# 1. Introduction

Neuronal signals contain oscillations at many frequencies, and are implicated in many cognitive functions. It is commonly thought that oscillations reflect fluctuations of neuronal excitability, whose phase coupling could be used for the dynamic communication between neuronal populations. As such, neuronal oscillations are studied often, and phase coupling is a core component in many interpretations of experimental modulations. A large share of our community studies oscillatory activity on the basis of extracranial recordings, such as electroencephalography (EEG) and magnetoencephalography (MEG).

Investigating oscillatory activity in extracranial recordings (EEG/MEG) however, can be challenging for the following two reasons. First, interpreting sensor-level activity is strongly hindered by the fact that the underlying sources generate overlapping sensor-level spatial patterns, due to volume conduction/common pickup. Due to this, it may appear that sensor-level activity is e.g. either suppressed or enhanced by an experimental manipulation, whereas different underlying sources are each modulated differently. Secondly, investigating neuronal activity in extracranial brain signals is hampered by the fact that sensor-level phase coupling not only reflects communicating neuronal populations, but can also reflect activity that is best described by a current dipole, which is produced by a point source. Especially when the distance between sources and sensors is large, as for EEG/MEG, it is likely to observe sensor-level phase coupling that can be described by a current dipole.

This tutorial describes an approach for analyzing oscillatory neuronal activity that deals with the above problems in a particular way: *extracting and investigating rhythmic components*. Rhythmic components are extracted using a source separation, or decomposition, technique denoted as SPACE (for Spatially distributed PhAse Coupling Extraction). Analyzing oscillatory activity using rhythmic components has three key aspects. First, it allows for a separation of sources with overlapping spatial and spectral patterns, and therefore can reveal sources which are difficult to isolate in conventional pairwise analyses. Secondly, the strength of a component in each trial is quantified by a single number (denoted as a *trial loading*). These trial loadings allow for a straightforward way of investigating task modulations of oscillatory activity at the level of the extracted components. Thirdly, identifying rhythmic components is a first step in the analysis of phase-coupled oscillatory networks, because they provide a parsimonious description of the interacting neuronal populations that have produced the pattern of phase coupling at the sensor-level. *(Note that source separation techniques have a different purpose than source localization techniques, e.g. beamforming, which can be considered complementary.)*

## Overview

In this tutorial you can find information on how you can extract rhythmic components from extracranial recordings, what these components are, how they can be interpreted, and how we can use these to investigate task modulations of neural activity.

Rhythmic components will be extracted using the MATLAB toolbox *nwaydecomp* and *FieldTrip*, located at https://github.com/roemervandermeij/nwaydecomp and http://www.fieldtriptoolbox.org. The *nwaydecomp* toolbox contains several algorithms to find structure in neural recordings, of which this tutorial will use SPACE. One of the uses of SPACE is to extract rhythmic components. Other uses will be described in other tutorials. If you use SPACE as described in this tutorial, please cite the following two reference papers (in addition to the FieldTrip reference paper, mentioned at its wiki page above).

1: van der Meij R, Jacobs J, Maris E (2015). *Uncovering phase-coupled oscillatory networks in electrophysiological data.* Human Brain Mapping

2: van der Meij R, van Ede F, Maris E (2016). *Rhythmic Components in Extracranial Brain Signals Reveal Multifaceted Task Modulation of Overlapping Neuronal Activity.* PLOS One

Note that extracting rhythmic components can easily take 1-2 days per subject on a single core machine. Distributed computing can reduce this by a factor roughly equal to the number of random initializations (e.g. 8-10x, see below). See Box 3 below for how this can be achieved.

### An example dataset

In this tutorial we will be extracting components from an example MEG dataset, recorded at the Donders Institute, Radboud University Nijmegen, The Netherlands. The recorded subject performed a language semantics task which, involved passively listening to sentences, of which we'll use two conditions. In the first condition, the last part of each sentence was fully semantically congruent (FC) with the preceding part. For example, "*The climbers finally reached the top of the mountain*". In the second condition, sentence endings were fully semantically incongruent (FIC) with the preceding part, e.g. *"The climbers finally reached the top of the tulip"*. We will use rhythmic components to investigate differences in neural activity between these conditions. The dataset can be downloaded here: ftp://ftp.fieldtriptoolbox.org/pub/fieldtrip/tutorial/Subject01.zip. Additional information about the dataset and the conducted experiment can be found here: http://www.fieldtriptoolbox.org/tutorial/shared/dataset.

## 2. Background: what are rhythmic components?

### What are decompositions?

Before introducing rhythmic components, which are extracted using a decomposition technique, it is useful to illustrate what a decomposition is. Decomposition techniques in the context of this tutorial are also known as dimensionality reduction, source separation, or feature extraction techniques. In the broadest sense, decompositions describe 'structure' in the data in a more parsimonious way. Consider the following toy example (see Figure 1). In part of an EEG experiment we obtain measurements from multiple EEG electrodes over the course of a few seconds. The numbers representing these recordings are arranged in a 2-dimensional matrix (Fig 1A). Two distinct oscillations are present in this recording, a slow one and a fast one. Whereas the slow oscillation is strongest at the electrodes at the top, the fast oscillation is strongest at the middle electrodes. A decomposition technique uses the variability over the two dimensions (space and time), to separate these oscillations into what are called *components*. Each of the components describes one of the oscillations, by two 1-dimensional *loading vectors* (Fig 1B). The spatial loading vector quantifies how strongly each electrode reflects, or loads, the time-course (the spatial pattern), and the temporal loading vector quantifies how strongly each time-point reflects the spatial pattern. Importantly, because the components describe the spatial and temporal patterns of the oscillations separately, they are easier to interpret and analyze than the original matrix. The components are also a parsimonious description of the original matrix, because they describe the same patterns with fewer numbers.

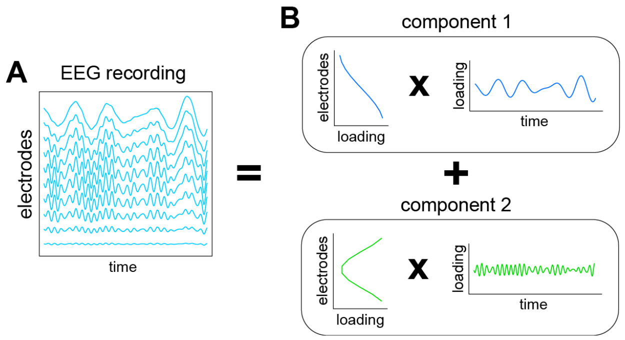

*Figure 1: illustration of a decomposition technique*

## What are rhythmic components?

Rhythmic components are extracted using a decomposition technique denoted as SPACE. They describe the structure of oscillatory neural activity in extracranial recordings, as measured over sensors, over time-windows, and present over frequencies. As such, rhythmic components describe the sensor-/electrode-level activity patterns produced by oscillatory neural sources over space, time and frequency. For the purpose of this tutorial, the time-windows will be complete trials of an experiment, but these time-windows can be any kind of temporal segmentation of a recording. Importantly, the structure is determined by *oscillatory phase coupling*, between sensors, over time, and over frequency. The sources a component can reflect can be single neural sources (e.g. Figure 2; likely a single source due to the 'dipolar' sensor-level pattern), so-called oscillating point sources, but also multiple interacting sources, so-called phase-coupled oscillatory networks (PCNs). These types of components, and how we can distinguish between them, is described in the last section of this tutorial. An example component extracted from an MEG recording is shown below.

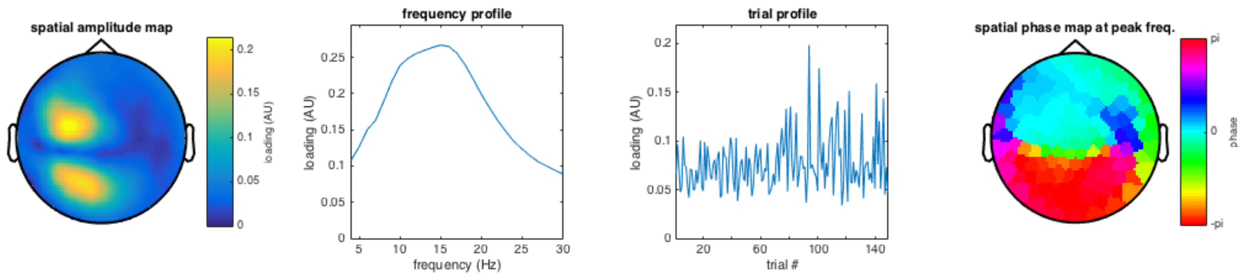

*Figure 2: example rhythmic component extracted from the example dataset*

## Rhythmic components consist of four parameters sets

A rhythmic component describes the activity pattern of an oscillatory neural source by four parameter sets (Figure 2). The *spatial amplitude map* (first panel) describes the degree to which the different sensors reflect the source that is described by the component. The *frequency profile* (second panel) describes which frequencies are involved in the component. The *trial profile* (third panel) describes the strength with which each component is present in each trial (or other type of time-window), which can be used to compare conditions. Finally, the *spatial phase maps* (last panel) describe, per frequency, the between-sensor phase relations that are induced by this source.

