcname()** functions, **eeg_funcname()** functions do not pop up interactive windows).

Important note: if **pop_funcname()** is a plotting function, then a new figure is created automatically only when the function is called in pop−up window mode. Otherwise, a **pop_funcname()** command should be preceded by a Matlab "**figure;**" command (see the **pop_spectopo()** example above). One can then create compound figures using Matlab "**>> subplot**" or the more advanced EEGLAB **sbplot()** function.

What do pop_functions return?

When called from the EEGLAB interface, pop_functions do not return variables. Instead, they alter (when called for) the EEG data structure itself. However, many of the visualization functions in the EEGLAB toolbox do return variables when useful (e.g., the data plotted after processing, etc.). In almost all cases, the EEGLAB calling pop_functions can return the same variables when they are called from a script or the commandline. For example:

```
>> [outdata, outvar, outtrials] = erpimage( EEG.data(12,:), zeros(1,EEG.trials),
EEG.times, '', 1, 1, 'nosort')
```

% or the equivalent

>> [outdata, outvar, outtrials] = pop_erpimage(EEG,1,12); *% ERP−image plot of channel 12*

VI.4. What is the structure of an EEGLAB dataset?

IV.4.1. EEG

EEGLAB variable "**EEG**" is a Matlab structure that contains all the information about the current EEGLAB dataset. For instance, again follow the **main tutorial file** until data epochs have been extracted and the ICA decomposition computed. Then type

>> EEG

This will produce the following command line output:

```
EEG =

                setname:'Epoched from "ee114 continuous"'
               filename:'ee114squaresepochs.set'
               filepath:'/home/arno/ee114/'
                   pnts:385
                 nbchan:32
                 trials:80
                  srate:128.087
                  xmin :−1
                   xmax:1.9998
                   data:[32x385x80 double]
                 icawinv:[32x32 double]
              icasphere:[32x32 double]
             icaweights:[32x32 double]
                 icaact:[]
                  event:[1x157 struct]
                  epoch:[1x80 struct]
               chanlocs:[1x32 struct]
               comments:[8x150 char]
       eventdescription:{'Type of the event'  ' '  'The latency at which the event occurred'  ''}
       epochdescription:{}
               specdata:[]
                specica:[]
                 averef:'no'
                 reject:[1x1 struct]
                  stats:[1x1 struct]
             splinefile:[]
                  times:[1x385 double]
```

Above, we have highlighted several important fields. See the help message of the **eeg_checkset()** function (which check data set consistency) for the meaning of all the fields.

Most fields contain single values or an array of values. However three important fields themselves contain structures: "**EEG.chanlocs**", "**EEG.event**" and " **EEG.epoch**". These are described below.

IV.4.2. EEG.chanlocs

This EEG structure field stores information about the EEG channel locations and channel names. For example, typing

```
>> EEG.chanlocs
```

now returns

```
ans =

1x32 struct array with fields:
   labels
   theta
   radius
   X
   Y
   Z
   sph_theta
   sph_phi
   sph_radius
```

Here, **EEG.chanlocs** is a struct array of length 32 (one structure for each of the 32 channels in this dataset). Typing

```
>> EEG.chanlocs(1)
```

returns

```
ans =

    labels:'FPz'
     theta:0
    radius:0.46
         X:[]
         Y:[]
         Z:[]
 sph_theta:[]
   sph_phi:[]
sph_radius:[]
```

These values store the channel location coordinates and label of the first channel ('FPz'). Note that the X, Y, Z and three spherical coordinate fields have not been computed yet, so they are empty). Use **pop_chanedit()** to compute them or menu item **Edit > Channel locations**. The value of any EEG structure field may also be modified manually from the Matlab command line (but see the cautions under Modifying the EEG structure below).

IV.4.3. EEG.event

This EEG structure field contains records of the experimental events that occurred while the data were being recorded. Typing

```
>> EEG.event
```

returns

```
ans =

1x157 struct array with fields:
   type
   position
   latency
   epoch
```

This dataset contains 157 events. In our example, each EEG.**event** structure contains four types of user−defined information: "EEG.event.**type**", the type of event, "EEG.event.**position**", the position at which the target appeared, "EEG.event.**latency**", the position (frame offset) in the continuous (whole) data matrix (EEG.**data**) at which the event occurred, and "EEG.event.**epoch**", the number of the data epoch in which the event occurred. Note that for events occurring at specific latencies, the field "EEG.event.**latency**" **must** be present and the user should take care to enter event latency information for continuous data as a field named "**latency**". The field "EEG.event.**type**" is a field used by the function **pop_epoch()** to extract event−related data epochs from continuous EEG data, so it is best to define it as well. For example, to inspect the stored information about the first event, type

```
>> EEG.event(1) % note the use of parentheses
```

returning

```
ans =

      type:'square'
  position:2
   latency:129
     epoch:1
```

The first event was of type "**square**". It appeared at position "**2**" (see the **main tutorial** for interpretation) at data point "**129**" of epoch "**1**". Note that the menu item **Edit > Event values** (calling function **pop_editeventvals()**) displays the latency of this event in milliseconds relative to the epoch time−locking event (i.e., for this event as "**0 ms** ", since this is a time−locking event). Use functions **eeg_lat2point()** and **eeg_point2lat()** to convert between event latencies in data points in milliseconds. Note that in future EEGLAB releases, "**latency**" information may be stored internally in second in the **EEG.event** structure for alignment with fMRI data in our **FMRILAB toolbox** .

