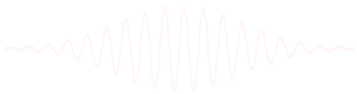


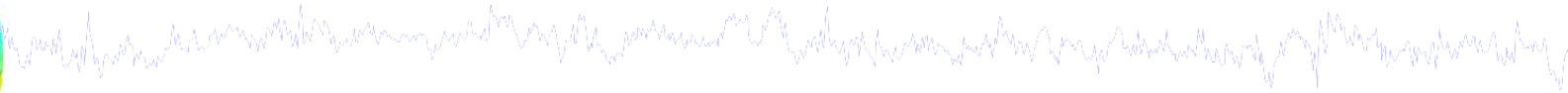
STUDY scripting



- 1. Build a STUDY from command line**
- 2. STUDY structure details**
- 3. Cluster ERP image**
- 4. Using STUDY raw data**



Build a STUDY



Most important option:

- Allows only one dataset to be loaded at once.
- Most STUDYs are too big to have all data loaded at once.

```
% Open eeglab:
```

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

```
% Set memory options:
```

```
pop_editoptions( 'option_storedisk', 1, 'option_savetwofiles', 1,...  
    'option_saveica', 1, 'option_single', 0, 'option_memmapdata',...  
    0, 'option_computeica', 1, 'option_scaleicarms', 0,...  
    'option_rememberfolder', 1);
```

```
% saves a file 'eeg_options.m' to your current working directory
```

```
% Initialize EEGLAB/STUDY variables:
```

```
STUDY = []; CURRENTSTUDY = 0; ALLEEG=[]; EEG=[]; CURRENTSET=[];
```

Build a STUDY



% Faster alternative to building a STUDY manually

% Example STUDY: 13 subjects, 3 conditions

% Define variables:

```
basedir = 'C:\...\EEGLAB_Workshop\STUDY\';
```

```
setnames = {'Memorize.set', 'Ignore.set', 'Probe.set'};
```

```
conds = {'Memorize', 'Ignore', 'Probe'};
```

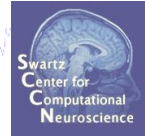
```
subjs = {'S01', 'S02', 'S03', 'S04', 'S05', 'S06', 'S07', ...  
'S08', 'S09', 'S10', 'S11', 'S12', 'S13'};
```

```
studyname = 'Sternberg';
```

```
taskname = 'Sternberg';
```

```
savename = 'mystern.study';
```

Load dataset info from commandline

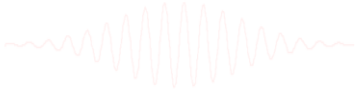


```
% Now loop through subjects and add to the STUDY:
index = 1; % initialize STUDY index
for subj = 1:length(subjs) % for each subject
    for cond = 1:length(setnames) % for each condition
        datset = [basedir,subjs{subj},'\',setnames{cond}];% concatenate strings
        [STUDY ALLEEG] = std_editset( STUDY, ALLEEG,'name',studyname,'task', taskname,...
            'commands',{{'index',index,'load',datset}, {'index',index,'subject',subjs{subj}},...
            {'index',index,'session',1}, {'index',index,'group',1},...
            {'index',index,'condition',conds{cond}}, {'index',index,'dipselect',.12}},...
            'inbrain','on', 'updatedat','off',...
            'savedat', 'off', 'filename', [basedir,savename]);
        index = index + 1; % update set file index
    CURRENTSTUDY = 1; EEG = ALLEEG; CURRENTSET = [1:length(EEG)]; % reassign EEGLAB variables
    [STUDY, ALLEEG] = std_checkset(STUDY, ALLEEG); % check STUDY for consistency
    end;
end;
eeglab redraw
```

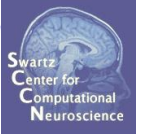
STUDY scripting



- 1. Build a STUDY from command line**
- 2. STUDY structure details**
- 3. Cluster ERP image**
- 4. Using STUDY raw data**

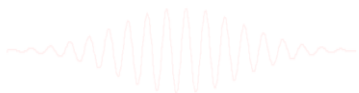


Understanding STUDY structure

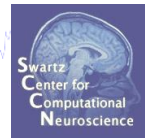
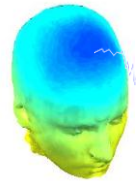


How are IC and subject/dataset

information stored in the STUDY structure?



STUDY structure



STUDY =

```
    name: 'Sternberg'  
    task: 'Sternberg'  
datasetinfo: [1x39 struct]  
    notes: ''  
filename: 'stern.study'  
filepath: 'C:\Users\julie\Documents\Workshops\Mallorca\STUDY'  
history: [1x7332 char]  
subject: {1x13 cell}  
    group: {''}  
    session: []  
condition: {'ignore' 'memorize' 'probe'}  
    setind: [3x13 double]  
    etc: [1x1 struct]  
preclust: [1x1 struct]  
    cluster: [1x1 struct]  
changrp: [1x71 struct]  
    saved: 'yes'
```

>>

Subject info in STUDY structure



**Gives information
for each dataset**

```
>> STUDY.datasetinfo
```

```
ans =
```

```
1x39 struct array with fields:
```

```
filepath
```

```
filename
```

```
subject
```

```
session
```

```
condition
```

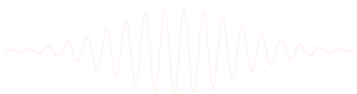
```
group
```

```
index
```

```
comps
```

**Each *dataset*
(NOT each subject)
has a unique
index (i.e., when
>1 condition, subjects
are represented > 1**

```
>>
```



Understanding STUDY structure



One cluster:

```
>> STUDY.cluster
```

```
ans = 26 = # of clusters
```

```
1x26 struct array with fields:
```

```
>> STUDY.cluster(6)
```

6 = cluster number

```
ans =
```

3 = # conditions
dataset indices

IC indices

3 conditions
STUDY.design indices

67 = # of channels

scalp map polarity

```

name: 'Cls 6'
sets: [3x17 double]
comps: [14 10 6 13 6 7 8 12 17 13 11 14...
        9 5 10 12 5]
parent: {'Parentcluster 1'}
child: []
preclust: [1x1 struct]
allinds: {3x1 cell}
setinds: {3x1 cell}
algorithm: {'Kmeans' [25]}
topo: [67x67 double]
topox: [67x1 double]
topoy: [67x1 double]
topoall: {1x17 cell}
topopol: [1 1 1 1 1 1 1 1 1 -1 1 1...
          1 1 1 1 1]

```

Understanding STUDY structure



```
>> STUDY.cluster(6)
```

```
ans =
```

```
    name: 'Cls 6'  
    sets: [3x17 double]  
    comps: [14 10 6 13 6 7 8 12 ... ]  
    parent: {'Parentcluster 1'}  
    child: []  
    preclust: [1x1 struct]  
    allinds: {3x1 cell}  
    setinds: {3x1 cell}  
algorithm: {'Kmeans' [25]}  
    topo: [67x67 double]  
    topox: [67x1 double]  
    topoy: [67x1 double]  
    topoall: {1x17 cell}  
    topopol: [1 1 1 1 1 1 1 1 1 ... ]
```

Component indices

```
>> STUDY.cluster(6).allinds
```

```
ans =
```

```
[1x17 double]
```

```
[1x17 double]
```

```
[1x17 double]
```

Which subject?

Which dataset(s)?

```
>> STUDY.cluster(6).allinds{1}
```

```
ans =
```

```
14 10 6 13 6 ...
```

```
>> STUDY.cluster(6).setinds{1}
```

```
ans =
```

```
1 2 3 4 5 ...
```

Index into STUDY.design

Understanding STUDY structure



Index into STUDY.design

```
>> STUDY.cluster(6).setinds{1}
```

```
ans =
```

```
1 2 3 4 5 ...
```

Current
design
index

```
>> STUDY.design(1).cell(2)
```

```
ans =
```

```
dataset: 5  
trials: {[1x266 double]}  
value: {'ignore' ''}  
case: 'S02'  
filebase: [1x60 char]
```

```
>> STUDY.datasetinfo(5)
```

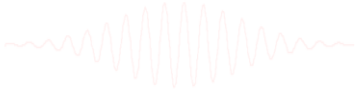
```
ans =
```

```
filepath: [1x53 char]  
filename: 'Ignore.set'  
subject: 'S02'  
session: 1  
condition: 'ignore'  
group: 1  
index: 5  
comps: [1x21 double]  
trialinfo: [1x266 struct]
```

STUDY scripting



- 1. Build a STUDY from command line**
- 2. STUDY structure details**
- 3. Cluster ERP image**
- 4. Using STUDY raw data**



Cluster ERP image



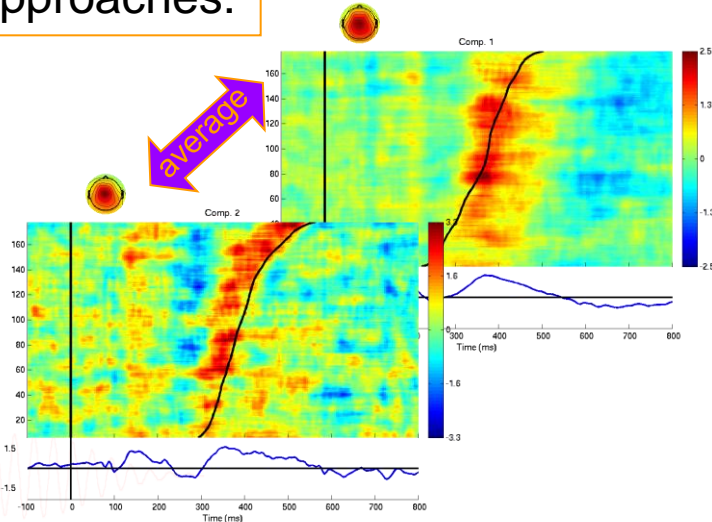
Purpose of ERP image:

- Observe single-trial dynamics of an IC activation (or power)

Purpose of *CLUSTER* ERP image:

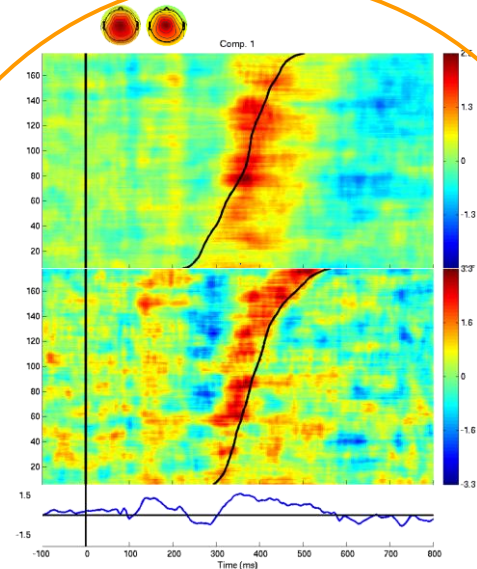
- Observe single-trial dynamics of multiple *matched* ICs from several subjects

