Cluster-level Randomization Tests

Motivation

In many studies, the answer to the research question involves a comparison between two experimental conditions. From introductory statistics courses we know that the T-statistic can be used for this type of comparison. More specifically, in a between-subjects design, we use the independent samples T-statistic, and in a within-subjects design we use the paired-samples T-statistic. Unfortunately, for EEG and MEG data (denoted as MEEG data in the following), the T-statistic alone will not answer our research questions. This is because the T-statistic can only be used for scalar observations, that is, observations that can be expressed as a single number only. In contrast, MEEG data are multidimensional (i.c., one-, two-, or three-dimensional; see further). As will be explained later in this tutorial, this dimensionality problem is also called the *multiple comparisons problem*. The Fieldtrip function `clusterrandanalysis.m` is a solution for this multiple comparisons problem. This function performs a statistical test of the difference between multidimensional observations in two or more conditions.

This tutorial starts with three sections that sketch the background of the randomization tests that are implemented in `clusterrandanalysis.m`. The next two sections are more tutorial-like. They deal with the analysis of an actual MEG dataset. In a step-by-step fashion, it will be shown how to statistically compare the data observed in two experimental conditions. If you want to skip the background information, then go directly to *Cluster-level Randomization Tests for Between-trial Experiments*.

Different Types of Data

To make things more concrete, we first have a look at the different types of data encountered in MEEG studies. Most familiar are the so-called s*patio-temporal* data. These data are two-dimensional: channels (the spatial dimension) X time-points (the temporal dimension). In some studies, these two-dimensional data are reduced to one-dimensional data by averaging over either a selected set of channels or a selected set of time-points. Somewhat less familiar are *spatio-spectral-temporal* data, including frequency as the third, spectral, dimension. Spatio-spectral-temporal data are preprocessed spatio-temporal data: The multiple time series in a spatio-temporal data matrix (one for every channel) are preprocessed by applying a (tapered) Fourier transform or a wavelet transform and retaining the power spectrum. In some studies, these three-dimensional data are reduced to one- or two-dimensional data by averaging over a selected set of channels, a selected set of time-points, and/or a selected set of frequencies. In the following, we assume three-dimensional data. This is the highest dimensionality that can be handled by `clusterrandanalysis.m`. If you prefer to think in terms of lower-dimensional data, just consider one or two of the three dimensions as a singleton dimension (i.e., a dimension that has only a single level).

To understand the statistical procedures one must also see how the structure in the experiment affects the structure in the data. Two concepts are crucial for describing the structure in an experiment:

(1) What is the unit of observation (also denoted as *unit*)?
(2) Is the experimental design between- or a within-units?

In neuroscience experiments, there are two types of units: subjects (humans or animals) and epochs of time within a subject. In MEEG studies, these epochs are usually called trials. We now turn to the second concept; whether the experimental design is between- or within-units. Units are observed in different experimental conditions. The research question involves whether these experimental conditions affect the dependent variable that is produced by or during these units. The units can be assigned to the experimental conditions according to two schemes: (1) every unit is assigned to one of a number of experimental conditions, or (2) every unit is assigned to multiple experimental conditions in a particular order. The first scheme is called a between-units, and the second a within-units experimental design. So, in a between-units design, there is one experimental condition per unit, and in a within-units design there are multiple experimental conditions per unit.

By combining the two units of observation and the two experimental designs, four types of experiments result. All four types of experiments are performed in neuroscience. However, between-trial and within-subjects experiments are much more common than the other two types of experiments, the between-subjects and the within-trial experiments. In this tutorial, we will only consider a between-trials, a within-trials, and a within-subjects experiment. The experiments can involve both spatio-temporal and spatio-spectral-temporal data. In this tutorial, we will give examples of both.

Computation of the Test Statistic

The test statistic computed by `clusterrandanalysis.m` is based on a cluster-finding algorithm that is implemented in `findcluster.m,` one of Fieldtrip's private functions. Before calling this cluster-finding algorithm, ordinary T-statistics are computed for all (channel, frequency, time)-triplets. These T-statistics are then thresholded at a user-specified value that is determined by `cfg.alphathresh`. For example, if `cfg.alphathresh` is equal to 0.05, the T-statistics are thresholded at the 95-th quantile for a one-sided test, and at the 2,5-th and the 97,5-th quantiles for a two-sided test. As will be explained in the next section (*The Null Distribution*), the value of `cfg.alphathresh` does not affect the type I error (false alarm) rate of the statistical test at the level of the complete data structure (all channels, frequencies, and time points); it only sets a threshold for considering a (channel, frequency, time)-triplet as a candidate-member of some large cluster of triplets (which may or may not be significant at the cluster-level).

The thresholded three-dimensional array of T-statistics is passed as an argument to the cluster-finding algorithm. This is done once for a one-sided test, and twice for a two-sided test (once for the positive T-statistics and once for the negative T-statistics). The cluster-finding algorithm finds all clusters of connected (channel, frequency, time)-triplets exceeding the threshold (`cfg.alphathresh`). Clusters can be connected via all three dimensions: the spatial, spectral and temporal dimension. For every cluster, the sum of the T-statistics is computed. This is our cluster-level statistic.

Almost always, there is more than one cluster. This confronts us with the problem of how to control the false alarm rate for all clusters jointly. This problem is often called the *multiple comparisons problem.* This false alarm rate for all clusters jointly is the probability when the null hypothesis holds of observing *at least one* cluster-level statistic that exceeds the critical value.