### Rhythmic components can be extracted without strong constraints

Rhythmic components are extracted using SPACE, which is inspired by a decomposition technique named PARAFAC/2[3-5]. Common decomposition techniques, like Principal and Independent Component Analysis (PCA and ICA) require constraints to extract components. For example, PCA can only extract components under the constraint that each (loading vector of a) component is orthogonal (uncorrelated) to all the others. In the case of ICA, components need to be statistically independent, which is an even stronger constraint. Rhythmic components can be extracted without constraints that strongly impact their interpretation (for the rationale behind this for PARAFAC, and SPACE by extension, see [6]). In more formal terms, SPACE does not require constraints such as the above to ensure uniqueness of the extracted components. This is described in more detail in the two SPACE reference papers.

### Two SPACE models

SPACE can extract components according to two models: SPACE-FSP (for Frequency-Specific Phase) and SPACE-time. These two 'sub-SPACEs' extract components that are very similar, but differ in how they describe between-sensor/electrode phase coupling.

SPACE-FSP is the model used in this tutorial to extract rhythmic components, and will be referred to simply as SPACE. In SPACE-FSP, each component has a spatial phase map for each frequency (fourth panel in Fig 2). These phase maps describe the consistent phase differences, over trials, between the sensors for each frequency at which these sensors are phase-coupled. As such, no particular relationship of phase differences *over frequencies* is enforced.

SPACE-time assumes a particular relationship of phase differences over frequency: those that are the result of time delays between sensors. Imagine two sensors whose activity reflects two groups of neurons, and imagine that one of these groups is transmitting information to the other with a time-delay (e.g., due to axonal conductance, synaptic integration, etc). If it is such that this communication is supported (or reflected) by an oscillatory signal in some frequency band, then the phase difference between the two groups will increase with frequency. This is because e.g., 10ms is 1/10th of a cycle at 10Hz, a ~1/9th of a cycle at 11Hz, ~1/8th of a cycle at 12Hz, and so on. SPACE-time uses the phase differences over frequencies to recover the time delay. Components extracted using SPACE-time are the focus of a different tutorial, but one which overlaps greatly with this one.

## 3. Background: two key issues when extracting rhythmic components

### Making sure rhythmic components are extracted accurately

To extract components, the iterative SPACE algorithm needs to be started randomly multiple times. Some of these random starts result in accurate components, some result in inaccurate components. Which random start should we choose? Because SPACE is an (alternating) Least Squares algorithm, we should *always* choose the random start with the highest amount of explained variance of the data. How do we know that there isn't another random starting point, which results in an even higher explained variance? We can never be certain, but by using the following rationale we can be reasonably certain. First, we sort random starts by their explained variance, and inspect whether explained variance has plateaued out. If so, we compare the components coming from random starts at this plateau. If they are (nearly) identical, we can be reasonably certain that we have found the 'most accurate' random start. All others need to be discarded.

Phrased differently, the SPACE algorithm can converge unto *local minima* of its loss function. To avoid this, we use multiple random starts and determine whether we have reached the *global minimum* using the rationale above.

In this tutorial, we will use software that takes care of the random initializing, and we will determine afterwards whether we have accurately extracted our components using several statistics. Importantly, the number of random initializations needs to be guessed. Because SPACE is computationally costly, and because this cost increases roughly linearly with the number of initializations, it is important to be conservative in the amount that we choose. In this tutorial, we begin with 10 random initializations, and then move forward from there. As a rule of thumb, the 'most accurate' random start is usually reached with 10-100 initializations.

## Determining the number of components to extract

The number of components to extract from the data cannot be determined analytically and needs to be determined empirically, similar to ICA and PARAFAC. This issue is completely separate from the random initializations described above, which holds for every step of what is discussed now. There are many different ways we can determine the number of components; some are currently implemented. In this tutorial, rather than obtaining an estimate of the true number of components that exist in the data, which might capture neural sources that are only active on a couple of trials, we will estimate the number of *reliable* components. Reliable in this context refers to being spatially and spectrally consistent over trials. In this tutorial we will do this by splitting the trials into two halves, extracting the same number of components from the full data and from both halves, and assess the similarity of the latter with the former (see below). If their similarity is sufficient, we increase the number of components to extract. If it is not, we lower the number of components. We do this until we have found the biggest number at which components are still reliable. The result of this procedure is that we will only extract components that are strongly present in both splits of trials, which may or may not be desired depending on the expected neural activity. In this tutorial we will use software that does the split-reliability determination automatically, and we will judge its success afterwards. Note that for each step, it is necessary to extract accurate rhythmic components, and, as such, each step requires multiple random starts (for each split, at each number of components to extract, etc).

Other implemented options for determining the number of components are to increase the number to extract until they no longer increase the explained variance by a certain number, or to simply extract a certain number of components, and judge their reliability afterwards. Each method has its advantages and disadvantages. Whatever method is used, it is important to distinguish between components that are driven by structure in the data, and those that are driven by noise.

(Note: components that are extracted depend on the other components (similar to ICA/PARAFAC, dissimilar to PCA without 'rotation'). That is, the first component when extracting two is not the same as the first component when extracting three. In practice though, they are often very similar.)

## 4. Extracting rhythmic components: from raw data to components

We will now extract components from the example MEG dataset described above. We will extract components using trials from two task conditions, the fully incongruent (FIC) condition, where sentences ended with a semantically mismatched word, and the fully congruent (FC) condition, where sentences ended with a semantically matching word.

### Procedure

The procedure to extract rhythmic components using the example dataset is described below. In the next section, we will visualize the components and compare component activity between conditions.

1) Preprocessing the raw data using *ft_definetrial* and *ft_preprocessing*
2) Calculating the input for SPACE, Fourier coefficients, using *ft_freqanalysis*
3) Determine the number of components to extract, and extract them, using *nd_nwaydecomposition*
4) Determine the appropriateness of the number of components extracted, and determine whether the amount of random initializations was sufficient
5) Visualize components

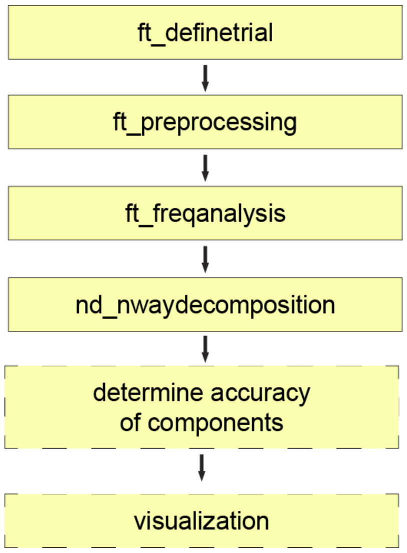

*Figure 3: schematic overview of the procedure. Broken lines indicate steps that are currently not implemented as a standalone function.*

## Preprocessing

Preprocessing and defining trials is done using *ft_definetrial* and *ft_preprocessing*. They require the original MEG dataset, which is available from
ftp://ftp.fieldtriptoolbox.org/pub/fieldtrip/tutorial/Subject01.zip.
We will define trials starting from the onset (t=0) of the critical word (incongruent or congruent) in the sentence up until 1.5s post-stimulus.

We first extract the trials of the fully incongruent condition:

```
% find the interesting segments of data
cfg = [];                                          % empty configuration
cfg.dataset                  = 'Subject01.ds';     % name of CTF dataset
cfg.trialdef.eventtype       = 'backpanel trigger';
cfg.trialdef.prestim         = 0;                  % seconds pre-stimulus
cfg.trialdef.poststim        = 1.5;                % seconds post-stimulus
cfg.trialdef.eventvalue      = 3;                  % event value of FIC
cfg = ft_definetrial(cfg);

% remove the trials that have artifacts from the trl
cfg.trl([15, 36, 39, 42, 43, 49, 50, 81, 82, 84],:) = [];

% preprocess the data
cfg.channel = {'MEG', '-MLP31', '-MLO12'}; % read all MEG channels except MLP31 and MLO12
cfg.demean  = 'yes';                       % do baseline correction with the complete trial
dataFIC = ft_preprocessing(cfg);
```

These data have been cleaned from artifacts by removing several trials and two sensors; see the FieldTrip visual artifact rejection tutorial for more information:
http://www.fieldtriptoolbox.org/tutorial/visual_artifact_rejection.