Two approaches:



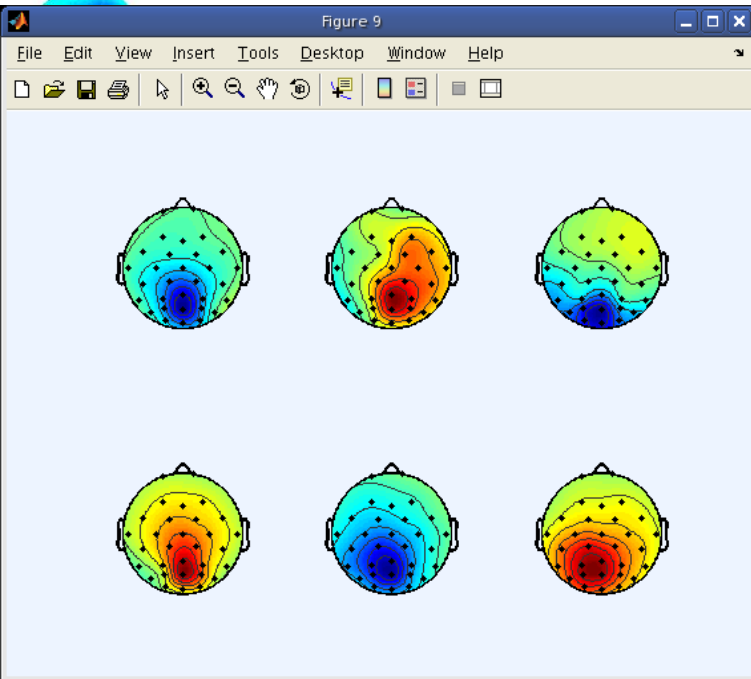
Average ERP images across ICs

OR

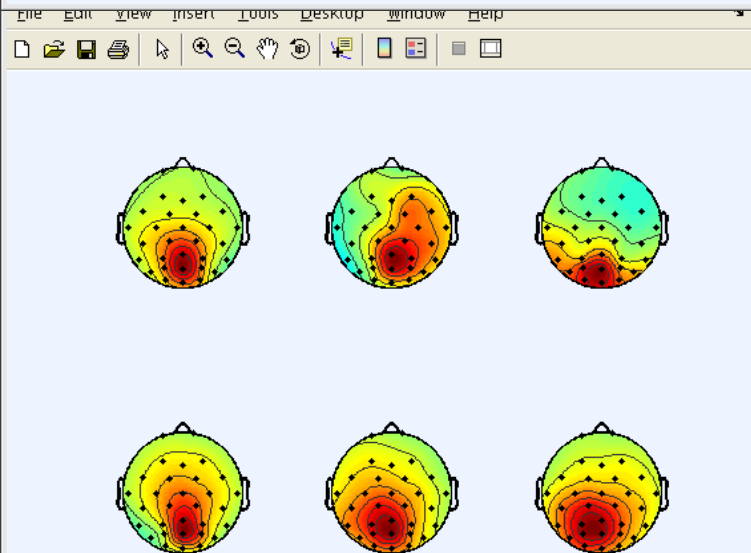


Merge trials across ICs

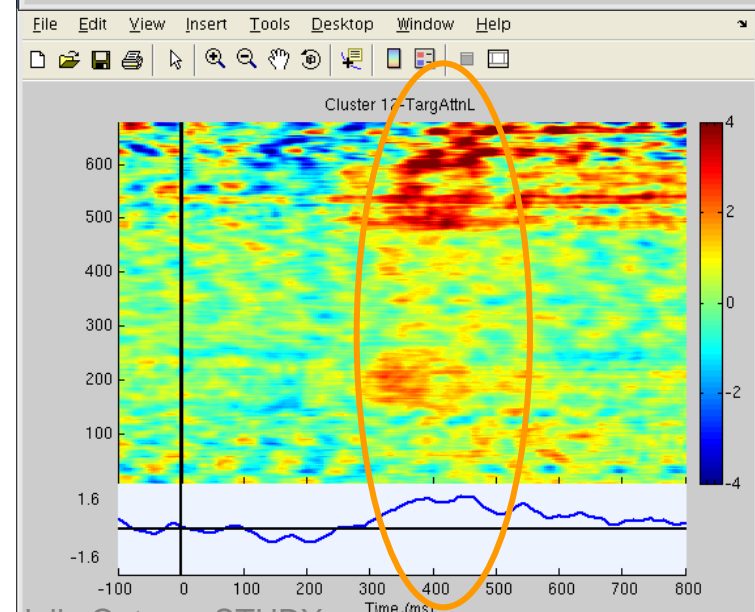
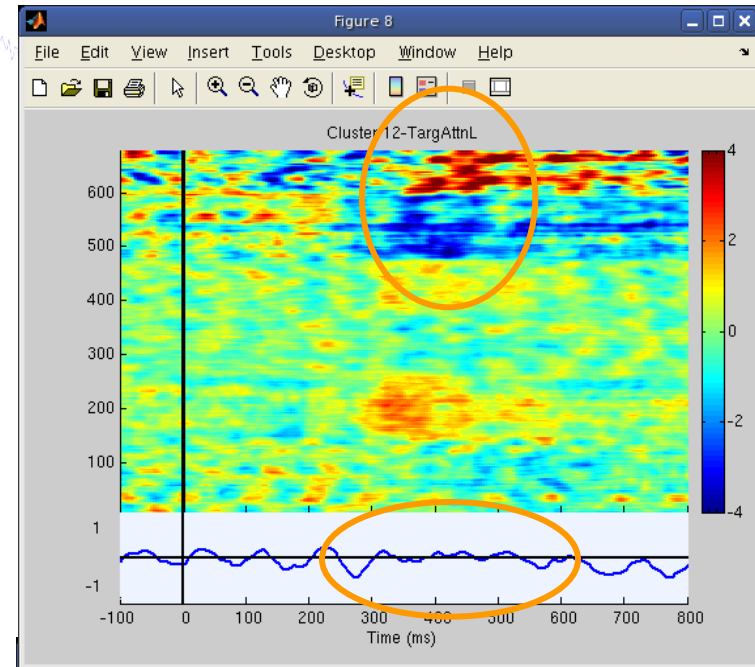
Cluster ERP image: match polarity



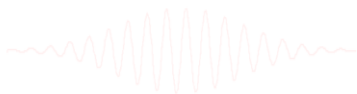
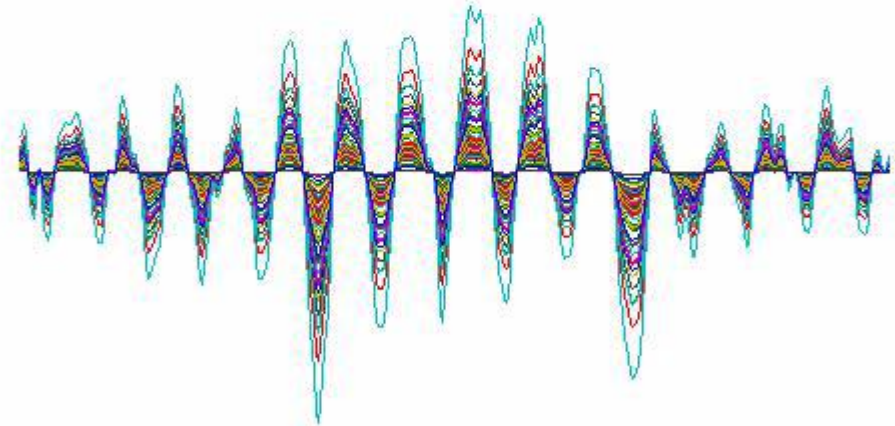
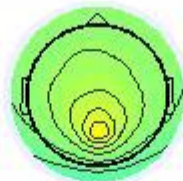
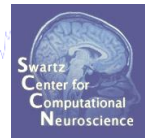
reversed polarities
reflect mismatched
scalp maps



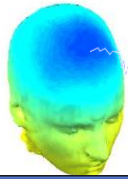
reorienting maps
and activations
gives a more
coherent picture



Movie of IC scalp map over time



Matching activation polarity



View and edit current component clusters -- pop_clustedit()

Study 'Attention': 181 of 181 components clustered

Select cluster to plot

- Cls 22 (4 ICs)
- Cls 23 (5 ICs)**
- Cls 24 (4 ICs)
- Cls 25 (7 ICs)

Plot scalp maps
Plot dipoles
Plot ERPs
Plot spectra
Plot ERSPs
Plot ITCs
Plot cluster properties