The Null Distribution

One way of solving the multiple comparisons problem is by taking the maximum of the different cluster-level test statistics (or minimum, in case of negative T-statistics) and computing the *null distribution* of this maximum. The null distribution is the distribution of the test statistic under the *null hypothesis*. The null hypothesis is a precise definition of what we mean by “no effect”. In a randomization test, the null hypothesis involves that the probability distributions of the MEEG data in the two conditions being compared are identical. Maris and Oostenveld (2005) spell out the definition of the null hypothesis in much more detail. The null distribution in a randomization test is also called the *randomization distribution.*

Given the randomization distribution of the maximum of the cluster-level test statistics, we can control the false alarm rate by computing p-values for the cluster-level statistics under this distribution of the maximum. This p-value is compared with a critical p-value, which is usually called the alpha-level, and specified in `cfg.alpha`. The alpha-level is the probability of *falsely* identifying at least one cluster-level statistic as being significant.

If the null hypothesis holds, then it is possible to actually compute the randomization distribution by randomly reassigning replications to conditions (in case of a between-units design) or by randomly permuting the order of paired observations (in case of a within-units design). The function `clusterrandanalysis.m` computes a Monte Carlo approximation of this null distribution. The computation of this Monte Carlo approximation involves that a user-specified number of randoms draws (specified in `cfg.nranddraws`) is taken from the computed null distribution, which is then used to construct an histogram that approximates the null distribution to any degree of accuracy that is required. The accuracy of the Monte Carlo approximation can be increased by choosing a larger value for `cfg.nranddraws`.

[If this is the first time you read this tutorial, skip this paragraph.] Computing the null distribution of the maximum of the different cluster-level test statistics is not the only way to solve the multiple comparisons problem. However, it is the default option in `clusterrandanalysis.m`, corresponding to the value `'maxsumt'` in the field `cfg.clusterteststat`. This solution to the multiplicity problem involves that every cluster-level test statistic is compared with the same critical value, the critical value under the null distribution of the maximum of the cluster-level test statistics. This critical value is determined by `cfg.alpha`, the so-called cluster-level false alarm rate. (This cluster-level false alarm rate should not be confused with `cfg.alphathresh`, the user-specifiable threshold for (channel, frequency, time)-triplets with large T-statistics.) Besides comparing every cluster-level test statistic with the same critical value it is also possible to compare the n-th largest cluster-level statistic with a critical value that is specific for this n-th largest cluster-level test statistic. These order-specific critical values are computed when `cfg.clusterteststat` has the value '`maxsumtminclustersize`'. With this option, order-specific critical values are computed for all clusters with a minimal cluster size that is user-specifiable (in `cfg.smallestcluster`). These critical values are determined such that, under the null hypothesis, the probability of observing a significant cluster-level test statistic *for any of the clusters of size larger than or equal to* `cfg.smallestcluster` is equal to `cfg.alpha`.

Cluster-level Randomization Tests for Between-trial Experiments

In this tutorial, we use the same data set that was also used in the tutorials on event-related averaging and time-frequency analysis. For the tutorial to be self-contained, we repeat a number of analysis steps that were described in these other tutorials. The data files you need for this tutorial can be found in the compressed folders ‘Subject01.zip’ and ‘ClusRandDistrib.zip’, which can be found under “Download” on the Fieldtrip homepage ‘http://www2.ru.nl/fcdonders/fieldtrip/’. In folder ‘ClusRandDistrib.zip’, you can also find the following Matlab scripts: `MEEGToolkitstats.m`, which contains the Matlab code used in this tutorial, and `maketrlSemCon.m`, which contains a so-called *trial function* (see further).

In a between-trial experiment, we analyze the data of a single subject. By means of a statistical test, we want to answer the question whether there is a systematic difference in the MEG recorded on trials with a fully congruent and trials with a fully incongruent sentence ending. We analyze the data of subject 01, whose data are in the folder ‘Subject01.ds’.

Preprocessing and Time-locked Analysis

We first extract the trials of the fully incongruent condition. The `preprocessing.m` function makes use of a so-called *trial function* to extract trials with a certain trigger number. For the data that is used in this tutorial, this trial function is `maketrlSemCon.m.` The trial function name is passed as an argument to `cfg.trialfun`. For this tutorial, you do not have to worry about the contents of `maketrlSemCon.m`, but for preprocessing your own data you might have to write your own trial function. In `cfg.trialfile`, you have to pass the name of the file that contains the trigger information. In our case, this information is in the same file as the one that contains the MEG signal ('Subject01.ds'). In Fieldtrip, a stimulus trigger is selected by specifying the *event type* and the *event value.* For the data that is used in this tutorial, the event type is specified in `cfg.trialdef.eventtype='backpanel trigger'`. (The label `'backpanel trigger'` is specific for the MEG acquisition system that is used at the FCDC, and should not worry you.) And the event value is specified in `cfg.trialdef.eventvalue=3`. The value 3 refers to fully incongruent sentence endings. All the other fields of the configuration (`cfg`) were described in the tutorial on event-related averaging.

```
cfg=[];
cfg.trialfile='Subject01.ds';
cfg.trialfun='maketrlSemCon';
cfg.trialdef.eventtype='backpanel trigger';
cfg.trialdef.eventvalue=3;
cfg.channel={'MEG'};
cfg.dataset='Subject01.ds';
cfg.blc='yes';
cfg.blcwindow=[-0.2 0];
cfg.lpfilter='yes';
cfg.lpfreq=35;
cfg.padding=0;
```

Store the output of `preprocessing.m` in `FICpreproc`:

```
[FICpreproc] = preprocessing(cfg);
```

We want to compare the fully congruent and fully incongruent condition. Therefore, we also have to extract the trials in the fully congruent condition. These trials were associated the trigger number 9. Thus, in our new configuration we must have `cfg.trialdef.eventvalue=9`. All other fields of `cfg` remain the same.