Then, we also extract the trials of the fully congruent condition:

```
% find the interesting segments of data
cfg = [];                                          % empty configuration
cfg.dataset                  = 'Subject01.ds';     % name of CTF dataset
cfg.trialdef.eventtype       = 'backpanel trigger';
cfg.trialdef.prestim         = 0;                  % seconds pre-stimulus
cfg.trialdef.poststim        = 1.5;                % seconds post-stimulus
cfg.trialdef.eventvalue      = 9;                  % event value of FC
cfg = ft_definetrial(cfg);

% remove the trials that have artifacts from the trl
cfg.trl([2, 3, 4, 30, 39, 40, 41, 45, 46, 47, 51, 53, 59, 77, 85],:) = [];

% preprocess the data
cfg.channel = {'MEG', '-MLP31', '-MLO12'}; % read all MEG channels except MLP31 and MLO12
cfg.demean  = 'yes';                       % do baseline correction with the complete trial
dataFC = ft_preprocessing(cfg);
```

As a final step, we combine both datasets using *ft_appenddata*:

```
% combine data
cfg = [];
datacomb = ft_appenddata([],dataFIC,dataFC);
```

In the new data structure *datacomb*, we can find which of the trials belonged to which condition by using the field *datacomb.trialinfo*. This field contains, in a 1xNtrials vector, the original trial codes used to define the trials (using cfg.eventvalue), which were 3 for the FIC condition and 9 for the FC condition. We will use this info later on in the tutorial.

## Spectral analysis

SPACE requires Fourier coefficients as input. Fourier coefficients describe the average phase and the average amplitude of oscillations, per frequency, in a particular time-window of the recordings, for each sensor/electrode. In this tutorial we will compute time-resolved Fourier coefficients, describing the average phase and amplitude of oscillations in a time-window that slides over the trial. We will use a time-window length that depends on frequency.

Below we compute Fourier coefficients using *ft_freqanalysis*, by doing wavelet convolution in the time-domain by multiplication in the frequency domain. There are other methods for spectral analysis, like filtering followed by the Hilbert transform, and short windowed Fourier Transforms. These approaches are mathematically equivalent given appropriate parameters [7]. The wavelets we will use are complex exponentials multiplied by a Hanning window/taper. The wavelets will be 5 cycles long, for frequencies from 4Hz to 30Hz. For those time-points where the wavelet cannot be fully immersed in the data, such as most time-points for 4Hz, the resulting Fourier coefficients will be NaNs.

```
% obtain Fourier coefficients
cfg = [];
cfg.foi         = 4:30;         % frequencies of interest in Hz
cfg.t_ftimwin   = 5./cfg.foi;   % sliding time-windows of 5 cycles wide
cfg.toi         = 0:0.010:1.5;  % time-window slides from 0 to 1.5s in steps of 10ms
cfg.method      = 'mtmconvol';  % wavelet convolution using mult. in freq. domain
cfg.taper       = 'hanning';    % windowing function for our wavelets
cfg.keeptrials  = 'yes';
cfg.output      = 'fourier';
cfg.pad         = 2;            % pad the data out to 2s, using spectral interpolation to
                                %    obtain Fourier coefficients at integer frequencies
freqdata = ft_freqanalysis(cfg,datacomb);
```

The above results in a *freqdata* data-structure. This structure contains amongst other fields, the *fourierspctrm*, which is a 4-way of Fourier coefficients.

For details on the requirements of the input when using Fourier coefficients computed outside of FieldTrip see *Box 1*. This additionally provides an important note on how SPACE uses multiple Fourier coefficients per trial, and on how to use other kinds of tapering.

For more information on the spectral analysis methods implemented in FieldTrip, see the time-frequency analysis tutorial: http://www.fieldtriptoolbox.org/tutorial/timefrequencyanalysis. The *freqdata* data-structure generated above is about 1.5GB big. If the size is an issue, it is safe to use the option *cfg.precision = 'single'*, which will reduce it by ~50%. SPACE will convert sections of the data to double precision where needed. Another way to reduce memory usage is described in *Box 2*.

*Box 1: Fourier coefficients for SPACE*

SPACE extracts components by using phase coupling structure in Fourier coefficients over sensors, frequencies, trials, and time-windows. In order to do this, the Fourier coefficients need to be arranged in a 4-way/dimensional array (NaNs where empty). The order of the dimensions of this 4-way array is given by the field *dimord*. It is a common field in FieldTrip input/output, and describes the dimensionality order of data-containing fields. When Fourier coefficients are computed using *ft_freqanalysis*, the dimord is *'rpttap_chan_freq_time'* (trials-by-sensor-by-frequency-by-time). When computing Fourier coefficients outside of FieldTrip, this field also needs to be present. If so, the *dimord* is required to be *'chan_freq_epoch_tap'*, and the Fourier coefficients 4-way array needs to follow this dimensionality. The first part refers to *channels*, being sensors in our case. The second part refers to *frequencies*. The third part refers to *epochs*, which are trials in our case. The epoching used can be trials, but this is not required. The fourth part refers to *tapers*, being time-windows in our case. This highlights an important aspect of SPACE. Each time-window per trial for which we computed Fourier coefficients, is actually thought of as a *Welch taper*. This is because SPACE, per frequency and trial, only uses the sensor-by-sensor cross products of the sensor-by-taper matrices (see Box 2). Each individual taper/time-windows serves to improve the accuracy of its cross product in a controlled manner (analogous to *Welch tapering*). In the *dimord* field the time-windows are referred to as tapers, because one can also use different kinds of tapering. This is useful for controlling the frequency resolution of the Fourier coefficients. A common one is *Slepian/DPSS* tapering, often simply called *multitapering*. This approach is discussed in detail in the FieldTrip time-frequency analysis tutorial: http://www.fieldtriptoolbox.org/tutorial/timefrequencyanalysis. When using *ft_freqanalysis*, the multiple tapers per time-window (when combining Slepian and Welch tapering) are handled automatically (see *Box 2* for reducing memory). When providing Fourier coefficients computed outside of FieldTrip, they need to be combined in the 4th dimension.

*Box 2: SPACE only uses the between-sensor cross products: an opportunity for memory usage reduction*

Though SPACE takes as input a 4-way array containing frequency- and trial-specific sensor-by-taper matrices, it only uses their sensor-by-sensor cross products (see also *Box 1*). SPACE inherits this from PARAFAC2. This leads to an important trick, which is carried out by the relevant functions, but is also of use for the end-user that computes and stores Fourier coefficients. This is because the sensor-by-taper matrices can become prohibitively large when using Fourier coefficients from a lot of tapers (time-windows). Because SPACE only uses the sensor-by-sensor cross products of these

matrices, we can replace the sensor-by-taper matrices (F) by a different matrix, as long as the cross product of this matrix is equal to the cross product of the original (FF*; where * is the complex conjugate transpose).
We can do this using the Eigendecomposition of the sensor-by-sensor matrix FF*. The Eigendecomposition of a symmetric matrix FF* can be written as follows:
$FF^* = VDV^*$
Where V is a matrix containing the Eigenvectors of FF* as columns and D is a diagonal matrix containing the corresponding (real-valued) Eigenvalues.
From this it follows that:
$FF^* = VDV^* = VD^{0.5}D^{0.5}V^* = (VD^{0.5})(D^{0.5}V)^*$
As such, the sensor-by-taper matrix F can be written as the product of the Eigenvectors of the Eigendecomposition of FF* and the square root of its eigenvalues.
$F = VD^{0.5}$
Because $VD^{0.5}$ will never bigger than sensors-by-sensors, memory usage can be reduced (as well as computation time).