Params
Params
Params

Create new cluster
Rename selected cluster
Merge clusters

Save STUDY set to disk /home/julie/WorkshopSD2007/STUD

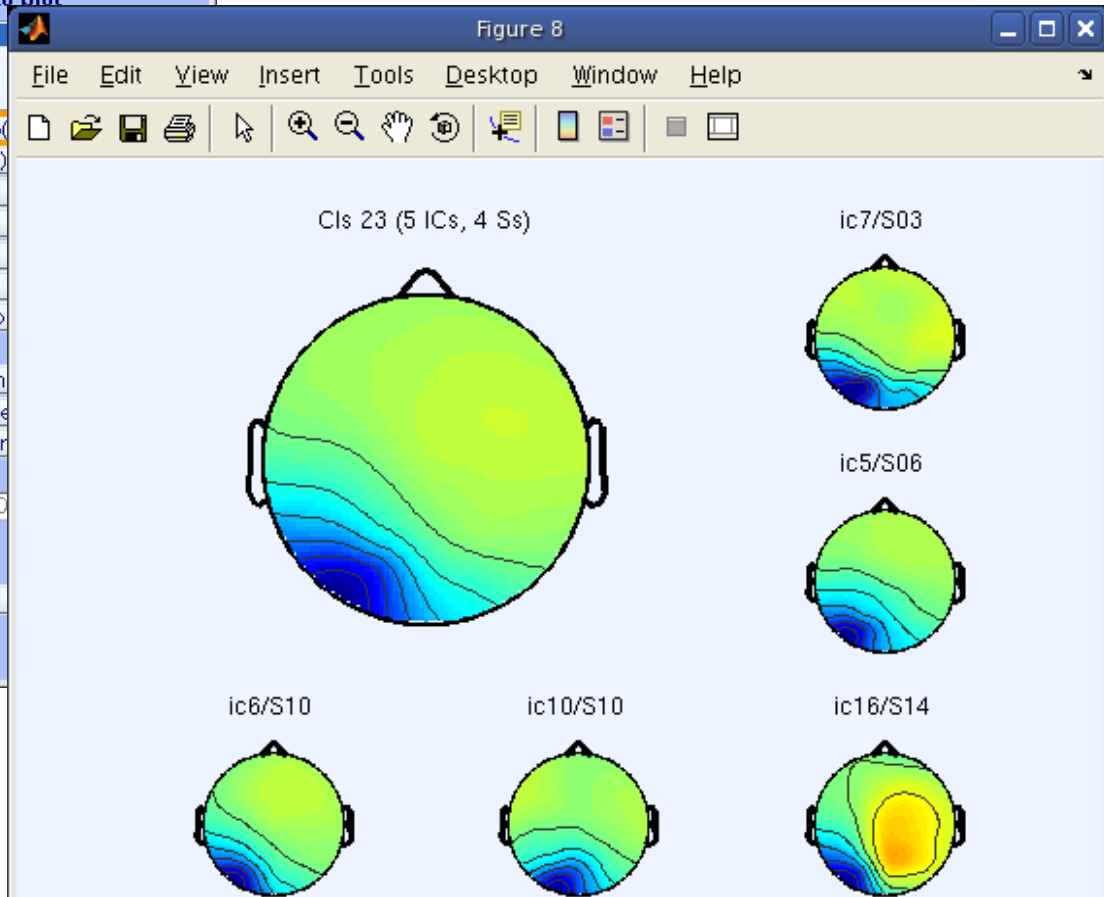
Cancel Help Ok

Select component(s) to plot

- All components
- S03 IC7
- S06 IC5
- S10 IC6

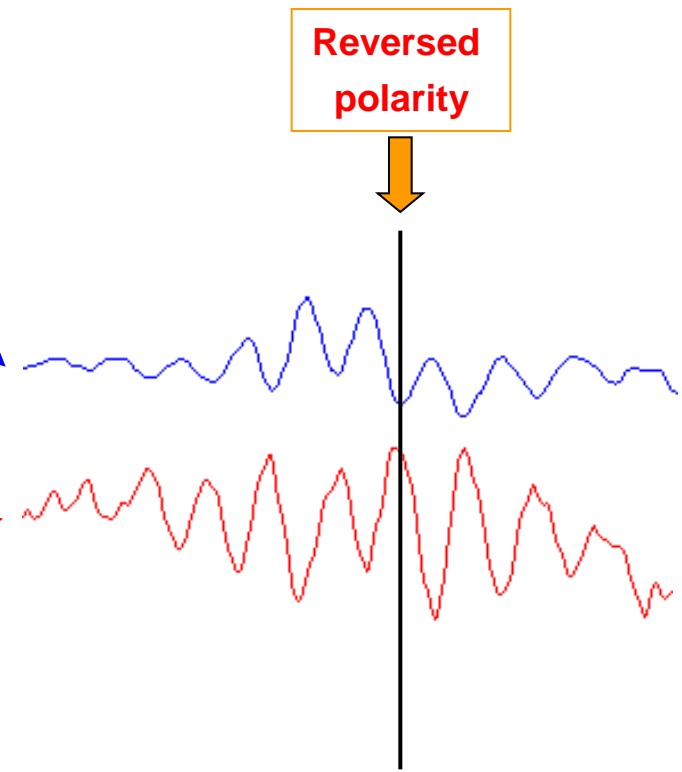
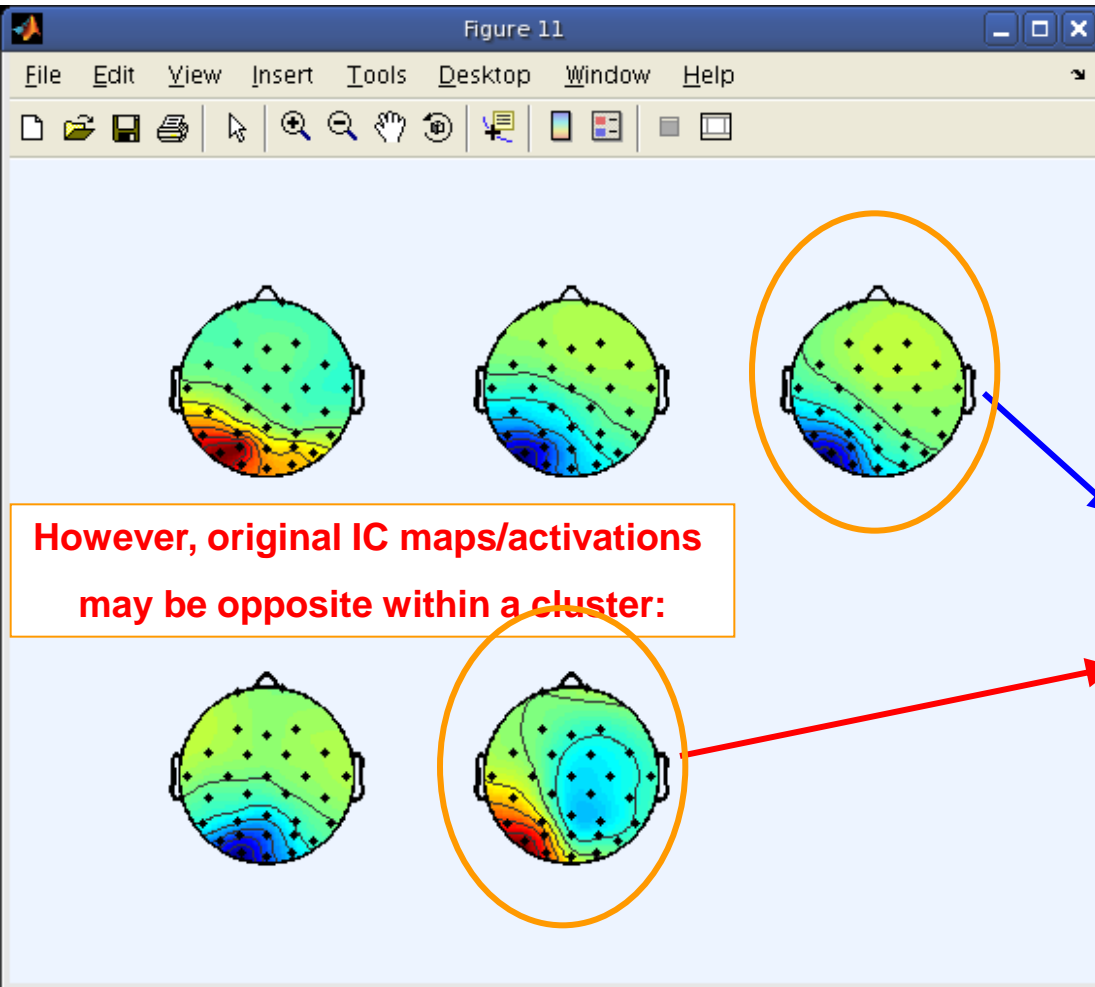
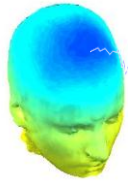
Plot scalp map(s)
Plot dipole(s)
Plot ERP(s)
Plot spectra
Plot ERSP(s)
Plot ITC(s)
Plot component pro