Store the output of `preprocessing.m` in `FCpreproc`:

```
cfg.trialdef.eventvalue=9;
[FCpreproc] = preprocessing(cfg);
```

Save the preprocessed data in the file `preprocdata`:

```
save preprocdata FICpreproc FCpreproc;
```

If you want to skip `preprocessing.m`, load the output from the file `preprocdataorig`:

```
load preprocdataorig;
```

The format of the preprocessed data is inconvenient for statistical analysis by means of `clusterrandanalysis.m`. A more convenient format is produced by `timelockanalysis.m` with the option `cfg.keeptrials='yes'`. The output of `timelockanalysis.m` then gives you an `.avg` field with the average event-related field, and a `.trial` field where the individual trial data can be found. The output is stored in `FICdata` and `FCdata` for, respectively, the fully incongruent and the fully congruent condition.

```
cfg = [];
cfg.keeptrials = 'yes';
FICdata = timelockanalysis(cfg, FICpreproc);
FCdata = timelockanalysis(cfg, FCpreproc);
```

Save the output of `timelockanalysis.m` in the file `timelockdata`:

```
save timelockdata FICdata FCdata;
```

If you want to skip `timelockanalysis.m`, load the output from the file `timelockdataorig`:

```
load timelockdataorig;
```

Cluster-level Randomization Tests for Event-related Averages

Cluster-level randomization tests are performed by the function `clusterrandanalysis.m`. This function takes as its arguments a configuration (`cfg`) and two or more data structures. These data structures are produced by `timelockanalysis.m` or some other function like `timelockgrandaverage.m` and `freqanalysis.m`. The output of these latter two functions will be used later, when we describe the statistical analysis of event-related fields in within-subjects experiments (for `timelockgrandaverage.m`) and of time-frequency representations in between-trial experiments (for `freqanalysis.m`).

The argument list of `clusterrandanalysis.m` must contain one data structure for every experimental condition. For comparing the data structures `FICdata` and `FCdata`, you must call `clusterrandanalysis.m` as follows:

```
[clusrand] = clusterrandanalysis(cfg, FICdata, FCdata);
```

The Configuration Settings

Some fields of the configuration (`cfg`), such as `channel` and `latency`, are not specific for `clusterrandanalysis.m`; their role is similar in other Fieldtrip functions. We first concentrate on the fields that are specific for `clusterrandanalysis.m`.

```
cfg.statistic   = 'indepsamplesT';  % independent samples T-statistic
cfg.alphathresh = 0.05;  % alpha level of the (channel, time)-specific test
        % statistic that will be used for  thresholding
cfg.makeclusters = 'yes';
cfg.minnbchan = 2; % minimum number of neighborhood channels that is required for a
        % selected (channel, time)-pair to be included in the clustering algorithm
        %(default=0).
cfg.clusterteststat = 'maxsum'; % test statistic that will be evaluated under the
        % randomization distribution.
cfg.onetwo = 'twosided'; % one-sided or two-sided test
cfg.alpha = 0.05; % alpha level of the randomization test
cfg.nranddraws = 100;  % number of draws from the randomization distribution
```

With `cfg.statistic='indepsamplesT'`, one chooses an independent samples T-statistic to compare the trials in the two experimental conditions (fully congruent and fully incongruent). This test-statistic will be calculated for every (channel, time)-pair and is later combined into a cluster-level test statistic (see further). In `cfg.statistic`, many other test statistics can be specified. Which test statistic is most appropriate depends on the data and the experimental design. For instance, if you want to compare three instead of two experimental conditions, you should choose an F-statistic (`'indepsamplesF'` or `'depsamplesF'`).

With `cfg.alphathresh`, one chooses the critical value that will be used for thresholding the (channel, time)-specific T-statistics. With `cfg.alphathresh=0.05,` every (channel, time)-specific T-statistic is compared with the critical value of the univariate T-test with an alpha-level (type I error rate) of 0.05. (This statistical test would have been the most appropriate test if we had observed a single channel at a single time-point.) The value of `cfg.alphathresh` does not affect the type I error rate of the *cluster-level* test. It only specifies how much evidence against the (channel, time)-specific null hypothesis is required in order for this pair to be considered a candidate-member of some large cluster of (channel, time)-pairs (which may or may not be significant at the cluster-level).

With `cfg.makeclusters='yes'`, one chooses to construct spatio-temporal clusters by combining spatially and temporally adjacent (channel, time)-pairs whose T-statistics exceed the threshold. With `cfg.makeclusters='no'`, no clusters are formed. In this case, every (channel, time)-pair can be considered as a single cluster of minimal size 1.

With `cfg.clusterteststat`, one chooses the test statistic that will be evaluated under the randomization distribution. This is the actual test statistic and it has to be distinguished from the (channel, time)-specific T-statistics that are used for thresholding. With `cfg.clusterteststat='maxsum'`, the actual test statistic is the maximum of the cluster-level statistics. A cluster-level statistic is equal to the sum of the (channel, time)-specific T-statistics that belong to this cluster. Taking the largest of these cluster-level statistics of the different clusters produces the actual test statistic.

The option `cfg.minnbchan` denotes the minimum number of neighborhood channels that is required for a selected (channel, time)-pair (i.e., a pair who's T-statistic exceeds the threshold) to be included in the clustering algorithm. With `cfg.minnbchan=0` (the default), it sometimes happens that two large clusters are connected via a narrow bridge of (channel, time)-pairs. Because they are connected, these two clusters are considered as a single cluster. Because of the limited spatial resolution of the MEEG-signal, such a narrow bridge is implausible from a biophysical perspective. Therefore, it makes sense to ignore all (channel, time)-pairs, selected on the basis of their T-statistics, if they have less than some minimum number of neighbors that were also selected on the basis of their T-statistics. This minimum number is assigned to `cfg.minnbchan`. This number must be chosen independently of the data.