## Extracting rhythmic components

Now that we have computed our Fourier coefficients, we can extract rhythmic components. We will do this using the function *nd_nwaydecomposition*. This function handles the multiple random initializations of the SPACE algorithm, and contains algorithms for determining the number of components to extract (see *Background*).

```
% extract rhythmic components
cfg = [];
cfg.model                = 'spacefsp';          % the model for extracting rhythmic components
cfg.datparam             = 'fourierspctrm';     % the field containing our Fourier coefficients
cfg.Dmode                = 'identity';          % necessary, see background/SPACE ref papers
cfg.ncompest             = 'splitrel';          % estimate number of components using splits
cfg.ncompeststart        = 5;                   % start from 5
cfg.ncompeststep         = 2;                   % increase number in steps of 2
cfg.ncompestend          = 50;                  % extract no more than 50
cfg.ncompestsrdatparam = 'oddeven';             % split trials by odd/even
cfg.ncompestsrcritval  = .7;                    % split-reliability criterion
cfg.numiter              = 1000;                % max number of iteration
cfg.convcrit             = 1e-6;                % stop criterion of algorithm: minimum relative
                                                %    difference in fit between iterations
cfg.randstart            = 10;                  % number of random initializations
cfg.ncompestrandstart  = 10;                    % number of random init. for split-rel. proc.
cfg.fsample              = datacomb.fsample;    % required for an internal correction
nwaycomp = nd_nwaydecomposition(cfg,freqdata);
```

We will use a split- reliability procedure for estimating the number of reliable components in the data. We do this by setting *cfg.ncompest = 'splitrel'*. For other approaches, see the function documentation. We determine the split-reliability by independently extracting components the full data and from two halves of the data, split along the trial dimension. One can also use other/more splits (see the function documentation). Reliability is then quantified using a coefficient akin to the Tuckers congruence coefficient [6]. This coefficient is computed between components extracted from the full data and components extracted from each split, per component, per parameter, and ranges from 0 to 1. A coefficient of 1 indicates that the parameter in question of two components is identical. We will interpret the computed coefficients afterwards. Using *cfg.ncompestsrcritval* we set how conservative we want to be in the procedure. It is the minimum coefficient at which components from the splits are considered similar as the components from the full data. We set it at 0.7, which can be loosely interpreted as requiring components to be 70% similar. Using the other *cfg.ncompest** options we can determine other aspects of the procedure.

Running the split-reliability procedure can take a very long time. In order to lower computation time, we are going to use a low number of random initializations, 10. This should take about 1-2 days on a single core of a CPU, depending on the hardware. Using a distributed computing setup, this can be reduced by a factor 5x-10x (see *Box 3*).

The next sections require the output generated by *nd_nwaydecomposition*. A copy of the output should have been distributed together with this tutorial for convenience.

*Box 3: using a distributed computing system to run random initializations in parallel*

To greatly speedup the extraction of rhythmic components it is possible to run the random initializations in parallel using a distributed computing system. Currently, two systems are supported, MATLABs Parallel Distributing Toolbox and Torque. The support for Torque depends on the FieldTrip *qsub* module (which needs to be on the MATLAB path). Other systems that are supported in *qsub* are implicitly supported as well.

Distributed computation of random initializations is specified with the following options:

```
cfg.distcomp.system          = string, distributed computing system to use, 'torque' or
                              'matlabpct' ('torque' requires the qsub FieldTrip module on
                               path, 'matlabpct' implementation is via parfor)
cfg.distcomp.timreq          = scalar, (torque only) maximum time requirement in seconds of
                               a random start (default = 60*60*24*1 (1 days))
cfg.distcomp.memreq          = scalar, (torque only) maximum memory requirement in bytes of
                               a random start (default is computed)
cfg.distcomp.inputsaveprefix = string, (torque only) path/filename prefix for temporarily
                               saving input data with a random name (default, saving is
                               determined by the queue system)
cfg.distcomp.matlabcmd       = string, (torque only) command to execute matlab (e.g.
                              '/usr/local/MATLAB/R2012b/bin/matlab') (default = 'matlab')
cfg.distcomp.torquequeue     = string, (torque only) name of Torque queue to submit to
                               (default = 'batch')
cfg.distcomp.mpctpoolsize    = scalar, (matlabpct only) number of workers to use (default is
                               determined by matlab)
cfg.distcomp.mpctcluster     = Cluster object, (matlabpct only) Cluster object specifying
                               PCT Cluster profile/parameters, see matlab help PARCLUSTER
```

**The output structure *nwaycomp***

The output MATLAB structure *nwaycomp* contains the following:

```
nwaycomp  =
        label: {149x1 cell}                % MEG sensor names
       dimord: 'A_B_C_L_D'                 % see below
         comp: {{1x5 cell}  {1x5 cell} ...} % each cell contains a component 1x5 cell-array
       expvar: 56.0151                     % explained variance
    tuckcongr: [28x1 double]               % N/A
      scaling: [7.6498e-11 6.7071e-11 ...]  % relative components ‘strengths’
   randomstat: [1x1 struct]                % details of random start procedure
 splitrelstat: [1x1 struct]                % details of split-reliability procedure
         grad: [1x1 struct]                % structure containing MEG sensor details
    trialinfo: [149x1 double]              % trial codes used in ft_preprocessing
         freq: [4:30]                      % frequencies used
         time: [1x151 double]              % time points used
          cfg: [1x1 struct]                % configuration options used
```

The function of some of these fields will become clear in the sections below. The two most important fields in the output are the *comp* field and the *dimord* field. The *comp* field is a cell-array, which in each cell contains a 1x5 cell-array containing the component-specific parameters. Which parameters are in which cell is described by the *dimord* (dimension order) field. This field is a crucial field in the output of most FieldTrip functions, and describes the content of each dimension in the most important output field (*comp*, in this case). A, B, C, L, and D describe the spatial amplitude map (A), the frequency profile (B), the trial profile (C), the spatial phase maps (L), and the between-component coherency matrices (D), respectively. The lettered labeling is the same as in the SPACE reference papers. Another field of interest is the *scaling* field. This contains a number, in the units of the input data, which can be used to judge the strength of a component relative to the other components.

The split-reliability procedure resulted in 8 components being extracted. The number of components, and the parameters of the components extracted, may differ between runs. If the difference is substantial, this is an indication additional random initializations are necessary.

## Did we use a sufficient number of random initializations to accurately extract components?

Before we interpret our components, we need to determine whether we should have used more random initializations. We do this by (1) judging whether the explained variance of random initializations has plateaued out, and (2) whether the random initializations at this plateau resulted in the same components (see *Background*). Whether the *number* of components is appropriate we will investigate in the next section.

We will now plot the information we need to make our decision in most cases. The necessary information is contained in the *nwaycomp.randomstat* field. This field is a MATLAB structure that contains details from the random start procedure. For those fields containing information per random initialization, the initializations are sorted by explained variance. We will use *randomstat.expvar*, containing the explained variance of each random initialization, *randomstat.congrall*, containing the congruence between initializations per component, per parameter, and *randomstat.congrglobmin*, containing the congruence between those initializations no more than 0.1% (absolute) away from the one with the highest explained variance. The *randomstat.globmininit* field describes which initializations these are. Congruence is computed using a coefficient similar to the split-reliability coefficient, going from 0 to 1 (perfectly similar). If the above does not provide sufficient detail for judging similarity between random initializations, *randomstat.congrcumul* contains the cumulative congruence between initializations of an increasing set of initializations (the first one, the first two, the first three, etc).

```
% plot details of the random initialization procedure
figure
% plot explained variance in first panel
subplot(1,3,1)
plot(1:10,nwaycomp.randomstat.expvar)
axis([1 10 0 nwaycomp.randomstat.expvar(1)+5])
xlabel('random start #'); ylabel('%')
title('explained variance of each initialization')
% plot congruence for those at the global optimum
subplot(1,3,2)
imagesc(nwaycomp.randomstat.congrglobmin)
caxis([.7 1]); axis xy; axis square; ylabel('component number'); colorbar
set(gca,'xtick',1:5,'xticklabel',{'A','B','C','L','D'})
title('congruence between initializations at global optimum')
% plot congruence for all initialzations
subplot(1,3,3)
imagesc(nwaycomp.randomstat.congrglobmin)
caxis([0 1]); axis xy; axis square; ylabel('component number'); colorbar
set(gca,'xtick',1:5,'xticklabel',{'A','B','C','L','D'})
title('congruence between all initializations')
```

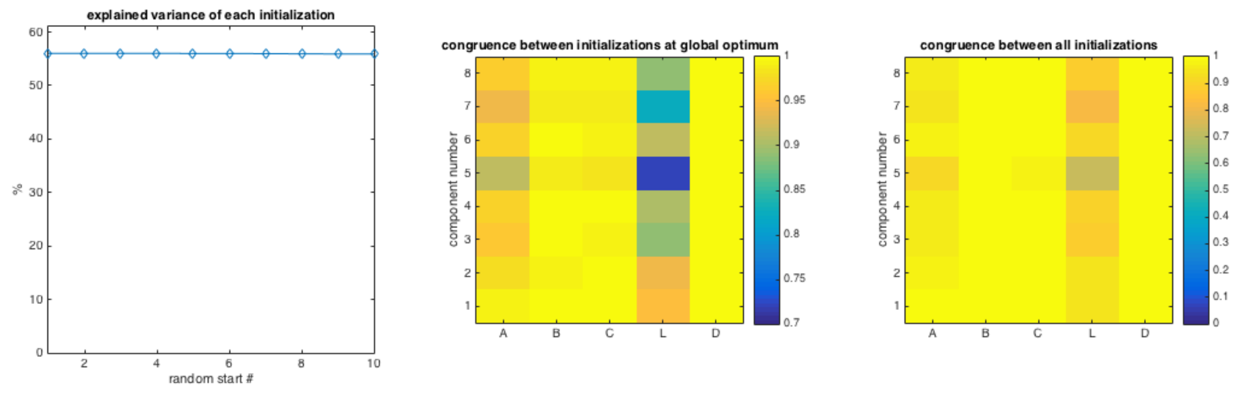

*Figure 4: details of the random initialization procedure. Note the different color scales on the two rightmost panels.*

The first panel of the above figure shows the explained variance of each initialization. At this zoom-level, explained variance appears to have plateaued out, and the 'most accurate' random start appears to have been obtained, and covers all 10 initializations. The second and third panel shows the Tuckers congruence coefficients (0 <-> 1) for each parameter, for each component. The second panel for only those initializations at the plateau, the third panel for all initializations. In this case they both contain all, but note the different color scale, whose purpose is the highlight differences at the global optimum (whose initializations are typically very similar).