Reassign selected com
Remove selected outlie
Auto-reject outlier cor

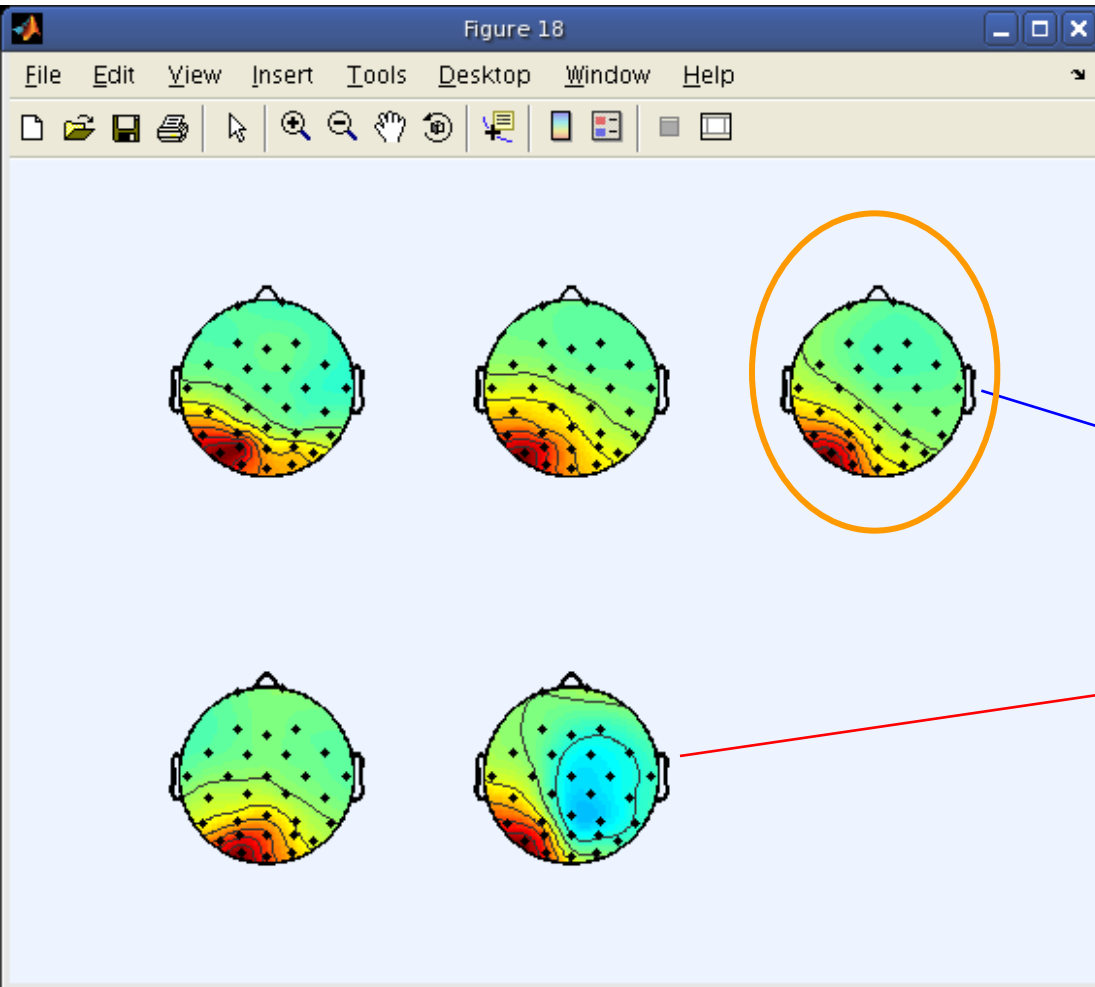


EEGLAB STUDY
matches polarities for you

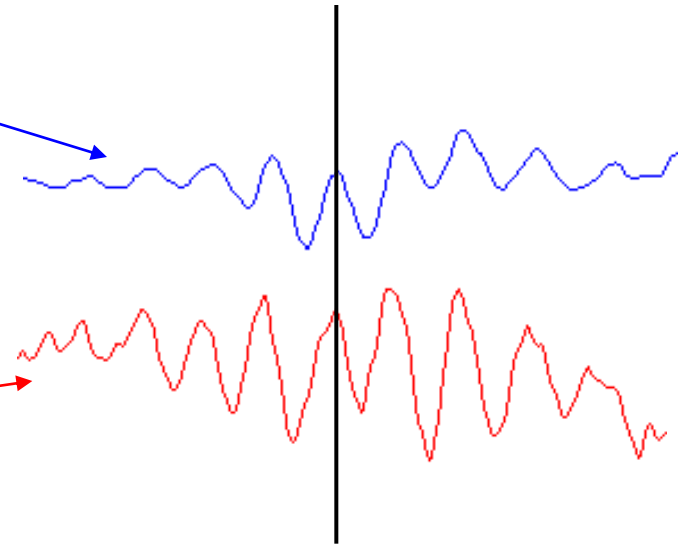
Matching activation polarity



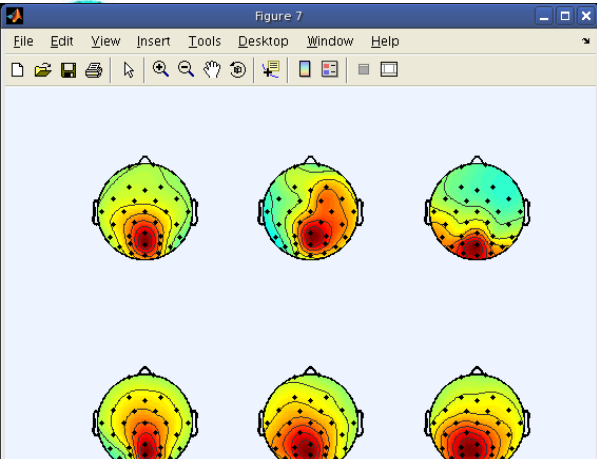
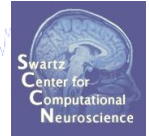
Matching activation polarity



**Reorient map AND
activation of
one IC to align**



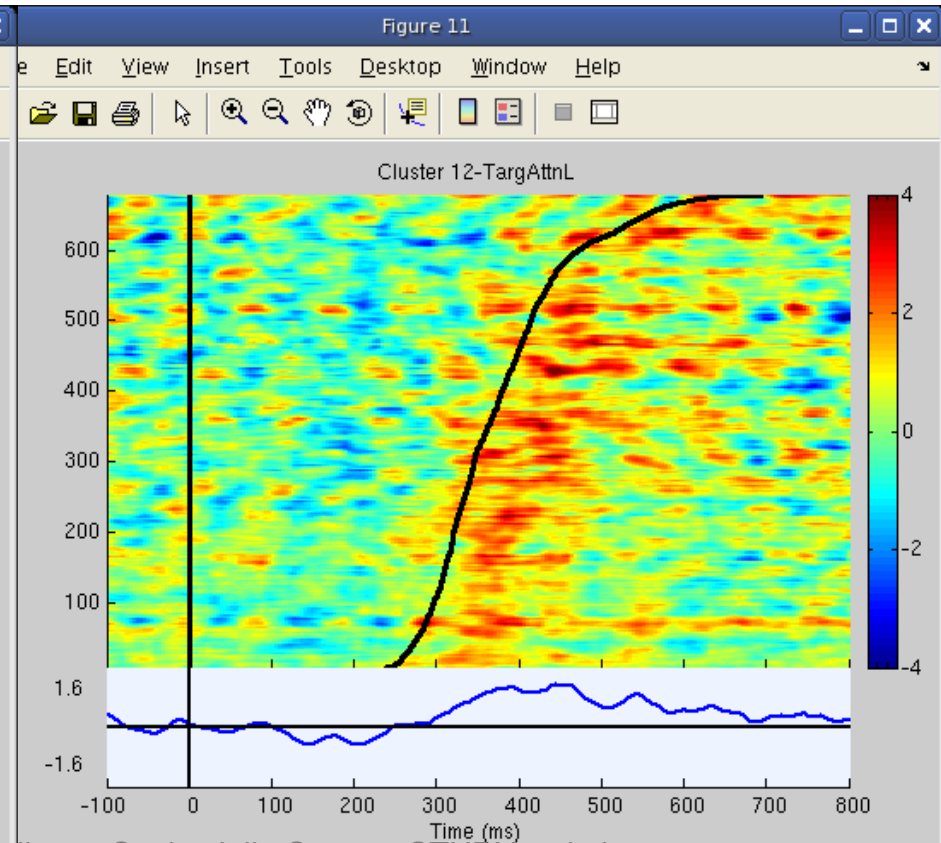
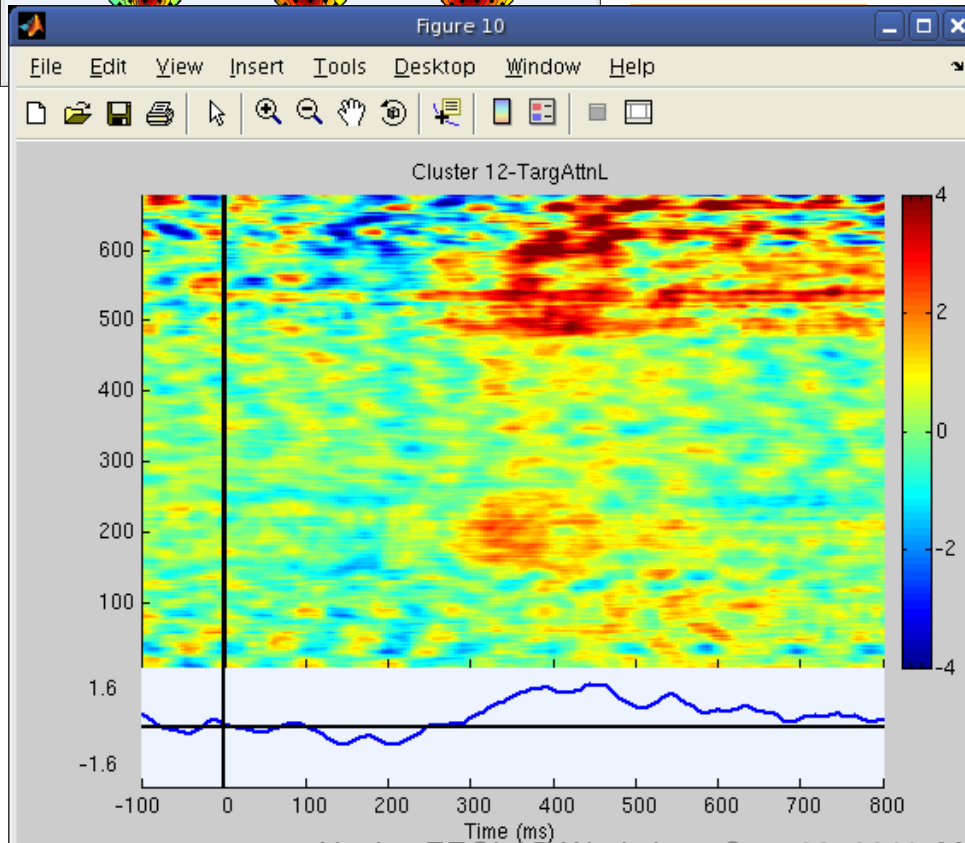
Cluster ERP image: RT sort



Consistent scalp maps



Sort cluster ERP image by response time

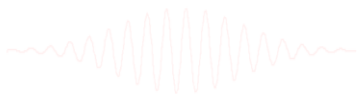


STUDY ERP image

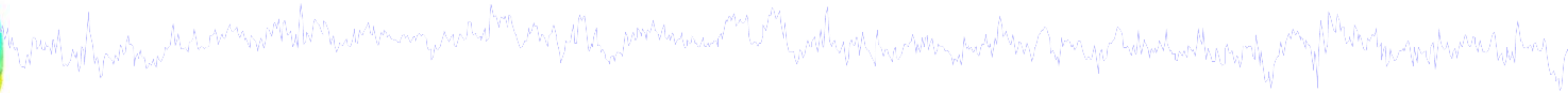


```
% plot all mean maps to get topo polarity:
STUDY = std_topoplot(STUDY,ALLEEG,'clusters',[2:length(STUDY.cluster)]);
design = 1;
clust = 7; % choose a cluster
cond = 3; % choose a condition (from STUDY.condition),
           % Probe is the only condition with a button press (RT)

(% requires memory options set to 'pre-calculate ica activations')
```



STUDY ERP image



```
% collect activations (correctly oriented) for all cluster ICs:
```

```
CURRENTSTUDY = 1; EEG = ALLEEG; CURRENTSET = [1:length(EEG)];
```

```
rts = []; allacts = zeros(1,length(ALLEEG(1).times),0);
```

```
for ic = 1:length(STUDY.cluster(clust).allinds{cond})
```

```
    design_idx = STUDY.cluster(clust).setinds{cond}(1,ic);
```

```
    setidx = STUDY.design(design).cell(design_idx).dataset;
```

```
    comp = STUDY.cluster(clust).allinds{cond}(1,ic);
```

```
    [ALLEEG EEG CURRENTSET] = pop_newset(ALLEEG, EEG, CURRENTSET,...
```

```
    'retrieve',setidx,'study',CURRENTSTUDY);
```

```
    for ep = 1:length(EEG.epoch)
```

```
        zlat = find(cell2mat(EEG.epoch(ep).eventlatency)==0);
```

```
        sortvar = [sortvar,EEG.epoch(ep).eventrt{zlat}];
```

```
    end;
```

```
    rmsuv = sqrt(mean(ALLEEG(setidx).icawinv(:,comp).^2)); % RMS at scalp
```

```
    acts = ALLEEG(setidx).icaact(comp, :, :) * rmsuv * STUDY.cluster(clust).topopol(ic);
```

```
    allacts(:, :, end+1:end+size(acts,3)) = acts;
```

```
end;
```

```
allacts = squeeze(allacts); % makes a (frames x trials) matrix
```

Only necessary
for sorting by or
plotting RT

STUDY ERP image



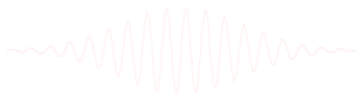
```
% PLOT activations:-----
```

```
clim = 2.5; % color limits
```

```
smoothby = 10; % Vertical smoothing across trials
```

```
figure; [outdata, outvar, outtrials, limits, axhndls, erp, amps, cohers, cohsig,  
ig, ampsig, outamps, phsangls, phsamp, sortidx, erpsig] = ...
```

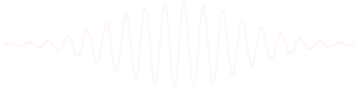
```
erpimage( allacts, sortvar, linspace(EEG.xmin*1000, ...  
EEG.xmax*1000, EEG.pnts), ['Cluster ', int2str(clust)], ...  
smoothby, 1, 'yerplabel', '', 'erp', 'limits', 'cbar', ...  
'caxis', [-clim clim], 'coher', [9 12 .01]);
```