With `cfg.onetwo`, one chooses between a one-sided and a two-sided statistical test. Choosing `cfg.onetwo='twosided'`, affects the output of `clusterrandanalysis.m` in the following way:

(1) The (channel, time)-specific T-statistics are thresholded from below as well as from above. This implies that both large negative and large positive T-statistics are selected for later clustering.
(2) Clustering is performed separately for thresholded positive and negative T-statistics.
(3) The critical value for the actual test statistic is two-sided: the lower critical value is equal to the minimum of the negative cluster-level statistics and the upper critical value is equal to the maximum of the positive cluster-level statistics.

With `cfg.alpha`, one controls the type I error rate of the randomization test (the probability of falsely rejecting the null hypothesis). The value of `cfg.alpha` determines the critical values with which we must compare the test statistic (i.c., the maximum and the minimum of the cluster-level statistics). The field `cfg.alpha` is not crucial. This is because the output of `clusterrandanalysis.m` (see further) contains a p-value for every cluster (calculated under the randomization distribution of the maximum/minimum cluster-level statistic). Instead of the critical values, we can also use these p-values to determine the significance of the clusters.

With `cfg.nranddraws`, one controls the number of draws from the randomization distribution. Remember that `clusterrandanalysis.m` approximates the randomization distribution by means of a histogram. This histogram is a so-called Monte Carlo approximation of the randomization distribution. This Monte Carlo approximation is used to calculate the p-values and the critical values that are shown in the output of `clusterrandanalysis.m.` In this tutorial, we use `cfg.nranddraws=100`. This number is too small for actual applications. As a rule of thumb, use `cfg.nranddraws=500`, and double this number if it turns out that the p-value differs from the critical alpha-level (0.05 or 0.01) by less than 0.02.

We now briefly discuss the `cfg`-fields that are not specific for `clusterrandanalysis.m`:

```
cfg.channel = {'MEG'}; % cell-array with selected channel labels (can be 'all')
cfg.latency = [0 1]; % time interval over which the experimental conditions must be
        %compared (in seconds)
```

With these two options, we have selected the spatio-temporal dataset involving all MEG sensors and the time interval between 0 and 1 second. The two experimental conditions will only be compared on this selection of the complete spatio-temporal dataset, which also contains the time interval beteen 1 and 2 seconds.

One should be aware of the fact that the sensitivity of `clusterrandanalysis.m` (i.e., the probability of detecting an effect) depends on the length of the time interval that is analyzed, as specified in `cfg.latency`. For instance, assume that the difference between the two experimental conditions extends over a short time interval only (e.g., between 0.3 and 0.4 sec.). If it is known in advance that this short time interval is the only interval where an effect is likely to occur, then one should limit the analysis to this time interval (i.c., choose `cfg.latency=[0.3 0.4]`.). Choosing a time interval on the basis of prior information about the time course of the effect will increase the sensitivity of the statistical test. If there is no prior information, then one must compare the experimental conditions over the complete time interval. This is accomplished by choosing `cfg.latency='all'`. In this tutorial, we choose `cfg.latency=[0 1]` because EEG-studies have shown that the strongest effect of semantic incongruence is observed in the first second after stimulus presentation.

Now, run `clusterrandanalysis.m` to compare `FCdata` and `FICdata` using the following configuration:

```
cfg.statistic   = 'indepsamplesT';
cfg.alphathresh = 0.05;
cfg.makeclusters = 'yes';
cfg.minnbchan = 2;
cfg.clusterteststat = 'maxsum';
cfg.onetwo = 'twosided';
cfg.alpha = 0.05;
cfg.nranddraws = 100;
cfg.channel = {'MEG'};
cfg.latency = [0 1];

[clusrand] = clusterrandanalysis(cfg, FCdata, FICdata)
```

Save the output of `clusterrandanalysis.m` in `clusrandFCvsFIC`:

```
save clusrandFCvsFIC clusrand;
```

If you want to skip `clusterrandanalysis.m`, load the output from the file `clusrandFCvsFICorig`:

```
load clusrandFCvsFICorig;
```

The Format of the Output

In the output of `clusterrandanalysis.m` there are separate output fields for positive and negative clusters. For the positive clusters, the output is given in the following pair of fields: `clusrand.posclusters` and `clusrand.posclusterslabelmat`. The field `clusrand.posclusters` is an array that provides the following information for every cluster:

1. In the field `sumstat`, the sum of the T-statistics in this cluster. In the above, this `sumstat` was called the *cluster-level statistic*.
2. In the field `pval`, the proportion of draws from the randomization distribution with a maximum cluster-level statistic that is larger than `sumtstat`.
3. In the field `size`, the number of elements in this cluster.

The elements in the array `clusrand.posclusters` are sorted according to their p-value: the cluster with the smallest p-value comes first, followed by the cluster with the second-smallest, etc. Thus, if the k-th cluster has a p-value that is larger than the critical alpha-level (e.g., 0.05), then so does the (k+1)-th. Type `clusrand.posclusters(k)` to see the information for the k-th cluster.

The field `clusrand.posclusterslabelmat` is a spatio-temporal matrix (i.e, one row per channel and one column per time point). This matrix contains numbers that identify the clusters to which the (channel, time)-pairs belong. For example, all (channel, time)-pairs that belong to the third cluster, are identified by the number 3. As will be shown in the following, this information can be used to visualize the topography of the clusters.