Simple Matlab commands can be used to manipulate EEGLAB data structures. For example, the following command lists the types of events contained in this dataset:

```
>> unique({EEG.event.type}) % note the use of curly braces

'square'    'rt'
```

EEGLAB function cell2mat() is useful for processing numeric event values. For example, the following command returns the array of positions of each event in an integer array.

```
>> cell2mat({EEG.event.position}); % note again the use of curly braces

ans =

  Columns 1 through 20

    2    2    2    2    2    2    2    2    2    2    2    1    1    1    1    1    1    1    1    1

  Columns 21 through 40

    1    2    2    2    2    2    2    2    2    2    2    1    1    2    1    2    2    1    1    1

  ...etc...
```

The following script or commandline code illustrates how to manipulate a dataset event list using the sample (epoched) dataset, by adding a new class of (pseudo) events to an existing dataset. We want the new event latencies to be 100 ms before existing type−'square' events. (Note: We might want to do this to allow sub−epoching relative to the new event latencies).

```
% Add new events to a loaded dataset
nevents = length(EEG.event);
for index = 1 : nevents
  if strcmpi(EEG.event(index).type, 'square')     % Add events relative to existing events
     EEG.event(end+1) = EEG.event(index);    % Add event to end of event list
     EEG.event(end).latency = ...     % Specify the event latency (in real data points)
         EEG.event(index).latency−0.1*EEG.srate;    % before the referent event
     EEG.event(end).type = 'cue';     % Make the type of the new event 'P'
  end;
end;

EEG = eeg_checkset(EEG, 'eventconsistency');     % Check all events for consistency
EEG = pop_editeventvals( EEG, 'sort', { 'epoch', [0], 'latency', [0] } );     % Re−sort events
[ALLEEG EEG CURRENTSET] = eeg_store(ALLEEG, EEG, CURRENTSET);   % Store
dataset
eeglab redraw     % Redraw the main eeglab window
```

IV.4.4. EEG.epoch

This structure is similar to the **EEG.event** structure, except that there is one entry per epoch. Note that EEGLAB functions never use this structure internally. It is computed from the **EEG.event** structure by eeg_checkset() (with flag 'eventconsistency') as a convenience for the user who may wish to use it in writing more advanced EEGLAB scripts.

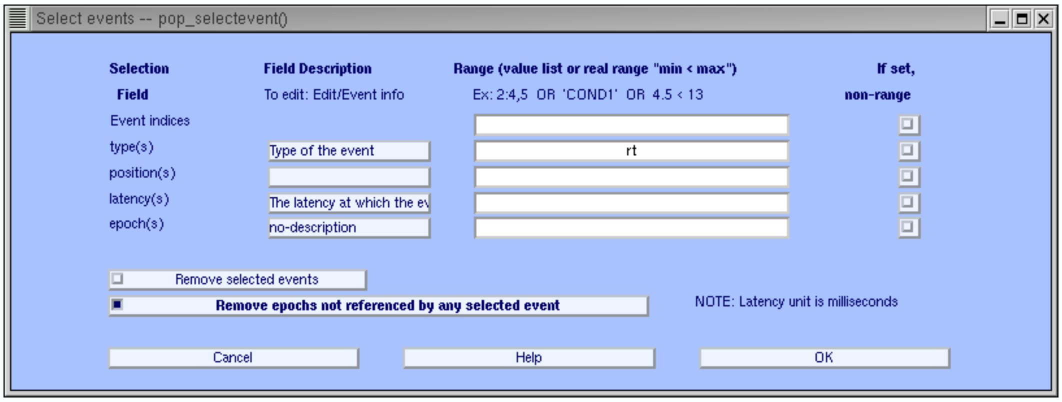
```
>> EEG.epoch
```

```
ans =

1x80 struct array with fields:
   event
   eventtype
   eventposition
   eventlatency
   eventepoch
```

Note that this dataset contains 80 epochs (or trials). Now type

```
        >> EEG.epoch(1)

ans =

            event:[1 2 3]
     eventlatency:{[0]  [695.2650]  [1.0823e+03]}
    eventposition:{[2]  [2]  [2]}
        eventtype:{'square'  'square'  'rt'}
```