STUDY scripting



- 1. Build a STUDY from command line**
- 2. STUDY structure details**
- 3. Cluster ERP image**
- 4. Using STUDY raw data**



Load data from commandline



**** Where is the raw data stored?**

Data for each subject is stored in the file path of that subject: `STUDY.datasetinfo(subj).filepath`

**** What is it called?**

File name format: `'design?_Subj_var1_var2.extension'`

extension = `'.ica*' or '.dat*' (for channel data)`

for example:

`design2_S04_memorize_3.icaerp % ERP data`

`design2_S04_memorize_3.icaersp % ERSP data`

`design2_S04_memorize_3.icaitc % ITC data`

`design2_S04_memorize_3.icaspec % Power spectrum data`

`design2_S04_memorize_3.icatopo % Scalp map data`

% Example of channel data file name:

`design2_S04_memorize_3.daterp % ERP data`

Load individual ERSPs



```
% load ERSP data for all ICs in a whole cluster at once:
```

```
clust = 5; % choose a cluster
```

```
cond = 1; % choose experimental condition
```

```
[STUDY logersp timevals logfreqs pgroup pcond pinter] = ...
```

```
std_erspplot(STUDY,ALLEEG,'clusters',clust, 'plotsubjects', 'off' );
```

```
% Check imported variables in workspace:
```

```
>> whos logersp
```

| Name | Size | Bytes |
|---------|-------------------------|--------------|
| Class | Attributes | |
| logersp | 3x1 3 conditions | 1983216 cell |

```
>> logersp
```

```
logersp =
```

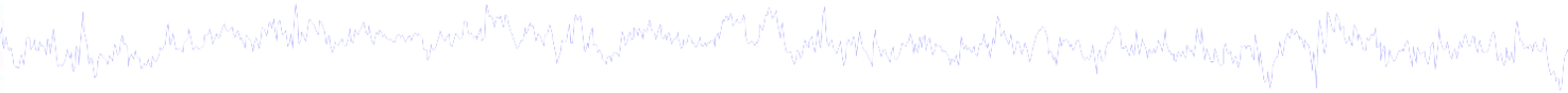
```
[72x153x15 single]
```

```
[72x153x15 single]
```

```
[72x153x15 single]
```

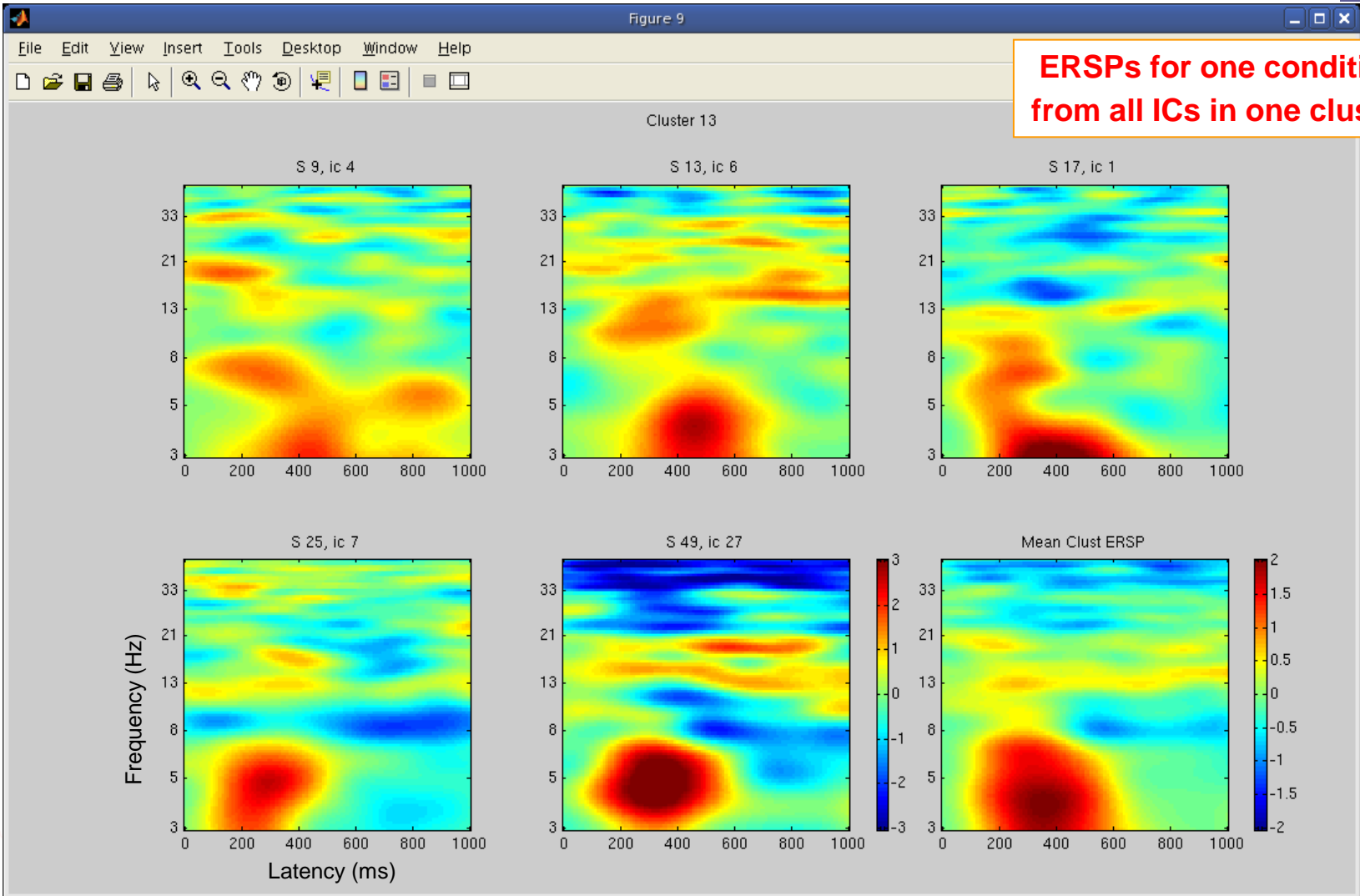
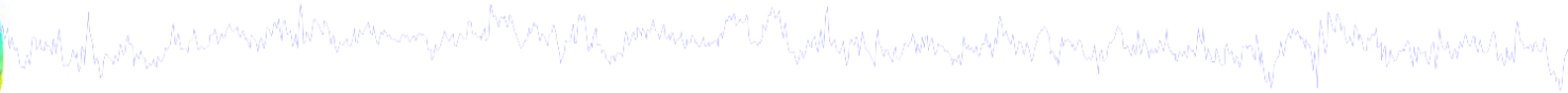
frequency x times x ICs (different subjects)

Plot individual ERSPs



```
% load ERSP data for all ICs in a single cluster:
clust = 5; % choose a cluster
cond = 1; % choose experimental condition
(>> help std_erspplot for function call)
[STUDY logersp timevals logfreqs pgroup pcond pinter] = ...
std_erspplot(STUDY,ALLEEG,'clusters',clust,'plotsubjects','on' );
% or plot them yourself from output:
figure; clim = 3; % standardize color limits
for ic = 1:size(logersp{cond},3) % all ICs in the cluster
    subplot(row,col,ic);
    imagesclogy(timevals, logfreqs, logersp{cond}(:, :, ic));
    set(gca,'clim', [-clim clim]); % adjust the color limits
    set(gca,'ydir','norm'); % plot low freqs at the bottom
    title(['IC ',int2str(STUDY.cluster(clust).comps(ic))]);
end;
textsc(['Cluster ',int2str(clust)], 'title');
cbar; % include a colorbar
```

Plot individual ERSPs



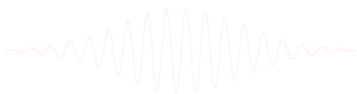
Raw data files



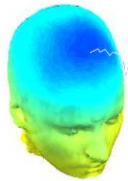
```
% Load *raw* ERSP data
```

```
load_string = 'C:\...\workshop\STUDY\S01\Memorize.icaersp';
```

```
ERSPdata = load('-mat',load_string); % .mat format!
```



Raw data structure



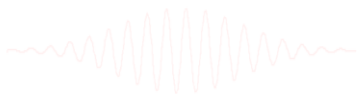
```
>> ERSPdata
```

```
ERSP dB data → comp1_ersp: [100 x 200 single]
dB baseline → comp1_erspbase: [1 x 100 single]
bootstrap limits → comp1_erspboot: [100 x 2 single]
                    comp2_ersp: [100 x 200 single]
                    comp2_erspbase: [1 x 100 single]
                    comp2_erspboot: [100 x 2 single]
100 frequency bins → freqs: [1 x 100 double]
200 time points → times: [1 x 200 double]
                    datatype: 'ERSP'
                    parameters: {1 x 26 cell}
                    datafile: [1 x 57 char]
```

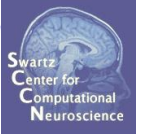
Annotations:

- 200 time points (points to the 200 in [100 x 200 single])
- upper and lower bootstrap limits (points to the 2 in [100 x 2 single])
- 100 frequency bins (points to the 100 in [100 x 200 single])
- bootstrap limits (points to the 2 in [100 x 2 single])

```
>> comp = 1;
>> oneic = ['ERSPdata.comp',int2str(comp),'_ersp'];
>> oneic = eval(oneic);
>> tms = find(RAWdata.times > 500 & RAWdata.times < 1000);
>> frs = find(RAWdata.freqs > 4 & RAWdata.freqs < 8);
>> mean(mean(oneic(frs,tms))) % mean raw power in window
```



Load other cluster raw data measures

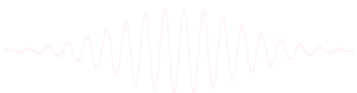


```
subjs = {'S01','S02','S03','S04','S05','S06','S07','S08','S09','S10','S11','S12','S13'};
conds = {'memorize','ignore','probe'};
des = 2;
subj = 1;

% raw ERSPs
load_string = [basedir,subjs{subj},'\Memorize.icaersp'];
load_string = [basedir,subjs{subj},'\design',int2str(des),'_',subjs{subj},conds{cond},'.icaersp'];

% OR, raw ITCs
load_string = [basedir,subjs{subj},'\Memorize.icaitc'];
load_string = [basedir,subjs{subj},'\design',int2str(des),'_',subjs{subj},conds{cond},'.icaitc'];

ERSPdata = load('-mat',load_string); % Run actual 'load' command
```