For the negative clusters, the output is given in the following pair of fields: `clusrand.negclusters` and `clusrand.negclusterslabelmat`. These fields contain the same type of information as `clusrand.posclusters` and `clusrand.posclusterslabelmat`, but now for the negative clusters.

By inspecting `clusrand.posclusters` and `clusrand.negclusters`, it can be seen that only the first positive and the first negative cluster have a p-value less than the critical alpha-level of 0.025. This critical alpha-level corresponds to a type I error rate of 0.05 in a two-sided test. By typing `clusrand.posclusters(1)` on the Matlab command line, you should obtain the following:

```
ans =

    sumstat: 1.2356e+004
    pval: 0
    size: 3505
```

And by typing `clusrand.negclusters(1)`, you should obtain the following:

```
ans =

    sumstat: -1.0250e+004
    pval: 0
    size: 3259
```

It is possible that the `pval`'s in your output are a little bit different from 0. This is because `clusterrandanalysis.m` uses Monte Carlo approximation to calculate these p-values. For instance, the p-value for a positive cluster is calculated as the proportion of random draws from the randomization distribution in which the maximum of the cluster-level statistics is more extreme than `sumstat` (the cluster-level statistic of this cluster).

Plotting the Results

To show the topography of the significant clusters, one can make use of the plotting function `topoplotER.m`. It will now be shown how `topoplotER.m` can be used to plot the so-called raw effect in the two significant clusters. The raw effect is the difference between the event-related averages for the two experimental conditions (FC and FIC). In the output of `clusterrandanalysis.m`, this raw effect can be found in the field `clusrand.raweffect`.

To select the raw effect in the two significant clusters, we make use of a so-called mask, which has the value 1 for all (channel, time)-pairs that belong to a significant cluster, and the value 0 otherwise. This mask is constructed as follows:

```
mask=(clusrand.posclusterslabelmat==1) | (clusrand.negclusterslabelmat==1);
```

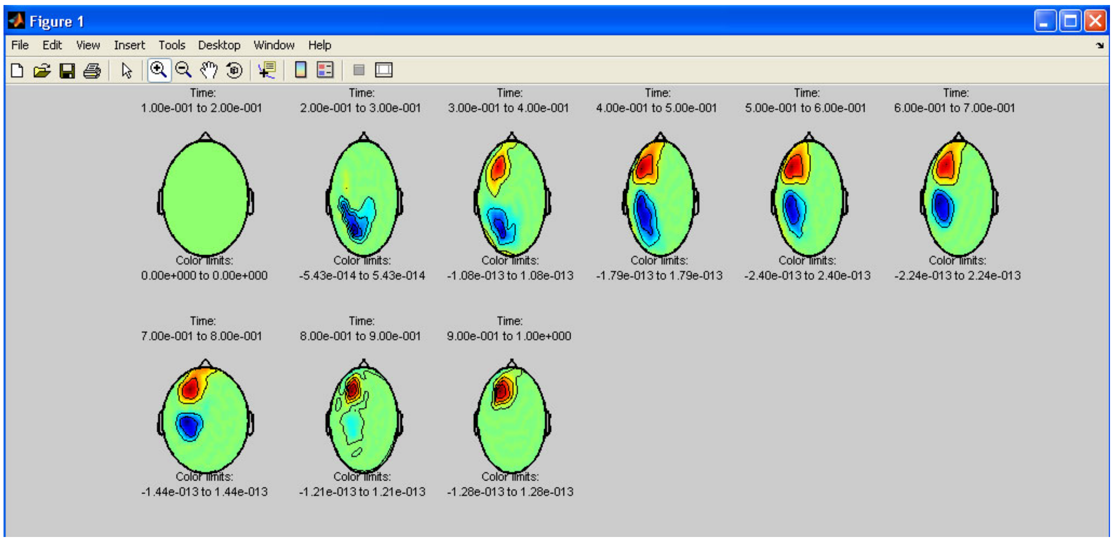
All the information topoplotER.m needs will be stored in `plotdata`:

```
plotdata.maskedraweffect=clusrand.raweffect.*mask;
plotdata.label=clusrand.label;
plotdata.time=clusrand.time;
```

In the assignment `plotdata.maskedraweffect=clusrand.raweffect.*mask`, a spatio-temporal matrix is constructed which contains the raw effect for all (channel, time)-pairs that belong to one of the two significant clusters, and the value 0 for all other (channel, time)-pairs.

To plot a sequence of ten topographic plots, equally spaced between 0 and 1 second, use the following configuration:

```
cfg=[];
cfg.yparam='maskedraweffect';
cfg.xlim=[0.1:0.1:1.0];
cfg.colorbar='no';
clf;
topoplotER(cfg,plotdata);
```



<u>Cluster-level Randomization Tests for Time-frequency Representations</u>

We will show how to statistically test the difference between the TFRs for the fully congruent (FC) and the fully incongruent (FIC) sentence endings. Before calculating the TFRs for the FC and the FIC sentence endings by means of `freqanalysis.m`, we have to rerun `preprocessing.m`. This is because the preprocessing configuration settings are usually different for event-related averaging and the calculation of TFRs. In particular, if the preprocessed data will be used for calculating TFRs, then it does not make sense to apply a high- or a low-pass filter, as we have done for event-related averaging. Therefore, we now rerun `preprocessing.m` with the following configuration settings:

```
cfg=[];
cfg.trialfile='Subject01.ds';
cfg.trialfun='maketrlSemCon';
cfg.trialdef.eventtype='backpanel trigger';
cfg.channel={'MEG'};
cfg.dataset='Subject01.ds';
cfg.blc='yes';
cfg.blcwindow=[-0.2 0];
```