For nearly all components and all parameters the coefficients are around .9+, and only one is around .8. As such, we can all say components were very similar over initializations. Combined with the explained variance plateauing out, we can be confident we have reached the 'most accurate' random start. If we want additional certainty, we can increase the number of random initializations to say, 20, and rerun the procedure.

## Did the component number estimation procedure succeed?

The next step is to determine whether we extracted an appropriate number of components, N. We will use the field *nwaycomp.splitrelstat*, containing details of the split-reliability procedure, similar to *nwaycomp.randomstat*. Of its fields, we first look at *splitrelstat.stopreason*, which indicates why the procedure stopped. If this says *'split-reliability criterion'*, which it does in this case, we can continue. This field can also mention whether the procedure failed by reaching the maximum N we set in *cfg.ncompestend*, or whether none of the components reached the split-reliability criterion ($N_{reliable}$<1).

Now that we know the procedure stopped because no additional reliable components could be extracted, we look into *splitrelstat.splitrelsucc* and *splitrelstat.splitrelfail*. These contain the split-reliability coefficients for the N at which all components were reliable, and for the next N, at which at least one parameter set of one component was unreliable.

```
% plot the split-reliability coefficients for N = success and N = fail
figure
% first plot for the N that succeeded
for isplit = 1:2
  subplot(2,2,isplit)
  imagesc(nwaycomp.splitrelstat.splitrelsucc(:,:,isplit))
  caxis([0 1]); axis xy; axis square; ylabel('component number'); colorbar
  set(gca,'xtick',1:5,'xticklabel',{'A','B','C','L','D'})
  title(['split-reliability coeff. split ' num2str(isplit) ' at N = success'])
end
% then plot for the N that failed
for isplit = 1:2
  subplot(2,2,isplit+2)
  imagesc(nwaycomp.splitrelstat.splitrelfail(:,:,isplit))
  caxis([0 1]); axis xy; axis square; ylabel('component number'); colorbar
  set(gca,'xtick',1:5,'xticklabel',{'A','B','C','L','D'})
  title(['split-reliability coeff. split ' num2str(isplit) ' at N = fail'])
end
```

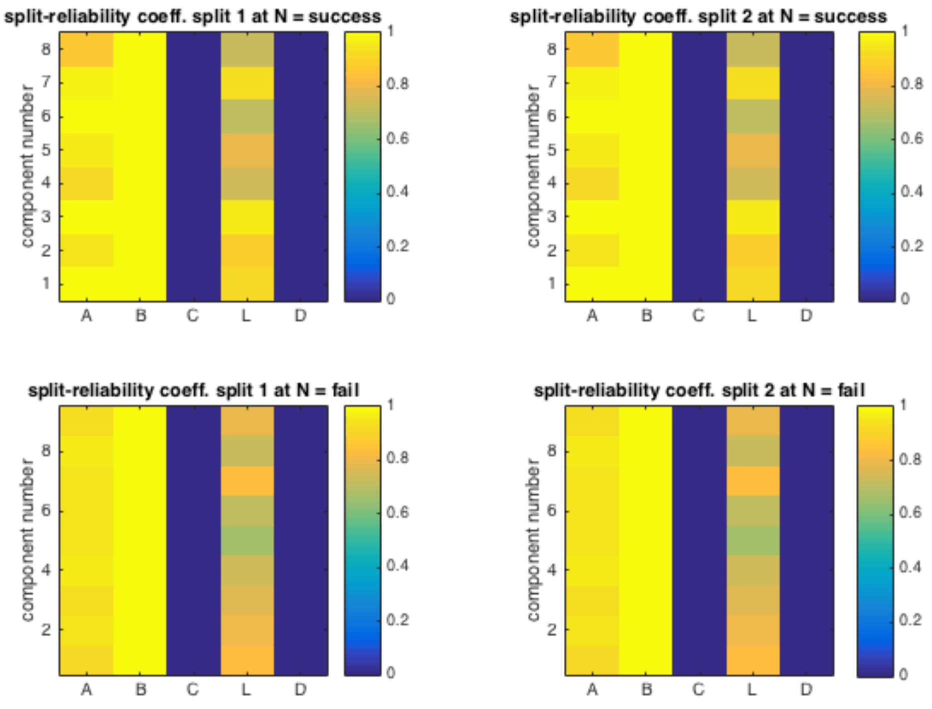

*Figure 5: details of the split-reliability procedure*

In this figure the coefficients for the trial profiles (C) and between-component coherency (D) are set to NaN. The trial profiles are not relevant for the split-reliability procedure, as it is the dimension the data is split over. The between-component coherency is also irrelevant (see above and the SPACE reference papers).

In the first row, visualizing the split-reliability coefficients at the highest successful N, we can observe that all split-reliability coefficients are above 0.7 for both splits (as expected given our settings). As such, the components extracted from the full data match those extracted from independent halves. We can further observe that most coefficients for the spatial amplitude maps (A) and frequency profiles (B) are substantially higher, indicating a high stability of these parameters over trials. The second row shows the coefficients at the N that failed. When we compare the first row to the second row, we see very little difference. The lowest coefficients in the second row belong to the spatial phase maps (L) of one or two components of both splits. These are the coefficients that caused the N to fail. The failure seems very minimal; coefficient(s) not lower than 0.6. Because all of the other coefficients are similar to the successful N, we could conclude that we were too conservative, and that we stopped the procedure prematurely. It is possible we could have extracted another 'reasonably reliable' component if we relaxed our criterion a little bit, or if we reran the procedure (i.e. the low coefficients might be the result of unlucky random starts). For now however, we will be conservative, and continue with the 8 components we extracted.

The field *splitrelstat* contains additional fields in case we wish to be more rigorous. For example, in the field *splitrelstat.randomstatsucc/fail*, we can find the *randomstat* fields for both splits of the data for the N at which the procedure failed/succeeded. Another example is the *splitrelstat.allcompsrc* field, which contains the split-reliability coefficients for all N's at which it was computed.

## 5. Visualizing and interpreting rhythmic components

Now we can finally turn to investigating the rhythmic components we extracted from our data. First, we'll investigate what frequencies of oscillatory activity each component reflects, and on which sensors it was present. This informs us about what oscillatory neural activity we extracted and described with rhythmic components. Then, we'll use the trial profiles of each component to investigate whether this neural activity was modulated by the task. Finally, we'll briefly touch on the type of neural source each component reflects, using the spatial phase maps.