The first field "**EEG.epoch.event**" contains the indices of all dataset events that occurred during this epoch. The fields "**EEG.epoch.eventtype**", "**EEG.epoch.eventposition**" and "**EEG.epoch.eventlatency**" are arrays containing values for each of the events (**EEG.epoch.event**) that occurred during the epoch. Note that the latencies in "**EEG.epoch.eventlatencies**" have been recomputed in units of milliseconds with respect to the epoch time−locking event. When there is only one event in an epoch, the epoch table is more readable. For instance, selecting "rt" type only in the pop−up window below (produced by menu item **Edit > Select events**) and pressing **OK** to create a new data set



Select events -- pop_selectevent()

Selection Field	Field Description To edit: Edit/Event info	Range (value list or real range "min < max") Ex: 2:4,5 OR 'COND1' OR 4.5 < 13	If set, non-range
Event indices			☐
type(s)	Type of the event	rt	☐
position(s)			☐
latency(s)	The latency at which the ev		☐
epoch(s)	no-description		☐

☐ Remove selected events

■ Remove epochs not referenced by any selected event

NOTE: Latency unit is milliseconds

Then typing

```
        >> EEG.epoch(1)
```

gives

```
ans =
             event:1
     eventlatency:1.0823e+03
    eventposition:2
        eventtype:'rt'
```

This means that epoch number "1" contains a single event of type "rt" at latency 1082.3 ms. Here, to extract an array of reaction times in every epoch, simply type

```
>> cell2mat({EEG.epoch.eventlatency}) % again using curly braces
```

Of course, making a new dataset as we just did simply to obtain an array containing the reaction time in each epoch is inefficient. EEGLAB function **eeg_getepochevent()** can achieve the same result more directly, even when the dataset structures contain multi−valued "**EEG.event. epoch**" and/or "**EEG.epoch.event**" arrays. For example, achieve the same result at above on the original sample dataset (with multiple events per epoch) by typing

```
>> eeg_getepochevent(EEG, 'rt', [], 'latency')
```

IV.4.5. Modifying the EEG structure

It is possible to add or directly modify EEG structure values from a script or the Matlab command line as long as one respects some simple rules:

1) If the EEGLAB option to process multiple data sets (selected via the **File > Maximize Memory** menu item) is **on,** then all available EEGLAB datasets are stored in the struct array **ALLEEG,** so the user should copy the current EEG dataset into **ALLEEG** each time the dataset is modified. Thus, after modifying the **EEG** structure (where **CURRENTSET** number is 1), to create a new dataset one might simply type

```
>> ALLEEG(2) = EEG;
>> CURRENTSET = 2;
```

This command might work as expected (if the new dataset is internally consistent and there is no previous dataset 2). However, in this case it would better to use the command **eeg_store()** which performs extensive dataset consistency checks before storing the modified dataset.

```
>> [ALLEEG EEG] = eeg_store(ALLEEG, EEG, 2);
```

If the option to maintain multiple data sets is off (**File > Maximize Memory** menu item), the variable ALLEEG is not used and EEG is the only variable that contains the dataset information. Any changes made by the user to the EEG structure are thus applied instantaneously and are irreversible. In this case, use **eeg_checkset()** to check the internal consistency of the modified data set.

```
>> EEG = eeg_checkset(EEG);
```

or

```
>> EEG = eeg_checkset(EEG, 'eventconsistency');
```

The second command above runs extra checks for event consistency (possibly taking some time to complete) and regenerates the **EEG.epoch** structures from the **EEG.event** information.

2) New EEG fields added to the **EEG** structure by users will not be removed by EEGLAB functions. For instance, extra information about a dataset might be stored in the user−added field:

```
>> EEG.analysis_priority = 1;
```

3) The following are the reserved variable names used by EEGLAB

```
EEG − the current EEG data set
ALLEEG − all the EEG data sets
CURRENTSET − the index of the current data set. Thus,
    >> EEG = ALLEEG(CURRENTSET);
LASTCOM − the last command used
ALLCOM − Matlab commands issued from the EEGLAB menu
during this EEGLAB session
```

Note that EEGLAB does not use global variables. The above variables are ordinary variables in the global Matlab workspace. All EEGLAB functions (with the exception of the main interactive window function **eeglab()** and a few other display functions) process one or more of these variables explicitly as input parameters.

We have tried to make EEG structures as simple and transparent as possible so that advanced users can use them to efficiently process their EEGLAB datasets.

IV.5. Sample scripts involving multiple datasets

Using EEGLAB data processing functions may involve understanding some subtleties about how they work and are called. Users should read carefully the documentation provided with each function (and even occasionally look in the script files to see exactly how data processing is performed, if desired).