For the FIC condition, store the output of `preprocessing.m` in `FICpreproc`:

```
cfg.trialdef.eventvalue=3;
[FICpreproc] = preprocessing(cfg);
```

And for the FC condition, store it in `FCpreproc`:

```
cfg.trialdef.eventvalue=9;
[FCpreproc] = preprocessing(cfg);
```

Save the preprocessed data in the file `preprocTFRdata`:

```
save preprocTFRdata FICpreproc FCpreproc;
```

If you want to skip `preprocessing.m`, load the output from the file `preprocTFRdataorig`:

```
load preprocTFRdataorig;
```

To calculate the TFRs, use `freqanalysis.m` with the following configuration:

```
cfg = [];
cfg.output        = 'pow';
cfg.sgn           = 'MEG';
cfg.method        = 'mtmconvol';
cfg.foi           = [5 10 20 40 80];
cfg.t_ftimwin     = 5./cfg.foi;
cfg.tapsmofrq     = 0.5*cfg.foi;
cfg.toi           = -0.5:0.05:1.4;
cfg.pad           = 'maxperlen';
cfg.keeptrials    = 'yes';
cfg.keeptapers    = 'no';
```

Calculate the TFRs for the two experimental conditions (FC and FIC):

```
TFRFC = freqanalysis(cfg, FCpreproc);
TFRFIC = freqanalysis(cfg, FICpreproc);
```

To save the TFRs in the file `TFR`:

```
save TFR TFRFC TFRFIC;
```

If you want to skip `freqanalysis.m`, load the output from the file `TFRorig`:

```
load TFRorig;
```

Now, run `clusterrandanalysis.m` to compare `TFRFC` and `TFRFIC.` Except for the field `cfg.latency`, the following configuration is identical to the configuration that was used for comparing event-related averages:

```
cfg=[];
cfg.statistic  = 'indepsamplesT';
cfg.alphathresh = 0.05;
cfg.makeclusters = 'yes';
cfg.minnbchan = 2;
cfg.clusterteststat = 'maxsum';
cfg.onetwo = 'twosided';
cfg.alpha = 0.05;
cfg.nranddraws = 100;
cfg.channel = {'MEG'};
cfg.latency = [0 1.4];

[clusrand] = clusterrandanalysis(cfg, TFRFC, TFRFIC);
```

Save the output of `clusterrandanalysis.m` in `clusrandTFRFCvsFIC`:

```
save clusrandTFRFCvsFIC clusrand;
```

If you want to skip `clusterrandanalysis.m`, load the output from the file `clusrandTFRFCvsFICorig`:

```
load clusrandTFRFCvsFICorig;
```

To show the topography of the first positive cluster, we again make use of `topoplotTFR.m`:

```
mask=(clusrand.posclusterslabelmat==1);
plotdata.powspctrm=clusrand.raweffect.*mask;
plotdata.label=clusrand.label;
plotdata.toi=clusrand.time;
plotdata.foi=clusrand.freq;
```

To plot a sequence of seven topographic plots for the lowest frequency (5 Hertz), equally spaced between 0 and 1.4 seconds, use the following configuration:

```
cfg=[];
cfg.xlim=[0:0.2:1.4];
cfg.ylim=[5 5];     % Frequency interval for which to plot
cfg.colorbar='no';
clf;
topoplotTFR(cfg,plotdata);
```

Try the other four frequencies.

Cluster-level Randomization Tests for Within-trial Experiments

We will show how to statistically test the difference between the time-frequency representations (TFRs) in the pre-stimulus (baseline) and the post-stimulus (activation) period of the fully incongruent (FIC) sentence endings. To perform this comparison by means of `clusterrandanalysis.m`, we have to select equal-length non-overlapping time intervals in the baseline and the activation period. For reasons that will become clear in the following, for the baseline period we choose [-0.5 -0.05], which is the time interval from 0.5 to 0.05 seconds before stimulus onset. And for the activation period we choose [0.95 1.4], which is the time interval from 0.95 to 1.4 seconds after stimulus onset.

We now calculate the TFRs for the baseline and the activation period. To calculate these TFRs, we need the preprocessed data `FICpreproc` as calculated for the statistical comparison between the TFRs for the FC and the FIC sentence endings.

If you want to skip `preprocessing.m`, load the output from the file `preprocTFRdataorig`:

```
load preprocTFRdataorig;
```

To calculate the TFRs, we use `freqanalysis.m` with the following configuration:

```
cfg = [];
cfg.output        = 'pow';
cfg.sgn           = 'MEG';
cfg.method        = 'mtmconvol';
cfg.foi           = [5 10 20 40 80];
cfg.t_ftimwin     = 5./cfg.foi;
cfg.tapsmofrq     = 0.5*cfg.foi;
cfg.pad           = 'maxperlen';
cfg.keeptrials    = 'yes';
cfg.keeptapers    = 'no';
```

These options are explained in the tutorial *Time-frequency Analysis Using Multitapers and Wavelets*. To calculate the TFR for the baseline condition:

```
cfg.toi          = -0.5:0.05:-0.05;
TFRFICbaseline = freqanalysis(cfg, FICpreproc);
```

And to calculate the TFR for the activation condition (using the same `cfg` as above, except for the field `toi`):

```
cfg.toi          = 0.95:0.05:1.4;
TFRFICactivation = freqanalysis(cfg, FICpreproc);
```

To understand why the time intervals [-0.5 -0.05] and [0.95 1.4] were chosen as, respectively, baseline and activation period, it is important to realize that power is calculated over time *intervals*. More specifically, for the power at 5 Hertz, `freqanalysis.m` uses the signal in a 1 second time window. This is specified in `cfg.t_ftimwin(1)`(the first element of the 5-element array `cfg.t_ftimwin`), which is equal to 1. To compare the baseline and the activation period, these time intervals must be chosen such that there is no overlap in the signal segments on which the power is calculated. This is achieved by our baseline and activation periods: the 5 Hertz power at 0.05 seconds *before* stimulus onset (the last time point in the baseline period) is calculated from the signal in the time interval [-0.55 0.45], and the 5 Hertz power at 0.95 seconds *after* stimulus onset (the first time point in the activation period) is calculated from the signal in the time interval [0.45 1.45], which is non-overlapping.