### Step 1: Inspecting the spatio-spectral structure of the extracted components

Using the code below, we can visualize the spatial amplitude maps and the frequency profiles.

```
% plot components in panel columns
figure
ncomp   = numel(nwaycomp.comp);
for icomp = 1:ncomp
  % plot spatial amplitude map (contained in the 1st cell)
  % set spatial plotting cfg
  cfg = [];
  cfg.layout    = 'CTF151';
  cfg.parameter = 'spatmap';
  cfg.comment   = 'no';
  cfg.marker    = 'off';
  cfg.zlim      = 'zeromax'; % the color scale will go from 0 to the max of the spat. amp. map
  % plot using ft_topoplotER
  subplot(2,ncomp,icomp + (ncomp*0))
  plotdat = [];
  plotdat.spatmap = nwaycomp.comp{icomp}{1};
  plotdat.label   = nwaycomp.label;
  plotdat.dimord  = 'chan';
  ft_topoplotER(cfg,plotdat);
  title({['comp. ' num2str(icomp)];...
        ['strength: ' num2str(nwaycomp.scaling(icomp)./nwaycomp.scaling(1),'%1.2f')];...
        '';...
        'spatial amp. map'})

  % plot frequency profile (contained in the 2nd cell)
  subplot(2,ncomp,icomp + (ncomp*1))
  plot(nwaycomp.freq,nwaycomp.comp{icomp}{2})
  axis([nwaycomp.freq([1 end]) 0 max(nwaycomp.comp{icomp}{2})*1.1])
  ylabel('loading (AU)'); xlabel('frequency (Hz)')
  title('freq. profile')
end
```

The components in the output are already sorted by their strength, contained in the scaling field (plotted relative to the first component). We will skip the trial profiles and spatial phase maps for now; we will investigate them in separate sections below. Note that we use a FieldTrip plotting function, *ft_singleplotER*, for plotting the spatial amplitude maps. This function requires a known 'FieldTrip layout', which describes the spatial configuration of the sensors (cfg.layout = 'CTF151'; in this case). More information on layouts can be found here: http://www.fieldtriptoolbox.org/template/layout (multiple common EEG/MEG sensor configurations are supported). Also note that we use the *trialinfo* field to determine which of the trials belonged to the FIC (fully incongruent) and the FC (fully congruent) condition, using the event codes described during trial segmentation (*Preprocessing* in section 4*)*.

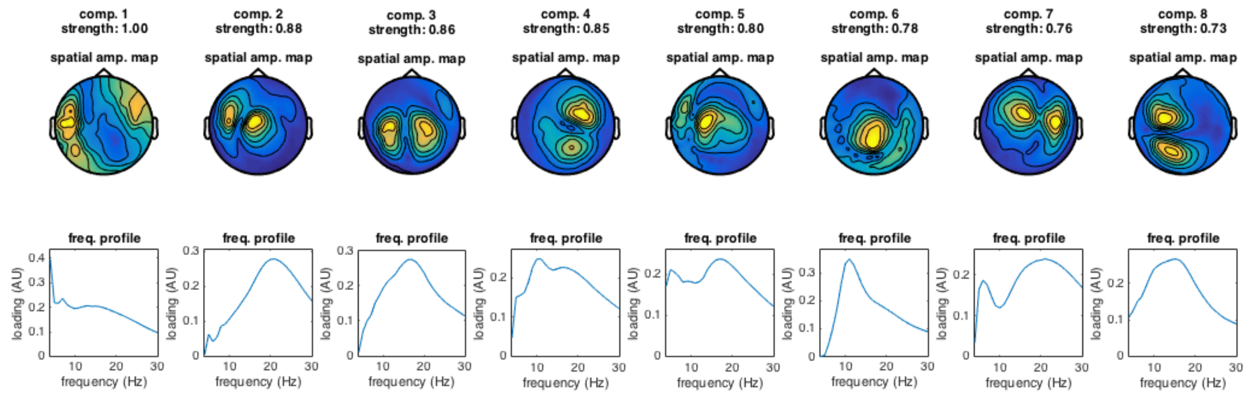

*Figure 6: all extracted components. (The color scale of the spatial amplitude maps starts at 0.)*

**Spatial and spectral content**

To find out more about the individual components we extracted, we inspect the spatial amplitude maps and the frequency profiles. When we look at component 2 for example, we see that its frequency profile is maximal in the beta range. As such, this component reflects beta oscillatory activity. We can further see that its spatial amplitude map has two blobs, roughly above sensorimotor cortex. This means the component likely reflects a so-called point source in sensory or motor cortex, which is discussed in more detail in the final section of this tutorial. We can make similar observations regarding the other components. Component 3 for example, is very similar, and also can be called a 'beta component'. The two blobs of its spatial amplitude map are more posterior, indicating a more posterior location of the source. Component number 6 is an 'alpha' component, and has a strong blob with a very weak second blob. Interestingly, component number 7 has both a peak in the theta and in the beta range. This likely means that oscillatory activity at those frequencies, occurred at approximately the same location with a similar relationship to the task, more on this below. When we look at the first component, its frequency profile looks like a typical power spectrum. This component might reflect an artifact (see below). Another observation we can make is that the component strengths at the top of the figure are all similar. That is, the extracted components explained similar amounts of variance.

Note, that due to the fact that our spectral analysis used time-windows that were shorter for higher frequencies, our frequency resolution decreases with frequency. As such, the frequency profiles are expect to be broader for higher frequencies, which can be seen in component 6. The left half of the frequency profile of this alpha component rises much quicker than that the right half decays, which is a necessary consequence of frequency-dependent window length.

All of the above aspects are important to observe, as all of these play an important role in interpreting what kind of oscillatory neural activity each component reflects. We didn't set out to investigate only the presence of oscillatory neural activity though. In the next section, we'll use each components trial profile to investigate whether the neural activity that they reflect, is modulated as a function of the task the subject performed. Before we move on however, there are several additional topics of interest regarding the spatial and spectral structure of components.

## A combination of two rhythms in one component

If two rhythms are combined, this indicates that the structure over sensors and over trials of the two rhythms is highly similar. Because of the split-reliability procedure, this is also highly consistent between splits of the data. A liberal interpretation of the combination of the two rhythms is that the same neuronal population produces them. As with all aspects of the extracted components, for additional certainty one can use the parameters of each component for a guided inspection of the input data, or perform an analysis of the residuals.

## Only relative parameter values can be interpreted

The absolute values of the parameters of components are arbitrary. Only the amplitude *ratios*, in case of the spatial amplitude map, the frequency profile, and the trial profile, and the between-sensor phase *differences* in the spatial phase maps, can be interpreted. For example, we cannot say that component 6 has the strongest alpha amplitude of all components. What we can say is that component 6, of all components, has the strongest relative alpha amplitude with respect to the other frequencies of component 6. That only relative parameters can be interpreted is due to an indeterminacy in the SPACE model, which it shares with PARAFAC, PCA and ICA.

## Lowest parameter loadings can indicate influence of noise per component

Under the assumption that no neural source is (1) equally present at all sensors, (2) equally involves every frequency, or, (3) equally present in every trial/epoch, the parameter loadings of a component should tend to 0. That is, the spatial amplitude map, the frequency profile, or the trial profile should have sensors, frequencies, or trials that are not involved in the component. Because of this, the lowest loading of a parameter set can be indicative of how strongly a component is influenced by noise in the data, as the lowest loading should be close to 0 under the above assumption. Though it is not displayed very clearly in the first SPACE reference paper, increasing the strength of simulated noise increased the level of the lowest loadings of the simulated components at frequencies and trials in which they were not present.

## A cautionary note on rhythmic components and 'artifacts'

The first component describes spectral structure in phase coupling over frequencies that is not commonly reported. Though $1/f^x$ type structure can be a signal of interest, this usually concerns spectral power, and not phase coupling. Given the spatial map of this component, and the broad frequency profile, it appears similar to phase coupling that is the result of a heartbeat artifact in MEG. Whatever the origin of this component, it may seem tempting to avoid cleaning the recordings, and assume artifacts will be captured by SPACE and do not affect the remaining components. This is a misconception, which also holds for ICA/PCA and many other techniques. The cleaner the data, the better these algorithms are at capturing structure we deem relevant. This is even more so when using a split-reliability, because artifacts, such as MEG jumps, are often unreliably present over splits, and will therefore cause the procedure to stop prematurely.