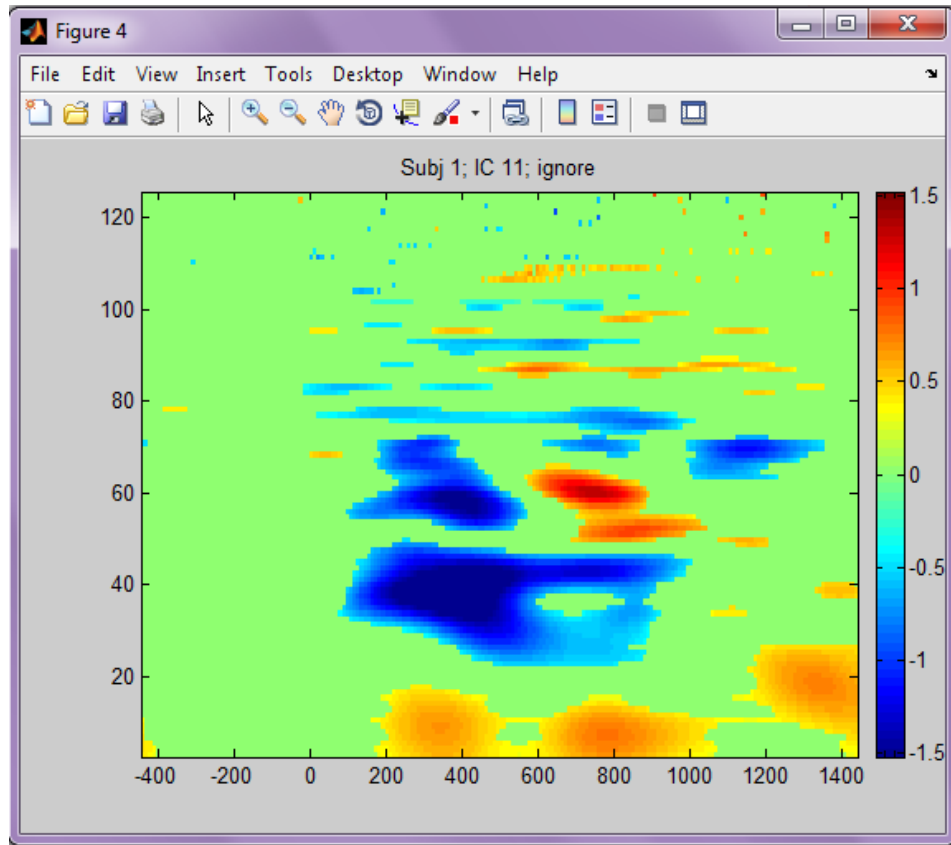
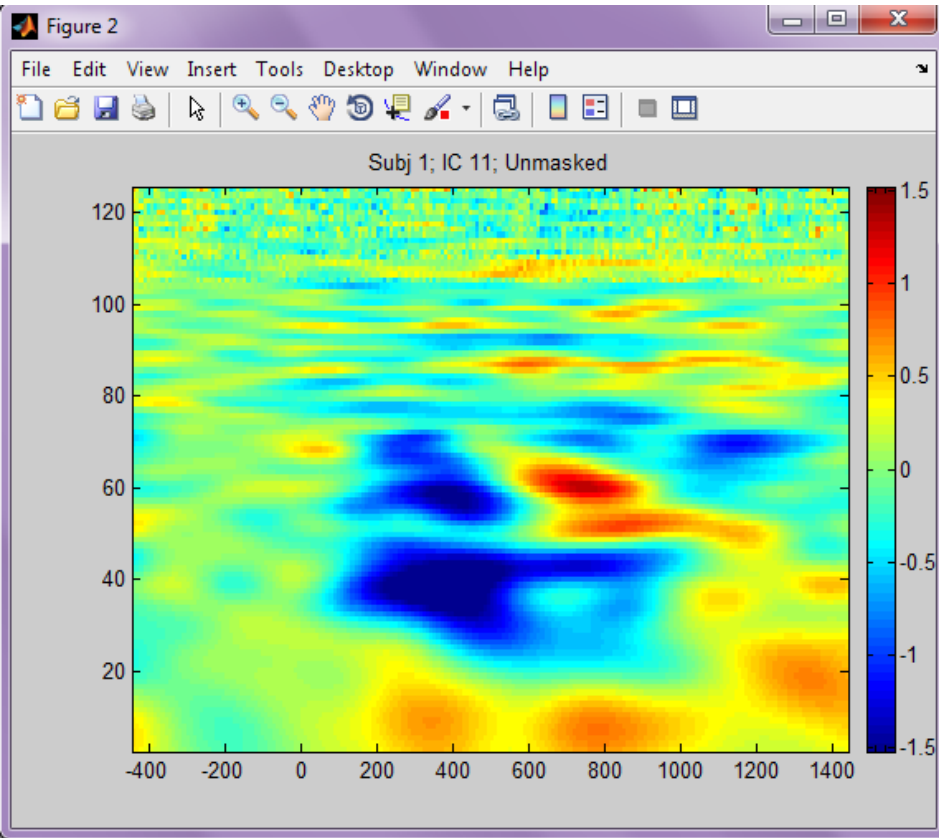
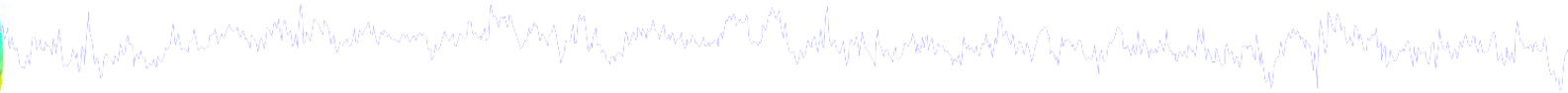
Load raw ERSP and subtract baseline



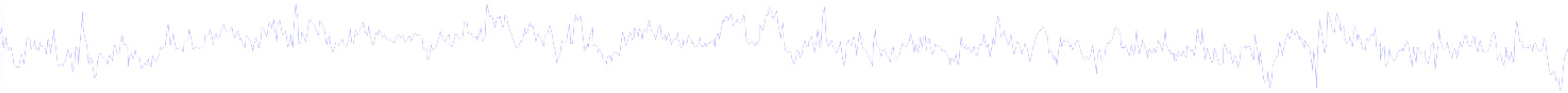
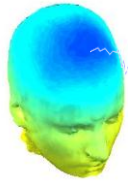
```
% Mask an ERSP using calculated bootstrap limits
ic = 7; % Choose and IC to plot (must be pre-calculated for this subject)
oneic = ['ERSPdata.comp',int2str(ic),'_ersp'];
oneic = eval(oneic);
onebase = ['ERSPdata.comp',int2str(ic),'_erspbase']; % to see the baseline spectrum
onebase = eval(onebase);
oneboot = ['RAWdata.comp',int2str(ic),'_erspboot'];
oneboot = eval(oneboot);

maskERSP = oneic;
maskERSP(find(oneic > repmat(oneboot(:,1),[1 size(oneic,2)])&...
    oneic < repmat(oneboot(:,2),[1 size(oneic,2)]))) = 0;
clim = 6; % set +/- color limits
figure; imagesc(ERSPdata.times,ERSPdata.freqs,maskERSP,[-clim clim]);
set(gca,'ydir','norm');
title(['Subj ',int2str(subj),'; IC ',int2str(ic)]);
cbar;
```

Mask raw ERSP with bootstrap values



Exercise

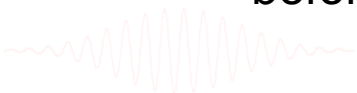


- **Novice**

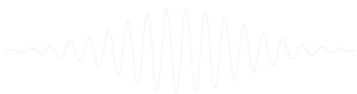
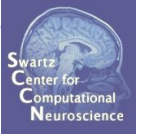
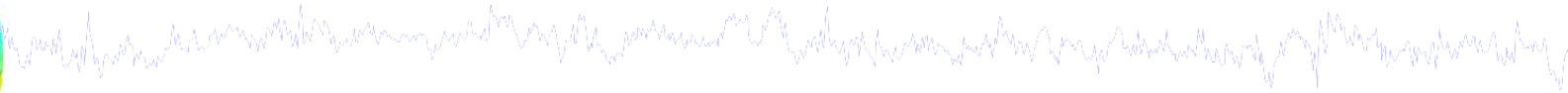
- Explore the STUDY structure: Identify subject and dataset number(s) for a single IC from one cluster
- Load ERSP or spectral data for a cluster and plot (`std_erspplot`)
 - > script a loop plotting all (or some) clusters

- **Intermediate / Advanced**

- Script a loop to build a STUDY from the command line
- Load *raw* ERSP data all ICs/subjs in a cluster and extract raw and/or baseline-corrected power in a time-frequency window
 - > (use `>> load('...icaersp',-mat)`)
- Plot an ERP image for a cluster. See what happens when you make `STUDY.cluster(clust).topopol` all positive (how many were -1 before?)



Supplementary lessons



STUDY.allinds/STUDY.setinds explanation



`STUDY.cluster(clust).setinds` and `STUDY.cluster(clust).allinds` are

cell arrays that are linked with the current `STUDY.design`. They are adjusted each time a new `STUDY.design` is selected.

`STUDY.cluster(clust).setinds` are indexes into the `STUDY.design` structure and `STUDY.cluster(clust).allinds` are the corresponding component indices.

In contrast, `STUDY.cluster(clust).sets` and `STUDY.cluster(clust).comps` fields correspond to each other but do NOT change when a new `STUDY.design` is selected. `STUDY.cluster(clust).sets` is a [cond x ncomps] matrix using all original STUDY conditions and gives the index of the corresponding dataset saved in `STUDY.datasetinfo`. Each column corresponds to the components listed in `STUDY.cluster(clust).comps`.

You will find that `STUDY.cluster(clust).allinds{1}`, for example, will have the same values as in `STUDY.cluster(clust).comps` (which are component indices included in the cluster). However, the *order* of the components may be different. That is because `STUDY.cluster(clust).allinds` refers to a different structure, the '`STUDY.design`', where the selected components might be different because of different subjects included in the current STUDY design. The actual `STUDY.datasetinfo` indices can be retrieved through `STUDY.setinds` and `STUDY.design`.



STUDY.allinds/STUDY.setinds explanation



Both `STUDY.cluster(clust).allinds` and `STUDY.cluster(clust).setinds` are cell arrays with the number of rows equal to the number of *primary independent variables* (i.e., conditions) in your selected STUDY design, and the number of columns equal to the number of *secondary independent variables*.

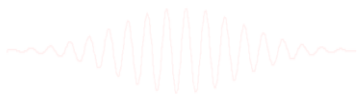
The matrix within one of these cells contains the number of columns equal to the number of *components* in the cluster.