IV.5.1. Performing time−frequency decompositions on multiple datasets

For example, when computing event−related spectral perturbation (ERSP) transforms for sets of data epochs from two or more experimental conditions, the user may want to subtract the same (log) power baseline in all conditions. The **pop_timef()** function and the toolbox **timef()** function it calls both return spectral baseline values that can be used in subsequent **timef()** computations. For instance, assuming that three sets of data epochs from three experimental conditions have been stored for each of 10 subjects in EEGLAB dataset files named 'subj[1:10]data[1:3].set' in directory '/home/user/eeglab ' and that the three datasets for each subject contain the same ICA weights, the following Matlab code would plot the ERSPs for the three conditions using a common spectral baseline for each of the 10 subjects:

```
eeglab;        % Start eeglab
figure;        % Open a blank Matlab figure
```

```
for S = 1:10  % For each of the ten subjects
  for s =1:3   % Read in three existing EEGLAB datasets
     setname = ['subj' int2str(S) 'data' int2str(s) '.set']; % Build dataset name
     EEG = pop_loadset(setname,'/home/user/eeglab/');  % Load the dataset
     [ALLEEG EEG] = eeg_store(ALLEEG, EEG, s);
     % Store the dataset in ALLEEG

     [ersp,itc,powbase{s}] =pop_timef( ALLEEG(s), ...
                    0, 1, [−100  600], 0, 'plotitc', 'off', 'plotersp', 'off' );
     % Run simple timef() for each dataset − Note curly braces after powbase
     % No figure is created because of options 'plotitc', 'off', 'plotersp', 'off'
   end  % End condition

   % Average the baseline spectra from the three conditions
   mean_powbase = mean([powbase{1}; powbase{2}; powbase{3}], 1);

   % Recompute and plot the ERSP transforms using the same baseline
   figure; % Create a new figure (optional figure('visible', 'off'); would create
            % an invisible figure)
   for s = 1:3  % For each of the three conditions
      sbplot(1,3,s);    % Select a plotting region
      pop_timef( ALLEEG(s), 0, 1, [−100  600], 0, 'powbase', mean_powbase, ...
                 'title', ['Subject ' int2str(S)]); % Compute ERSP using
mean_powbase
   end % End condition plot
   plotname = ['subj' int2str(S) 'ersp' ];  % Build plot name
   eval(['print −depsc ' plotname]);  % Save plot as a color .eps file
   close;  % Close the Matlab figure
end % End subject
```

The computation above is repetitive, and may be time consuming if there are many epochs in each dataset. Therefore it may be best performed by an EEGLAB Matlab script. Writing scripts using EEGLAB functions makes keeping track of data parameters and events relatively easy, while maintaining access to the flexibility and power of the Matlab signal processing and graphics environment.

Notes:

- Normally, the user might want to accumulate and save the ERSPs and other variables returned by **timef()** above to make possible quantitative comparisons between subjects.
- In the current version of EEGLAB the cross−coherence function **crossf()** can calculate significance of differences between coherences in two conditions.
- In the future, **timef()** will be extended to compare multiple ERSP and ITC transforms directly.
- The same type of iterative normalization (illustrated above) may be applied for the "baseamp" parameter returned by **pop_erpimage()**.

IV.5.2. Creating a scalp topography animation

The first solution to create scalp topography animations is to use the toolbox function **eegmovie()** from the command line. A second solution is to dump a series of images to disk, then to assemble them into a movie using an other program. For instance, type in

```
counter = 0;
for latency = −100:10:600 %−100 ms to 1000 ms with 10 time steps
   figure; pop_topoplot(EEG,1,latency, 'My movie', [] ,'electrodes', 'off'); % plot
   print('−djpeg', sprintf('movieframe%3d.jpg', counter)); % save as jpg
   counter = counter + 1;
end;
```

Then use " ***% convert movieframe*.jpg mymovie.mpg***" under Unix to assemble the images into a movie.

IV.5.3. Saving the ICA weight and sphere matrix of the current dataset

To save, the ICA weight and the sphere matrix of the current dataset as text files, type in

```
W = EEG.icaweights;
S = EEG.icasphere;
save −ascii icaweights.txt W
save −ascii icasphere.txt S
```

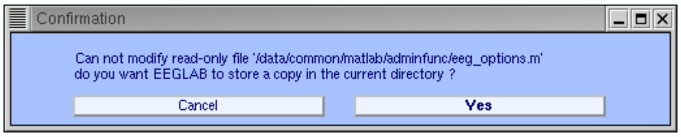
To save the a weight matrix that includes sphering (i.e. the global ICA weights*sphere matrix), type

```
WS = EEG.icaweights * EEG.icasphere;
save −ascii icaweights.txt WS
```

A1. Options to Maximize Memory and Disk Space

A1.1 Maximize memory menu

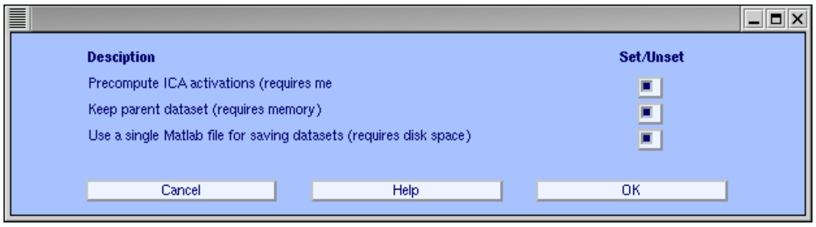
This section is intended for users who use large EEGLAB datasets and need to optimize their use of main memory (RAM) and disk space. To modify the relevant EEGLAB options, select **File > Maximize memory**. If you cannot modify the file **eeg_options.m** in the toolbox distribution, the window below will pop up.