## Step 2: Analyzing task modulations of neural activity at the level of components

An important feature of rhythmic components is that they allow for a comparison of neural activity at the level of components, instead of at the level of sensors. This can be done using the trial profile, which describes, by a single value, how strongly each component was active in each trial. With these values we can, for example:

- compare strength of component-specific neural activity between conditions (comparable to comparing power/coherence over all sensors)
- compute correlations between component strengths and reaction times
- compare component strengths and its relation to response accuracy

We will do the first one below, followed by a discussion on how to extend analyses to the group-level.

```
% plot trial profile of components
figure
ncomp   = numel(nwaycomp.comp);
for icomp = 1:ncomp
  % plot trial profile (contained in the 3rd cell)
  subplot(1,ncomp,icomp)
  FICtrials = nwaycomp.comp{icomp}{3}(nwaycomp.trialinfo == 3);
  FCtrials  = nwaycomp.comp{icomp}{3}(nwaycomp.trialinfo == 9);
  nFICtrials = numel(FICtrials);
  nFCtrials  = numel(FCtrials);
  means = [mean(FICtrials) mean(FCtrials)];
  sems  = [std(FICtrials) std(FCtrials)] ./ sqrt([nFICtrials nFCtrials]);
  errorbar(means,sems)
  ylim = get(gca,'ylim'); set(gca,'ylim',[0 ylim(2)+max(sems)*3]);
  set(gca,'xlim',[1-.2 2+.2],'xtick',[1 2],'xticklabel',{'FIC','FC'})
  [h p,ci,stats] = ttest2(FICtrials,FCtrials);
  title({['comp. ' num2str(icomp)];...
        ['strength: ' num2str(nwaycomp.scaling(icomp)./nwaycomp.scaling(1),'%1.2f')];...
        '';...
        'trial profile';...
        ['FIC vs FC t = ' num2str(stats.tstat,'%1.2f')]})
end
```

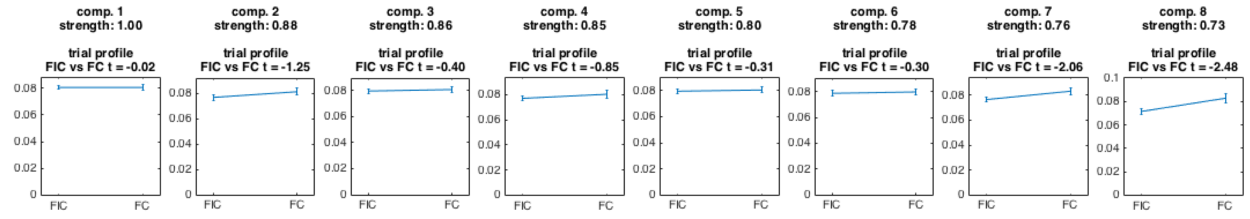

*Figure 7: trial profiles of all extracted components.*

### Single subject task modulations

To investigate task modulations of the neural activity the components reflect, we plotted their trial as a function of task condition (mean+/- SEM). Above each plot we expressed the difference between conditions as a t-value. We will only focus on those components whose t-value exceeds the arbitrary threshold of 2. *(Note, though we used a t-value, we could have used any other arbitrary coefficient reflecting to a condition difference. In essence, this t-value is nothing more than a difference in 'power', expressed in units that are not component-specific.).* The two components that surpass the threshold are component #7 and 8. Component #7 is the theta/beta component we highlighted above, which a dipolar sensor-level distribution that possibly matches a sensorimotor source location. Component #8 is a beta component as well, which has a dipolar sensor-level distribution possibly reflecting a left temporal dipole. Determining the exact location in the brain of the source a component reflects can be of interest, as is the case for component #8 in a language experiment.

To investigate the origin in the brain of components, follow-up analyses are necessary. As the spatial amplitude map of component #8 appears to be dipolar (see the last section for more details), a simple dipole fitting or scanning approach is very suitable (ideally using subject-specific anatomical information). For those components whose spatial amplitude maps are not as clearly dipolar, a different approach could be more suitable, e.g. projecting the spatial amplitude maps through a series of subject-specific spatial filters computed using a beamforming approach. These follow-up analyses are not the subject of the this tutorial. Additional info on these techniques can be found in the following two tutorials: http://www.fieldtriptoolbox.org/tutorial/natmeg/dipolefitting and http://www.fieldtriptoolbox.org/tutorial/beamformer.

**Group-level analysis of task modulations**

In the above, we found two components that showed differential activity between conditions. Though interesting, our goal is often not to make an inference tied to a specific individual, but rather one that holds across a population of interest. As such, we measure multiple subjects and perform some kind of aggregation over subjects. A very common approach is to compute a sensor-level average of some aspect of the data, e.g. power, over subjects. When extracting rhythmic components however, this is no longer possible (nor necessary as noise reduction), as each individual's components can only be analyzed separately. As such, to be certain of generality of results such as in Figure 6/7, we need to aggregate components in some other manner over subjects. Two examples of such an aggregation are described below. The statistical procedures used are simple; other more sensitive (multi-level) approaches are very feasible, but left to the reader.

Explorative hypotheses

The first concerns a scenario in when the main hypothesis is an explorative one. That is, is any oscillatory activity in any location different between conditions? In this scenario, we could first select, per subject, all components where the difference between conditions passes a certain threshold. Then, over subjects, we can determine how many of these components are similar, using the spatial amplitude maps (or a source-localized variant) and frequency profile. Similarity can be judged by e.g. the similarity coefficient described in section 4, and components considered as similar after a threshold. If a statistically significant proportion of subjects, using a simple binomial test, show the same components, we can be certain that oscillatory activity with this spatio-spectral signature is different between conditions.

Targeted hypotheses

In a second scenario the hypothesis is framed from the perspective of a particular location. That is, is any oscillatory activity in location A different between conditions? (A being either a spatial map, or a source localized variant). In this case we could first select, per subject, all components that originated from location A. Then, per subject, we would determine how many components' activity is different between conditions, using some threshold. If many 'location A' components show differences between conditions over subjects, then we can be reasonably certain that activity in location A differs between conditions. Whether this number of components is reasonable given the total number of 'location A' components can be determined by computing a subject-specific proportion, and then comparing the mean proportion over subjects to 0 using a simple one-sample t-test. *(Note: one should be careful in using within-subject proportions without a spatial/spectral/temporal restriction, as one can always increase the number of components, adding unreliable/noise components eventually, making the proportion arbitrarily low.)*

## Step 3: How to distinguish between types of neural sources that components can reflect

Rhythmic components can reflect different kinds of neural sources. In general, components reflect either one of two types: so-called oscillating point sources or phase-coupled oscillatory networks (PCNs). What kind of source is most applicable to a component can be determined using the spatial phase maps, which we will plot below.

### Components reflecting point sources

One type of source a component can reflect is an (oscillating) point source. A point source is a source that produces measurable activity, but has negligible dimensions. In the context of our extracranial measurements, this means that based on the activity that we see, we cannot infer anything about the size of the neural tissue that generated it. This also means that we cannot infer anything about the neuronal interactions 'inside' of such a source. It is a theoretical construct that has been very useful, because many activity patterns at the scalp are well described by point sources. (This is, at least partly, due to the distance from the sensors to the neural source.). A point source in the brain produces a dipolar potential distribution at the level of the scalp. One 'pole' is always of the opposite polarity of the other pole. Because we investigate *oscillating* point sources, for which the polarity flips every oscillatory cycle, the absolute polarity is not meaningful. When using spectral analysis to describe the phase and amplitude of oscillations, the oscillations at sensors in one pole are always of opposite phase of those at sensors in the other pole. That is, they have a phase difference of pi at every time point. Within pole sensors have the same phase, i.e. the phase difference between sensors within a pole is always 0.

Following from the above, a rhythmic component that reflects a point source can be identified by having (1) two poles in the spatial amplitude map, and (2) phase difference between poles in the spatial phase maps that are pi, for each frequency (that has high loading). As we will find out below, almost all of the components we extracted in this tutorial can be most accurately described as point sources.

**Components reflecting phase-coupled oscillatory networks (PCNs)**
The second type of source is a distributed source that reflects interacting neuronal populations, whose activity can be distinguished in our recordings, called a phase-coupled oscillatory network (PCN). On one extreme end, a PCN is a source distributed over the entire brain. On the other end, a PCN is a distributed source that is best described by two point sources. The activity that PCNs produce at the level of the scalp can result in phase differences that are different from pi. As such, any component whose between-sensor phase differences are different from pi, i.e. any component that is not a single point source, reflects a PCN. This can PCN be a for example a travelling wave, with between-sensor phase differences that increase with the distance between sensors, but also two phase-coupled point sources (having 4 visible poles). As mentioned briefly in the background, a possible cause of such phase differences are time delays between populations of neurons.

There is another way in which the components we extracted might reflect PCNs, and that is if between-component interactions exist, thus forming PCNs in combinations. This topic still has some uncertainties, and it is discussed in detail in the Supplementary Material of the second SPACE reference paper. It is not treated in the following.