STUDY.cluster(clust).setinds* are indices into the structure *STUDY.design.cell

Take an example from 'design 1' and 'cluster 6':

```
>> design_num = 1;
```

```
>> clust = 6;
```



STUDY.allinds/STUDY.setinds explanation

Now, if you have a cluster such as:

```
>> STUDY.cluster(clust)
ans =
    name: 'Cls 6'
    sets: [3x17 double]
    comps: [14 10 6 13 6 7 8 12 17 13 11 14 9 5 10 12 5]
    parent: {'Parentcluster 1'}
    child: []
    preclust: [1x1 struct]
    allinds: {3x1 cell}
    setinds: {3x1 cell}
    algorithm: {'Kmeans' [25]}
    topo: [67x67 double]
    topox: [67x1 double]
    topoy: [67x1 double]
    topoall: {1x17 cell}
    topopol: [1 1 1 1 1 1 1 1 1 -1 1 1 1 1 1 1 1]
```

Here we have a STUDY design with **3** conditions since `STUDY.cluster(clust).setinds` and `STUDY.cluster(clust).allinds` have **3 rows**.

Assume you are interested in finding the `STUDY.datasetinfo` index for the first component in the `STUDY.cluster(clust).allinds{1}` field:

STUDY.allinds/STUDY.setinds explanation



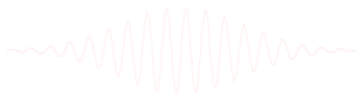
```
>> STUDY.cluster(clust).allinds{1}
```

```
ans =
```

```
[14 10 6 13 6 7 8 12 17 13 11 14 ... ]
```

In this case, we are interested in component **14**, but we do not yet know which subject or dataset this is referencing.

(Note that in this STUDY example, `STUDY.cluster(clust).allinds{1}` and `STUDY.cluster(clust).allinds{2}` will be identical because both conditions contain the same ICs).



STUDY.allinds/STUDY.setinds explanation



To find the dataset index, we have to look in `STUDY.cluster(clust).setinds{1}`:

```
>> STUDY.cluster(clust).setinds{1}
ans =
```

```
[1      2      3      4      5      6      7      7      8      9     10     10     11...]
```

Again, remember that these indices are NOT referring to `STUDY.datasetinfo`, but they will get you there eventually...

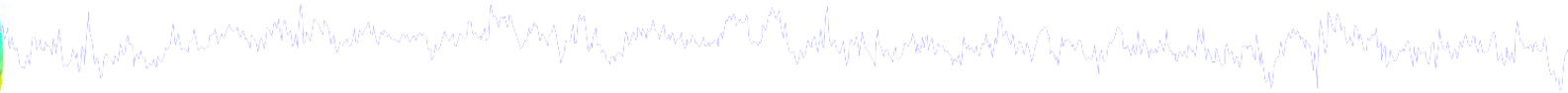
Take the first index, in this case **1**, and plug that into `STUDY.design(1).cell`:

```
>> STUDY.design(design_num).cell(1)
```

```
ans =
```

```
dataset: 2
 trials: {[1x267 double]}
  value: {'ignore' ''}
   case: 'S01'
filebase: [1x60 char]
```

STUDY.allinds/STUDY.setinds explanation

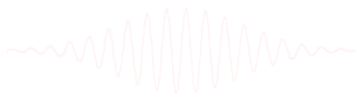


NOW, we can retrieve the actual `STUDY.datasetinfo` index, which in this case is dataset number **2**.

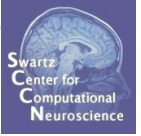
```
>> STUDY.datasetinfo(2)
```

```
ans =
```

```
    filepath: [1x53 char]
    filename: 'Ignore.set'
      subject: 'S01'
     session: 1
   condition: 'ignore'
        group: 1
        index: 2
        comps: [1x23 double]
   trialinfo: [1x267 struct]
```



Define variables

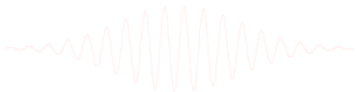


`% concatenate string variables:`

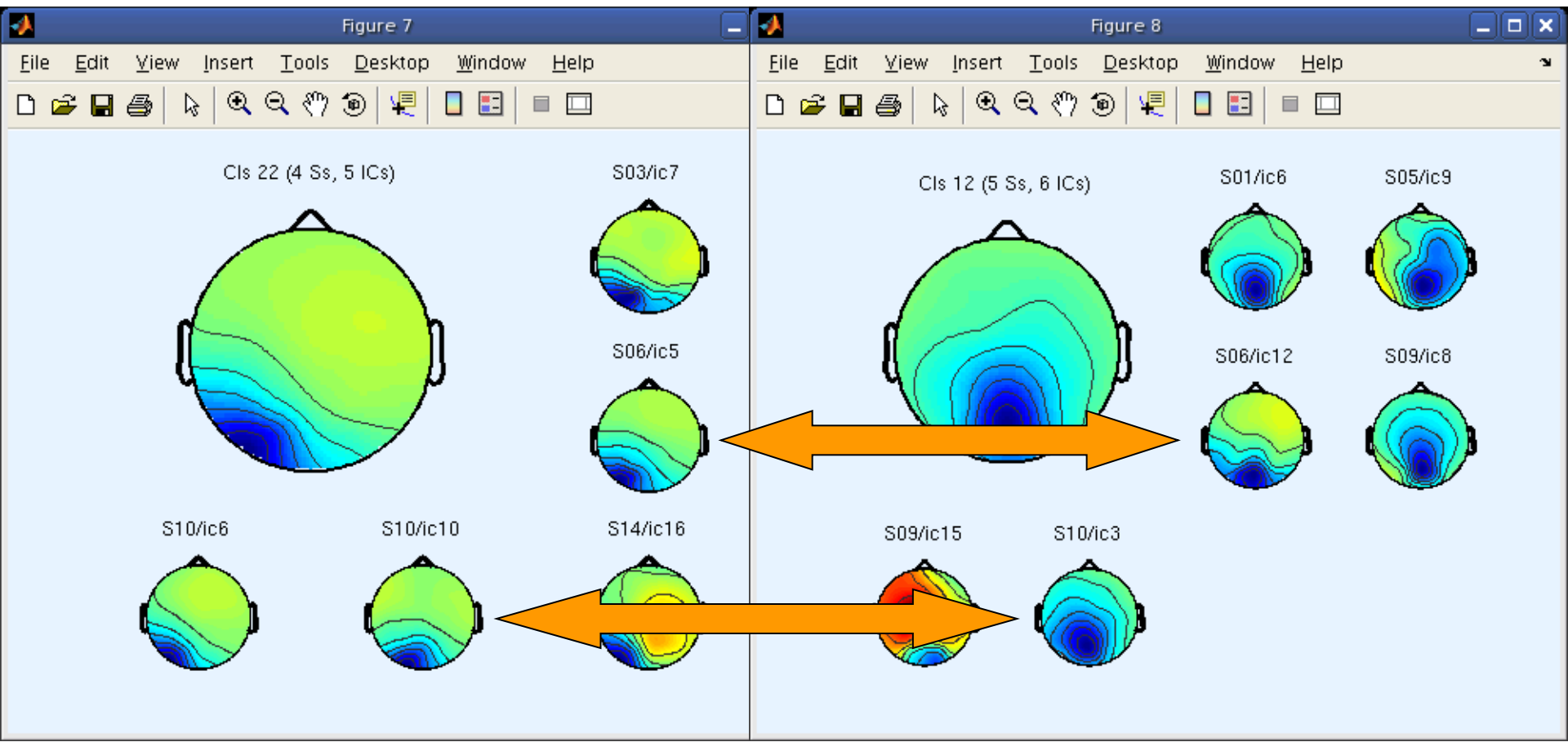
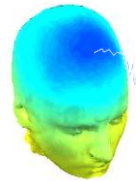
`[] % strings inside brackets will be concatenated`

```
dataset = [basedir,subjs{subj}, '\\',setnames{cond}];
```

```
C:\EEGLAB_Workshop\STUDY\S01\Memorize.set
```

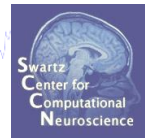
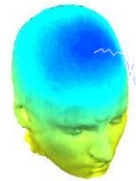


Cross coherence between clusters



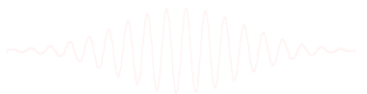
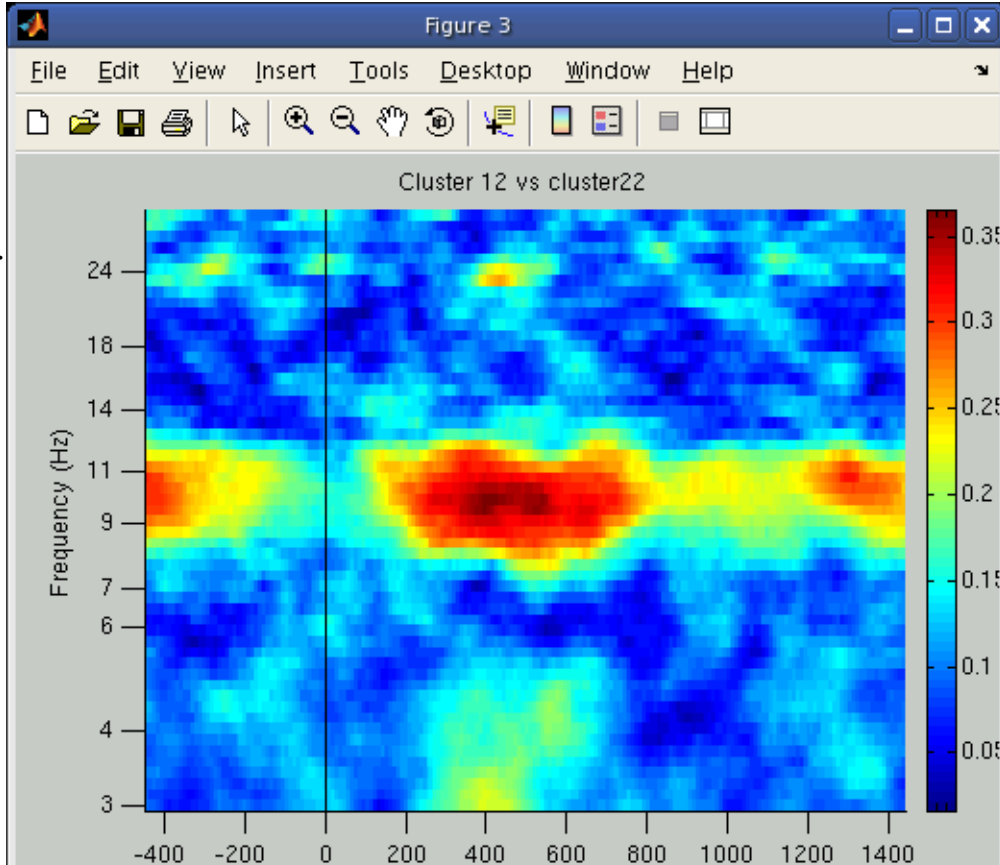
**cross coherence between clusters
requires 2 clusters with common subjects**