Confirmation

Can not modify read-only file '/data/common/matlab/adminfunc/eeg_options.m'
do you want EEGLAB to store a copy in the current directory ?

Simply press **Yes**. A copy of **eeg_options.m** will be stored in the local directory and modified when you change any of the EEGLAB options. (**Note:** For EEGLAB to use this file, the current directory (.) must appear BEFORE the EEGLAB toolbox directory in the Matlab path; see **path()**). If the original **eeg_option.m** file is writable, EEGLAB will modify it instead.

If the original distribution copy of the options file is modified, the new options will affect all EEGLAB sessions using that file. If, however, EEGLAB (or the user from the Matlab or shell commandline) copies the options file to the current working directory, then only EEGLAB sessions having this directory in their Matlab path (before the EEGLAB distribution directory) will follow the modified options. Now the following window will pop up.



Desciption	Set/Unset
Precompute ICA activations (requires me	■
Keep parent dataset (requires memory)	■
Use a single Matlab file for saving datasets (requires disk space)	■

If the top option is set, all the ICA activitations (time courses of activity) will be pre−computed each time the ICA weights and sphere matrices are specified or changed. This may nearly double the main memory (RAM) required to work with the dataset. Otherwise, ICA activitations will be computed by EEGLAB only as needed, e.g. each time EEGLAB calls a function that requires one or more activations. In this case, only the activations needed will be computed, thus using less main memory.

When the middle option is set, EEGLAB can hold in memory more than one dataset at a time. New datasets are created by most of the operations called under the **Tools** menu. With this option set, users can undo dataset changes immediately by going back to working with the parent (previous) dataset. Processing multiple datasets may be necessary when comparing datsets from separate subjects or experimental conditions.

If the bottom option is set, all dataset information, including the data, is saved in a 64−bit double−precision Matlab .mat−format file with extension **.set** or **.sets**. If this option is unset, EEGLAB saves datasets using two files: one **.set** or **.sets** file containing dataset parameters and event information, plus a second **.fdt** file containing the data in 32−bit float format (with little−endian byte order). Unsetting this option minimizes the disk space required to save large numbers of EEGLAB datasets.

A1.2. The icadefs.m file

Using a text editor, you should edit file "icadefs.m" in the distribution before beginning to use EEGLAB. This file contains EEGLAB constants used by several functions. In this file, you may:

- Specify the filename directory path of the EEGLAB tutorial. Using a local copy of this EEGLAB tutorial (available online at sccn.ucsd.edu/eeglab/eeglabtut.html requires a (recommended) **Tutorial Download**.

  TUTORIAL_URL = 'http://sccn.ucsd.edu/eeglab/eeglab.html'; % online version

- Reference the fast binary version of the **runica()** ICA function "**ica**" (see the **Binica Tutorial**).This requires another (recommended) download from **http://sccn.ucsd.edu/eeglab/binica/**

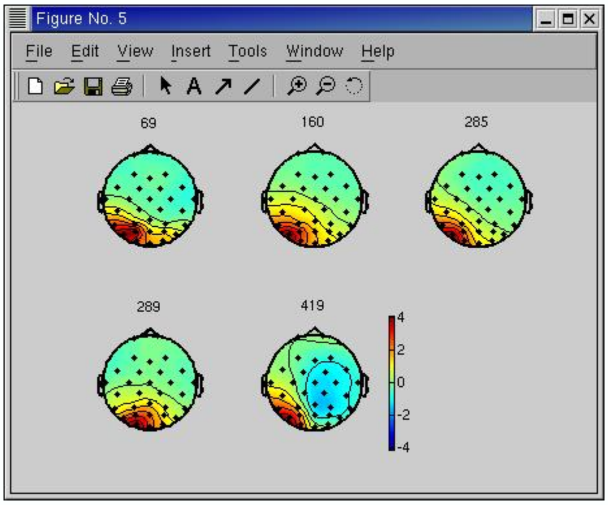
ICABINARY = 'ica_linux2.4'; % <=INSERT name of ica executable for binica.m

You can also change the colors of the EEGLAB interface using other options located in the **icadefs** file.