To save the TFRs in the file `TFRbasact`:

```
save TFRbasact TFRFICbaseline TFRFICactivation;
```

If you want to skip `freqanalysis.m`, load the output from the file `TFRbasactorig`:

```
load TFRbasactorig;
```

To compare `TFRFICbaseline` and `TFRFICactivation` by means of `clusterrandanalysis.m`, we use the following configuration:

```
cfg=[];
cfg.statistic  = 'actvsblT';
cfg.alphathresh = 0.05;
cfg.makeclusters = 'yes';
cfg.minnbchan = 2;
cfg.clusterteststat = 'maxsum';
cfg.onetwo = 'twosided';
cfg.alpha = 0.05;
cfg.nranddraws = 100;
cfg.channel = {'MEG'};
cfg.latency = 'all';
```

With `cfg.statistic='actvsblT'`, we choose the so-called activation-versus-baseline T-statistic. By means of this statistic, we compare the power in every (channel, frequency, time)-element in the activation period with the corresponding *average* power in the baseline period. This average is taken over time. The comparison of the activation and the average baseline power is performed by means of a pairwise T-statistic, with the pairs corresponding to the trials.

To run `clusterrandanalysis.m`:

```
[clusrand] = clusterrandanalysis(cfg, TFRFICactivation, TFRFICbaseline);
```

Save the output of `clusterrandanalysis.m` in `clusrandTFRFICactvsbas`:

```
save clusrandTFRFICactvsbas clusrand;
```

If you want to skip `clusterrandanalysis.m`, load the output from the file `clusrandTFRFICactvsbasorig`:

```
load clusrandTFRFICactvsbasorig;
```

By inspecting `clusrand.posclusters` and `clusrand.negclusters`, you will see that there is one significant positive cluster and no significant negative clusters.

To show the topography of the significant clusters, one can make use of the plotting function `topoplotTFR.m`. We will use this function to plot the raw effect in the significant positive cluster. This raw effect is the difference between the TFRs in the activation and the baseline period. In the output of `clusterrandanalysis.m`, this raw effect can be found in the field `clusrand.raweffect`.

To select the raw effect in the significant positive cluster, we make use of a mask:

```
mask=(clusrand.posclusterslabelmat==1);
```

All the information needed by `topoplotTFR.m` will be stored in `plotdata`:

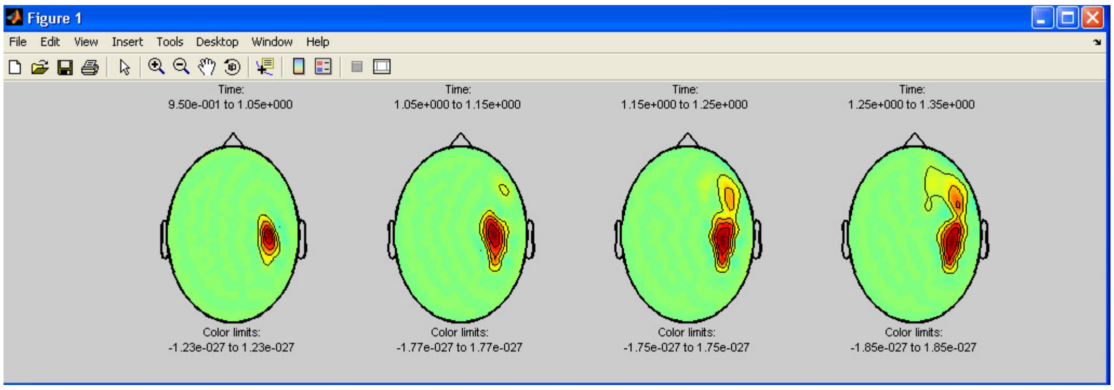
```
plotdata.powspctrm=clusrand.raweffect.*mask;
plotdata.label=clusrand.label;
plotdata.toi=clusrand.time;
plotdata.foi=clusrand.freq;
```

In the assignment `plotdata.powspctrm=clusrand.raweffect.*mask`, a spatio-spectral-temporal matrix is constructed which contains the raw effect for all (channel, frequency, time)-triplets that belong to the significant cluster, and the value 0 for all other (channel, frequency, time)-triplets. (The odd fieldname `powspctrm` is is due to the fact that `topoplotTFR.m` was originally written for plotting TFRs, and not masked difference-TFRs.) In this tutorial, use `topoplotTFR.m` to make topographic plots, separately for each of five frequencies.

To plot a sequence of four topographic plots for the lowest frequency (5 Hertz), equally spaced between 0.95 and 1.35 seconds, use the following configuration:

```
cfg=[];
cfg.xlim=[0.95:0.1:1.35];
cfg.ylim=[5 5];     % Frequency interval for which to plot
cfg.colorbar='no';
clf;
topoplotTFR(cfg,plotdata);
```

The resulting plot should show four scalp topographies with a uniform green colour. This is because the significant cluster does not contain 5 Hertz (channel, frequency, time)-triplets. Now, by replacing `[5 5]` in `cfg.ylim=[5 5]`, try the other four frequencies. With in `cfg.ylim=[20 20]` (the so-called beta-range), you should obtain the following plot.