Cross coherence between clusters



Gives average phase coherence between members of two different IC clusters

```
figure;  
imagesclogy(times, freqs, mean(coh, 3));  
set(gca, 'ydir', 'norm'); hold on;  
plot([0 0], [get(gca, 'ylim')], 'k-');  
  
title(['Cluster ', ...  
int2str(clusts(1)), ' vs cluster ', ...  
int2str(clusts(2))]);  
  
% include a colorbar  
% for coh values:  
cbar;
```



STUDY cross coherence



```
clust1 = 6;
```

```
clust2 = 23;
```

```
% Crossf parameters:-----
```

```
type = 'phasecoher';
```

```
alpha = .01;
```

```
cycles = [3 0.5]; % wavelet cycles
```

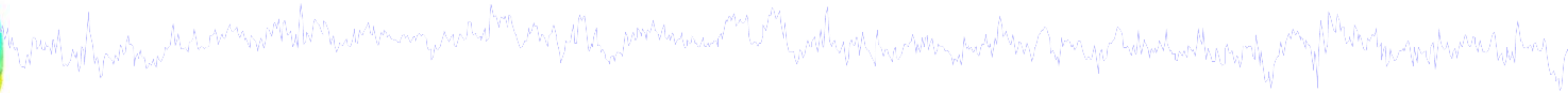
```
freqscale = 'log';
```

```
frqlim = [3 30]; % calculation frequency limits in Hz
```

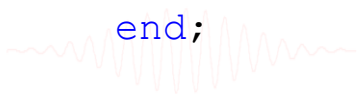
```
tmlims = [-100 1000]; % [min max] times in ms for window
```

```
cond = 1; % memorize only
```

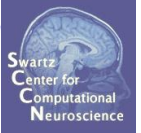
STUDY cross coherence



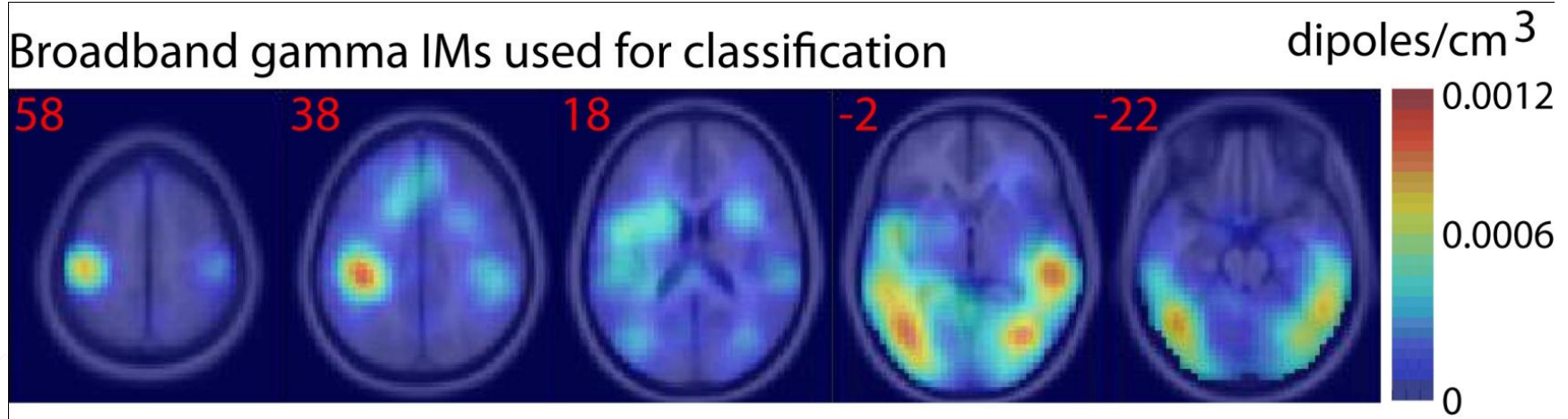
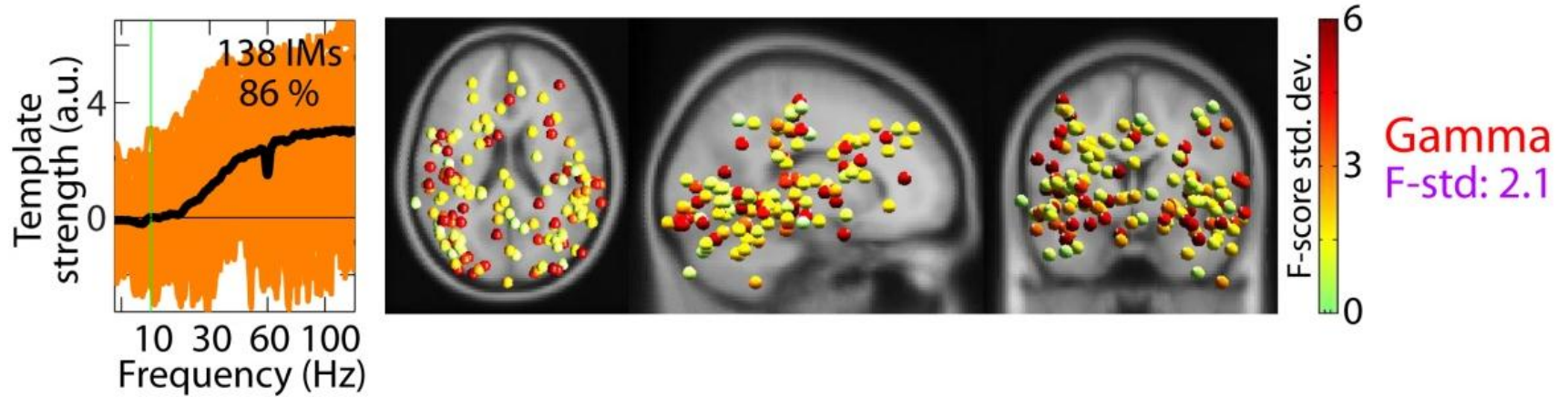
```
allrts = cell(1,0); p=1;
for ic = 1:length(STUDY.cluster(clust1).comps)
    setidx = STUDY.cluster(clust1).sets(cond,ic);
    comp1 = STUDY.cluster(clust1).comps(ic);
    [ALLEEG EEG CURRENTSET] = pop_newset(ALLEEG, EEG, CURRENTSET,...
    'retrieve',setidx, 'study',CURRENTSTUDY);
    subjmatch = find(STUDY.cluster(clust2).sets(cond,:) == setidx);
    for c = 1:length(subjmatch)
        comp2 = STUDY.cluster(clust2).comps(subjmatch(c));
        [coh(:,:,p),mcoh,times,freqs,cohboot(:,p),cohang(:,:,p)] = ...
            newcrossf(EEG.icaact(comp1,:) ,EEG.icaact(comp2,:),...
            EEG.pnts,[EEG.xmin*1000 EEG.xmax*1000], EEG.srate, cycles,...
            'alpha', alpha,'winsize',EEG.srate,'newfig','off',...
            'type',type,'freqs',frqlim,'freqscale' ,freqscale,...
            'savecoher',0 , 'plotamp' , 'off','plotphase' , 'off' );
        p = p+1;
    end;
end;
```



Dipole density plotting



PURPOSE: to visualize distributions of dipoles in 'MRI-esque' way



Dipole density plotting

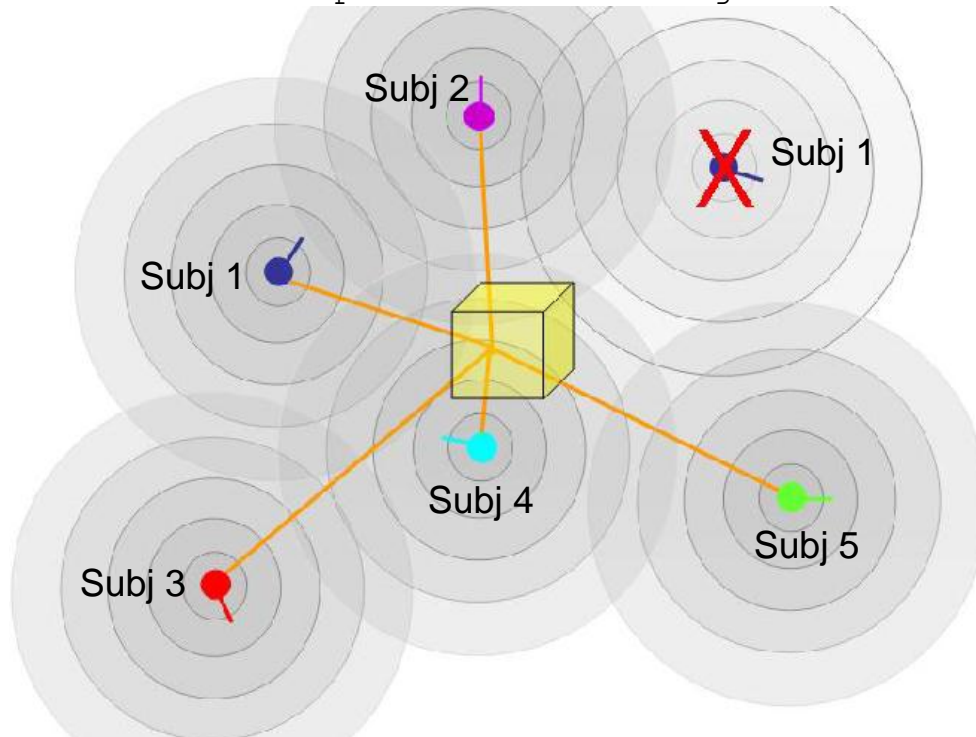


'method' - ['alldistance' | 'distance' | 'entropy' | 'relentropy'

'alldistance' - {default} takes into account the gaussian-weighted distances from each voxel to all the dipoles. See 'methodparam' (below) to specify a standard deviation (in mm) for the gaussian weight kernel.

'distance' - takes into account only the distances to the nearest dipole for each subject. See 'methodparam' (below).

**Explanation
of 'method'
argument
('distance')**



Dipole density plotting – commandline only



```
cond = 1;    clust = 3;

dipsources = struct('posxyz', [], 'momxyz', [], 'rv', []);    n = 1;
nowidx = 0; % initialize
for ic = 1:length(STUDY.cluster(clust).comps)
    setidx = STUDY.cluster(clust).sets(cond, ic);
    comp = STUDY.cluster(clust).comps(ic);
    if setidx ~= nowidx % don't call in if already active
        [ALLEEG EEG CURRENTSET] = pop_newset(ALLEEG, EEG, CURRENTSET, ...
            'retrieve', setidx, 'study', CURRENTSTUDY);    nowidx = setidx;
    end;
    model = EEG.dipfit.coordformat;
    dipsources(1, n).posxyz = EEG.dipfit.model(comp).posxyz;
    dipsources(1, n).momxyz = EEG.dipfit.model(comp).momxyz;
    dipsources(1, n).rv = EEG.dipfit.model(comp).rv;    n = n + 1;
end;
dipoledensity(dipsources, 'method', 'alldistance', 'methodparam', 10, ...
    'coordformat', model);
```