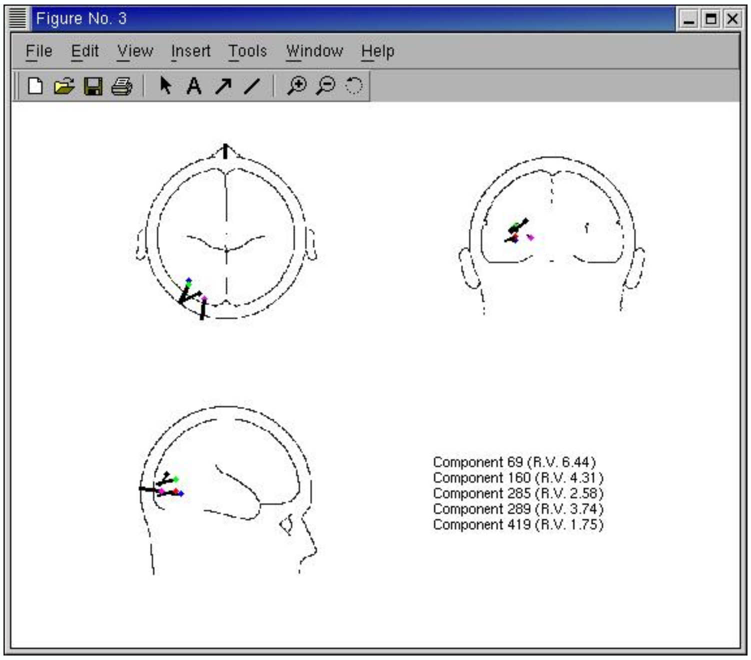
A2. Equivalent dipole source localization of independent components using BESA

A2.1. Component localization by equivalent dipoles: A simple example

Equivalent dipole source localization by BESA (MEGIS Software GmbH, Munich) can be applied to independent component maps under EEGLAB. The following window shows five independent component scalp maps from five different subjects. Each of these components was labelled by the experimenters as representing a left−occipital alpha rhythm generator.



BESA dipole localization applied to these component maps, accounted for each map with a single equivalent dipole in a three−shell spherical model with low residual variance (R.V.) Note that unlike in many BESA analyses, we rarely need to ignore signals at some scalp electrodes since ICA typically isolates artifactual signals into separate independent components, and independent components likely account mainly for activity with a connected synchronous cortical domain whose scalp projection should match that of a single equivalent dipole. (Note: This does not mean, however, that some independent components may be generated by tightly linked bilateral activity in domains joined through e.g. corpus callosum. In this case, the source may often be well modelled by two symmetricly placed equivalent dipoles).



Component 69 (R.V. 6.44)
Component 160 (R.V. 4.31)
Component 285 (R.V. 2.58)
Component 289 (R.V. 3.74)
Component 419 (R.V. 1.75)

A2.2. How to localize components using BESA

BESA is a commercial software (www.besa.de) developed by Michael Scherg and colleagues for localizing brain source equivalent dipoles using a three−shell spherical model.

We have developed Matlab functions to automatically export/import and display BESA dipole information. To localize components, you must run BESA 2.x, 3.x, 99 or BESA 2000 Since BESA 99 and 2000 do not support macro files, if you use these versions you must also download BESA 3.0 (the previous version running macros) from the BESA web site (see below) .

BESA 3.0 installation for BESA 2000 and BESA 99 users

1 − Download BESA30 DOS−Window Upgrade from 2.x from the BESA web site (**even if you don't have BESA 2.x**)

2 − Install the hardlock driver for BESA 2000 (on the BESA 2000 CD. Note: you must usually go through the whole installation of BESA 2000 to install the hardlock driver).

3 − Download the BESA30 for BESA2000−Hardlock file and **place it in the BESA 3.0 directory** (if you only have a BESA 99 license, you should download BESA30 for BESA99−EEG−Hardlock or BESA30 for BESA99−MEG+EEG−Hardlock).

4 − Click on the file downloaded in (3) to start BESA3.0

5 − The BESA3.0 manual is not provided on the Internet. We use the BESA 2.2 Manual and find that most of the commands also work in version 3.0.

The EEGLAB BESA functions can also be used with BESA 2.x. However, BESA 2.x cannot import more than 66 channels, and has some restrictions about path lengths under Windows, we strongly recommend the use of BESA 3.0.

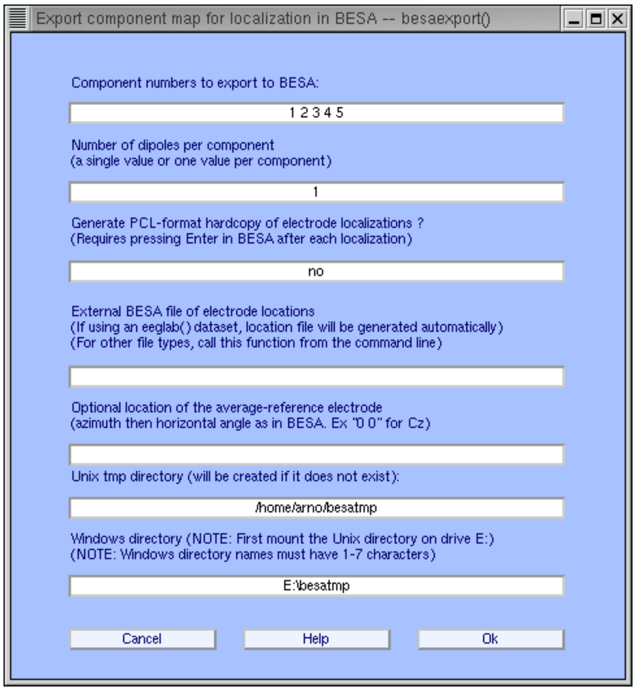
A2.3. Exporting component information

You must first restart EEGLAB with the BESA−function menu included by typing

```
>>  eeglab besa
```

Note that this call will not erase your current dataset(s), but will simply add a new item to the **Tools** menu: **Tools > Localize dipoles using BESA**.