**Visualizing and inspecting the spatial phase maps of components**

```
% plot spatial phase maps at peak and surrounding frequencies
figure
ncomp   = numel(nwaycomp.comp);
for icomp = 1:ncomp
  % determine peak frequency
  [dum peakind] = max(nwaycomp.comp{icomp}{2});
  if peakind==1, peakind = peakind + 1; end % exception handling for first component

  % determine peak sensor for renormalization
  [dum peaksensind] = max(nwaycomp.comp{icomp}{1});

  % set spatial plotting cfg
  cfg = [];
  cfg.layout        = 'CTF151';
  cfg.parameter     = 'spatmap';
  cfg.comment       = 'no';
  cfg.marker        = 'off';
  cfg.interpolation = 'nearest';
  cfg.style         = 'straight';
  cfg.maskparameter = 'mask';
  cfg.colormap      = hsv;
  cfg.zlim          = [-pi pi];

  % plot spatial phase map at peak and surrounding frequencies
  for imap = 1:3
    subplot(4,ncomp,icomp + (ncomp*(imap-1)))
    % gather phase and renormalize for easier interpretation
    L = exp(i*2*pi*nwaycomp.comp{icomp}{4}(:,(peakind+(imap-2))));
    L = L .* conj(L(peaksensind));
    % use spatial amplitude map as mask for easier interpretation
    mask = nwaycomp.comp{icomp}{1};
    mask(mask < median(mask)) = NaN;
    plotdat = [];
    plotdat.spatmap = angle(L);
    plotdat.mask     = mask;
    plotdat.label    = nwaycomp.label;
    plotdat.dimord   = 'chan';
    ft_topoplotER(cfg,plotdat);
    title({['comp. ' num2str(icomp)];...
           [' phase map ' num2str(nwaycomp.freq((peakind+(imap-2)))) 'Hz']})
  end
end
```

Note that the spatial phase maps in the output range from 0 to 1, and need to be rescaled to go from -pi to pi. We additionally renormalized the spatial phase maps such that the strongest sensor has a phase of 0, as they are normalized in the output to have a mean of 0, weighted by the spatial amplitude map. As noted above, we can freely do this, as the absolute phase is irrelevant, only the phase different with other sensors is meaningful. Furthermore, we masked away all phases of sensors whose spatial amplitude map weighting fell below its median.

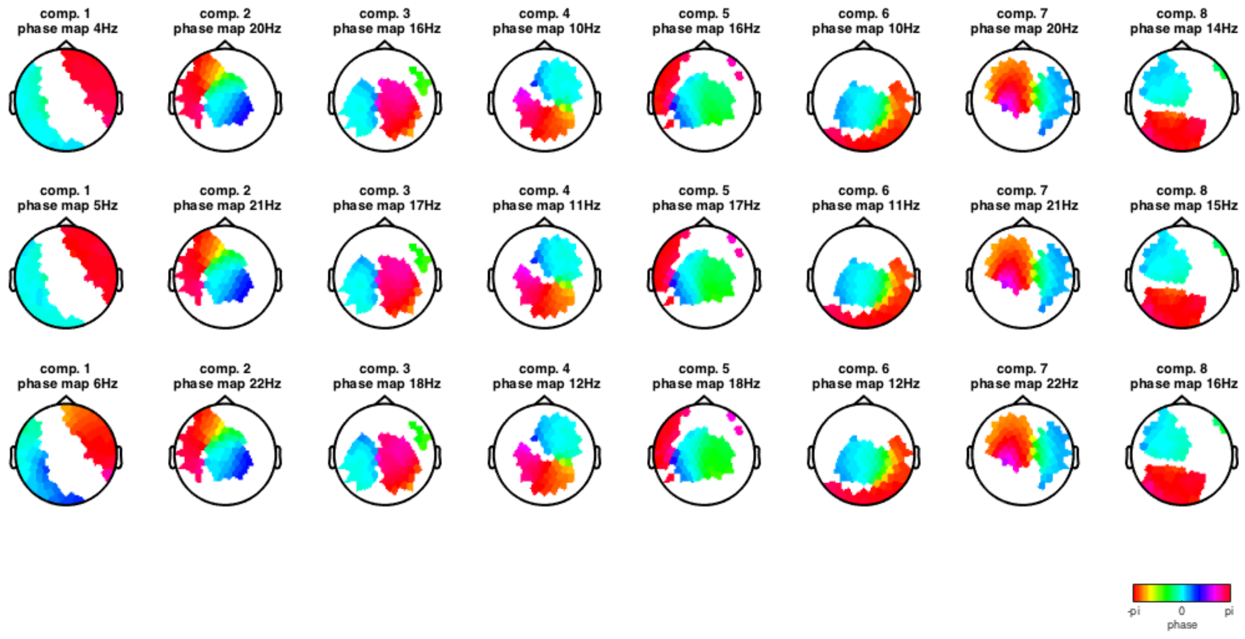

*Figure 8: spatial phase maps at peak frequency and surrounding frequencies*

**Spatial phase maps at the dominant frequency of a component**

Let's first focus on the peak frequency, in the middle row. In the spatial amplitude maps in Figure 6, we see that all components except the first appear to be point sources (two poles), some clearer than others. In the spatial phase maps we can observe that the phase differences between the poles of each of these components is close to pi (i.e. one pole is red and the other cyan), and that the phase differences within poles is close to 0. This holds even for those components, whose spatial amplitude map had a very weak second pole (#5 and #6). As such, the spatial phase maps at the peak frequency provide further evidence that the components describe point sources.

We can further observe that for some components the variability in between-sensor phase differences increases with the distance from the center of the pole (in terms of magnitude of the spatial amplitude maps). As the phase difference between the strongest sensors in each of the two poles of these components is still pi, they still likely reflect a point source. As such, this phase variability is likely a fitting error. (*Remember that the lower split-reliability coefficients belonged to the spatial phase maps? This indicates that they were more variable between splits, which can be due to noise.)* This could be the result of weak phase coupling between components. This topic is treated in the second SPACE reference paper, in its Supplementary.

**Phase differences over frequencies**

We now investigate the spatial phase maps at frequencies surrounding the peak frequency. We do this to be sure that the between-pole phase differences of pi we see at the peak frequency are not, by coincidence, limited to just the peak frequency. This could be the case for example, when a component reflects a PCN whose between-sensor phase differences are the result of a time delay, and this time delay is close to half the cycle length of the peak frequency (e.g. 50ms at a peak frequency of 10Hz).

The spatial phase maps at the neighboring frequencies are very similar to those at the peak frequency, providing further evidence that most components reflect point sources. If we want additional certainty, we could quantify this using a congruence coefficient as used in the previous section, and/or a combination of the two SPACE models referred to above. This is currently not implemented, but very feasible. As a rule of thumb for when to inspect the spatial phase maps at all frequencies, the less a spatial amplitude map appears like a dipole, e.g. a tripole or quadripole, the more important it is to inspect the spatial phase maps over frequencies, as these maps can produce insights into the nature of the neuronal source.

**What kind of sources did our components reflect?**

In the above, we observed that (1) the spatial amplitude maps have a dipolar sensor distribution, (2) the spatial phase maps show between-pole phase differences of pi and within pole phase differences of 0, and, (3) the spatial maps did not vary substantially over frequencies. As such, all components, except the first, likely reflect point sources, and not the interaction of neuronal populations. What kind of source the first component reflects is unclear. This requires additional analyses, e.g. using its parameters as a ‘filter’ to look at the raw data. *(For an example of a PCN in MEG recordings, see the supplementary material of the second SPACE reference paper.)*

## Concluding remarks

This tutorial covered how to extract rhythmic components from extracranial recordings, such as EEG/MEG, what practical issues arise when we do, and how to visualize components. Further, we demonstrated how we can use the trial profile to investigate task modulations of neural activity at the level of components, and talked briefly on how to conduct group-level analyses of components. Finally, we demonstrated how we can determine whether the components we extract reflect interacting neural populations or not. If you would like to see further examples of the latter (as we did not find interacting populations in the current example data), you can go through the Supplementary Material of the second SPACE reference paper. If you would like to see how SPACE can be used on intracranial data, such as ECoG, using SPACE-time to extract predominantly PCNs, there is another tutorial that focuses on this.

## References

1. van der Meij, R., Jacobs, J. & Maris, E. Uncovering phase-coupled oscillatory networks in electrophysiological data. *Hum Brain Mapp* **36**, 2655-2680 (2015).
2. van der Meij, R., van Ede, F., Maris, E. Rhythmic Components in Extracranial Brain Signals Reveal Multifaceted Task Modulation of Overlapping Neuronal Activity. *PLOS One* **11** (2016).
3. Harshman, R.A. Foundations of the PARAFAC procedure: model and conditions for an 'explanatory' multi-mode factor analysis. *UCLA Working Papers in Phonetics* **16**, 1-84 (1970).
4. Kiers, H.A.L., Ten Berge, J.M.F. & Bro, R. PARAFAC2 - Part I. A direct fitting algorithm for the PARAFAC2 model. *Journal of Chemometrics* **13**, 275-294 (1999).
5. Carrol, J.D. & Chang, J. Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition. *Psychometrika* **35** (1970).
6. Bro, R. PARAFAC. Tutorial and applications. *Chemometrics Intell. Lab. Syst.* **38**, 149-171 (1997).
7. Bruns, A. Fourier-, Hilbert- and wavelet-based signal analysis: are they really different approaches? *J. Neurosci. Methods* **137**, 321-332 (2004).