Cluster-level Randomization Tests for Within-subjects Experiments

We now consider randomization testing for experiments that involve multiple subjects, each of which is observed in multiple experimental conditions. Typically, every subject is observed in a large number of trials, each one belonging to one experimental condition. Usually, for every subject, averages are computed over all trials belonging to each of the experimental conditions. Thus, for every subject, the data are summarized in an array of condition-specific averages. The randomization test that will be applied in the following, informs us about the following null hypothesis: The probability distribution of the array of condition-specific averages remains the same under permutation of these condition-specific averages. Maris and Oostenveld (2005) discuss this null hypothesis in detail.

We now describe how to use `clusterrandanalysis.m` to statistically test the difference between the event-related averages for the fully incongruent (FIC) and the fully congruent (FC) sentence endings. Because subjects might differ with the orientation of the dipole that is responsible for the effect (which would lead to cancellation of effects when axial gradiometers are used) we calculated synthetic planar gradients using `meginterpolate.m` (see the tutorial on event-related averaging).

To load the data structures containing the event-related averages:

```
load gravgerfcporig;
```

The event-related averages for the fully incongruent and the fully congruent sentence endings are stored in, respectively, `avg_erf_cp_FIC` and `gravg_erf_cp_FC.`

We now perform the randomization test using `clusterrandanalysis.m`. The configuration settings for this analysis differ from the previous settings in a single field only: `cfg.statistic='depsamplesT'` instead of `cfg.statistic='indepsamplesT'` or `cfg.statistic='actvsblT'`.

```
cfg=[];
cfg.statistic  = 'depsamplesT';
cfg.alphathresh = 0.05;
cfg.makeclusters = 'yes';
cfg.minnbchan = 2;
cfg.clusterteststat = 'maxsum';
cfg.onetwo = 'twosided';
cfg.alpha = 0.05;
cfg.nranddraws = 100;
cfg.channel = {'MEG'};
cfg.latency = [0 1];

[clusrand] = clusterrandanalysis(cfg, gravg_erf_cp_FIC, gravg_erf_cp_FC);
```

If you want to skip `clusterrandanalysis.m`, load the output from the file `clusranderfcpFICvsFCorig`:

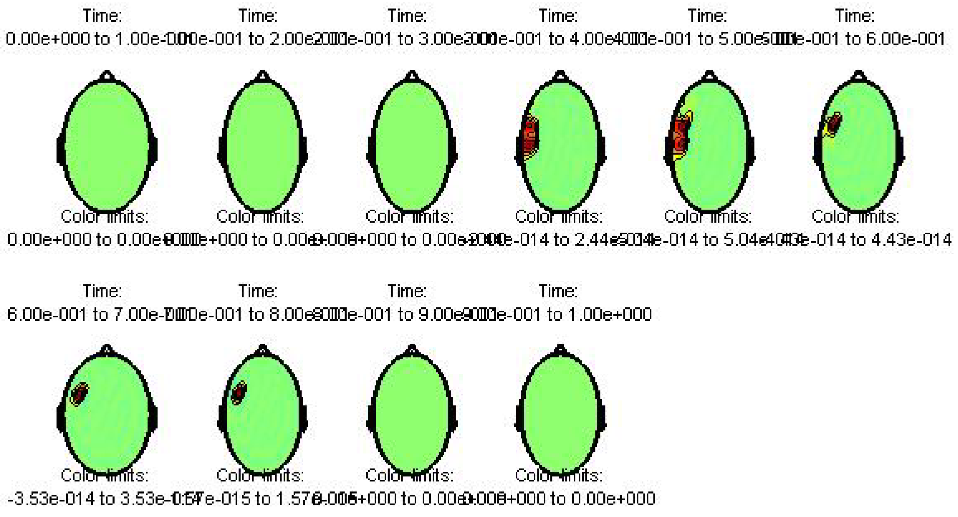
```
load clusranderfcpFICvsFCorig;
```

From inspection of `clusrand.posclusters` and `clusrand.negclusters`, we observe that there is only one significant positive cluster and no significant negative cluster.

To plot the results:

```
mask=clusrand.posclusterslabelmat==1;

plotdata.label=clusrand.label;
plotdata.time=clusrand.time;
plotdata.maskedraweffect=clusrand.raweffect.*mask;
cfg=[];
cfg.yparam='maskedraweffect';
cfg.colorbar='no';
cfg.xlim=[0:0.1:1];
clf;
topoplotER(cfg,plotdata);
```


Time:
0.00e+000 to 1.00e-001
Time:
1.00e-001 to 2.00e-001
Time:
2.00e-001 to 3.00e-001
Time:
3.00e-001 to 4.00e-001
Time:
4.00e-001 to 5.00e-001
Time:
5.00e-001 to 6.00e-001
Color limits:
0.00e+000 to 0.00e+000
Color limits:
0.00e+000 to 0.00e+000
Color limits:
0.00e+000 to 0.00e+000
Color limits:
-2.44e-014 to 2.44e-014
Color limits:
-5.04e-014 to 5.04e-014
Color limits:
-4.43e-014 to 4.43e-014
Time:
6.00e-001 to 7.00e-001
Time:
7.00e-001 to 8.00e-001
Time:
8.00e-001 to 9.00e-001
Time:
9.00e-001 to 1.00e+000
Color limits:
-3.53e-014 to 3.53e-014
Color limits:
-1.57e-015 to 1.57e-015
Color limits:
0.00e+000 to 0.00e+000
Color limits:
0.00e+000 to 0.00e+000