After ICA components have been computed, select menu item **Tools > Localize dipoles using BESA > Export components to BESA**. This calls the function besaexport(). The following window will appear:


Export component map for localization in BESA -- besaexport()

Component numbers to export to BESA:
1 2 3 4 5

Number of dipoles per component
(a single value or one value per component)
1

Generate PCL-format hardcopy of electrode localizations ?
(Requires pressing Enter in BESA after each localization)
no

External BESA file of electrode locations
(If using an eeglab() dataset, location file will be generated automatically)
(For other file types, call this function from the command line)

Optional location of the average-reference electrode
(azimuth then horizontal angle as in BESA. Ex "0 0" for Cz)

Unix tmp directory (will be created if it does not exist):
/home/arno/besatmp

Windows directory (NOTE: First mount the Unix directory on drive E:)
(NOTE: Windows directory names must have 1-7 characters)
E:\besatmp

In the first edit box, enter the indices of the components to be localized using BESA. The other edit boxes are self explanatory. In case you choose to export the electrode locations from EEGLAB (by leaving blank the 4th edit box), it is very important to check that the polar coordinates have not been shrunk by any other program (e.g., the Fz radius value should be 0.25). If you imported a file with 3−D coordinates into EEGLAB, that should not be an issue. Note that under Windows the last edit box will not appear. After pressing **OK**, the following text will be displayed on the command line:

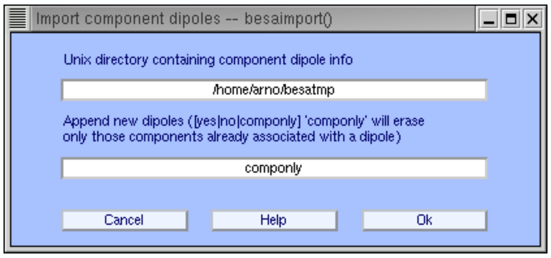
```
Accessing directory...
Exporting electrode locations ...
Exporting component maps...
Exporting parameter file...
Writing macro file...
NOW RUN BESA, PRESS "A" (for automatic processing) and enter filename
"E:\besatmp\macro"
```

Now, run BESA on your Windows machine. In BESA, type "**A**" and then enter the filename given above ("**E:\besatmp\macro**"). Dipole localization will be performed automatically. Since the number of characters in each BESA macro file is limited, the EEGLAB Matlab function generates many macro files that are called by parent macro files. In this way, up to 900 components can be localized at a time. Also note that minimizing the BESA window will greatly speed up the computation.

Occasionally BESA will not optimize the dipole location (e.g., when residual variance remains 98% or more). If this happens, simply rerun the macro corresponding to the badly localized component (for instance **macro10.aut** for component 10).

A2.4. Importing BESA component locations

Select menu item **Tools > Localize dipoles using BESA > Import dipoles from BESA** (calling the function **besaimport()**). The following interactive window will pop up.


Import component dipoles -- besaimport()

Unix directory containing component dipole info

/home/arno/besatmp

Append new dipoles ([yes|no|componly] 'componly' will erase only those components already associated with a dipole)

componly

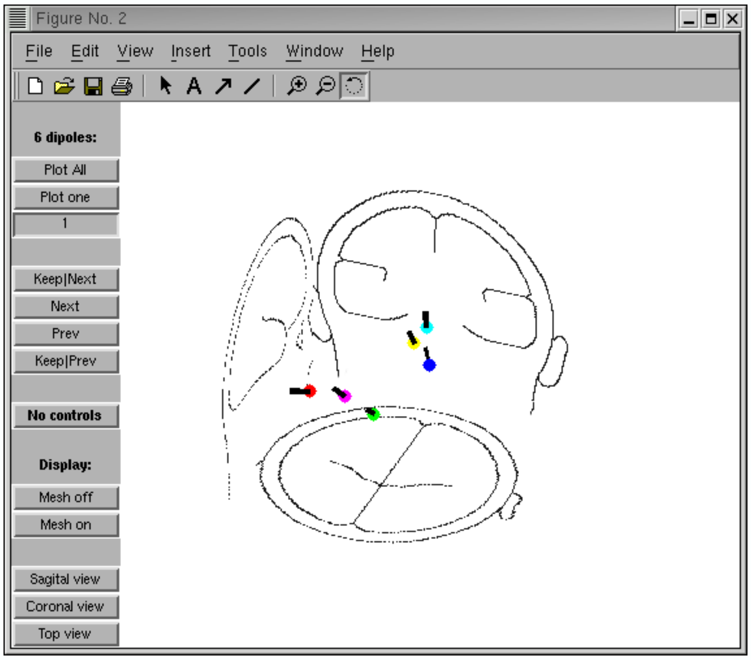
In the upper edit box, you must provide the location of the directory in which the dipole information has been saved by BESA. In the lower edit box, you can either (1) import dipoles and erase all previous information ("**no**" option), (2) append the newly computed information ("**yes**" option), or (3) only import dipoles for components that do not have yet associated dipoles ("**componly**" option). Press **OK** and the function will return information about the components that have been imported:

```
Reading component 1 (1 dipoles) residual variance 2.09 %
Reading component 2 (1 dipoles) residual variance 15.67 %
Reading component 3 (1 dipoles) residual variance 1.94 %
Reading component 4 (1 dipoles) residual variance 1.23 %
Reading component 5 (1 dipoles) residual variance 3.49 %
Reading component 6 (1 dipoles) residual variance 12.88 %
```

Note: We usually only trust dipole locations with low residual variance (e.g., below 5%, though this figure may vary with number of channels, montage, etc.).

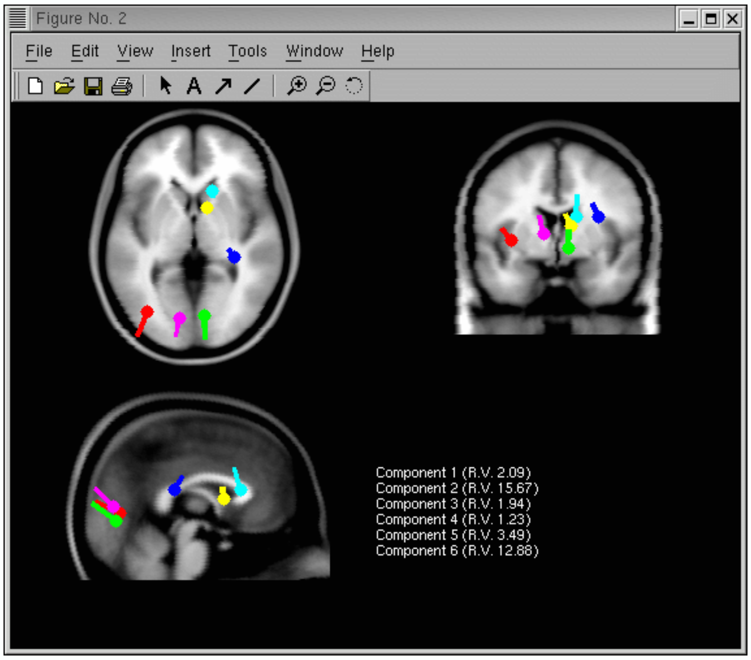
A2.5. Visualizing dipole locations

There are four menu items that visualize equivalent dipole locations. They all call the same function besaplot() with different options. Here, we illustrate two of these four menu items: First, **Tools > Localize dipoles using BESA > Plot dipoles on BESA head**.



Here, the dipoles are plotted in 3−D (Note: Use the buttons on the bottom−left to select a specific view). Note that the background images were captured from BESA and the scaling was adjusted manually using visual fitting of a sphere mesh and of the BESA images (press the "**Mesh on**" button (lower left) to display the mesh sphere). Note that the barlength of the dipole varies with the absolute component amplitude.

We also fit the head sphere model to an average−brain MRI average images (available for download from the ICBM project). MRI images were first shrunk along the X or Y axis to match the 3−D sphere mesh. We then changed the aspect ratio of the axis so that the MRI images' original aspect ratio was preserved (and the sphere mesh became an ellipsoid). A dipole location summary, plotted on the top of the MRI image, can be shown using menu item **Tools > Localize dipoles using BESA > Plot dipoles on BESA head**.



Finally, by calling the plotting function from the command line

```
>> besaplot(EEG.sources);
```

many more options can be specified and one or more equivalent dipoles can be plotted individually for each component.

A2.6. Miscellany

Using other dipole localization software

You may use the **besaexport()** function to export the electrode localization file and the component activations, then try to localize dipoles using some other localization software. If you succeed in doing this, or if you write dedicated functions for this, please contact us.

Localizing dipoles from ERP averages or continuous EEG

Because of the theoretical and practical advantages of using ICA for source localization, we have only implemented equivalent dipole localization for independent components. However, localizing ERP or EEG time series would simply involve exporting the data to BESA. For instance, to export an ERP raw data file that BESA can read, type in

```
>> ERP = mean(EEG.data,3);
>> save −ascii ERP.raw ERP
```

You may also use the electrode location file generated by **besaexport()**.