FIELDTRIP TOOLBOX TUTORIAL REPORT

PRADEEP KUMAR YADAV | BM18MTECH11005

TUTORIAL 2-

Dealing with artifacts

Background: what is an artifact?

- Some unwanted and unexpected feature in data.
- Artifacts are undesired data which is acquired from EEG and MEG.
- It may be physiological or non_physiological from origin.

How does FieldTrip manage artifacts?

first identify the artifacts and then subsequently removing them.

Artifacts can be visualize or can be predicted when we see the response .

- rejecting the piece of data containing the artifact (e.g. for a short-lived artifact)
- subtracting the spatio-temporal contribution of the artifact from the data (e.g. for line noise)

  fieldtrip provide the the function to detect the artifacts whether data is continuous or trial-based depending on whether your data is stored on disk or already in memory.

  Detecting the artifacts in continuous data to apply filters(band pass filter) and zoom out the recording.

Rejecting segments of data containing artifacts

In this type of artifact detection and rejection, pieces of data that contain artifacts are identified and removed from the data set

Manual/visual detection

In this manually and visualize the artifacts in continuous data trails.

And identify the affected with artifacts and reject the artifact.

The functions that are available for manual artifact detection are

ft_rejectvisual

The **ft_rejectvisual** function works only for segmented data (i.e. trials) that have already been read into memory.

```
FT_REJECTVISUAL shows the preprocessed data in all channels and/or trials to
 allow the user to make a visual selection of the data that should be
 rejected. The data can be displayed in a "summary" mode, in which case
 the variance (or another metric) in each channel and each trial is
 computed. Alternatively, all channels can be shown at once allowing
 paging through the trials, or all trials can be shown, allowing paging
 through the channels.
```

ft_databrowser

The **ft_databrowser** function works both for continuous and segmented data and also works with the data either on disk or already read into memory.

```
FT_DATABROWSER can be used for visual inspection of data. Artifacts that were
 detected by artifact functions (see FT_ARTIFACT_xxx functions where xxx is the type
 of artifact) are marked. Additionally data pieces can be marked and unmarked as
 artifact by manual selection. The output cfg contains the updated specification of
 the artifacts.
```

After detecting the time-segments with the artifacts, you should call **ft_rejectartifact** to remove them from your data (when the data is already in memory) or from your trial definition (when the data is still on disk).

a proper ICA unmixing of your data requires that the atypical artifacts (e.g. electrode movement, squid jumps) are removed **prior** to calling **ft_componentanalysis**. After you have determined what the bad components are, you can call **ft_rejectcomponent** to project the data back to the sensor level, excluding the bad components.

Automatic detection

Fieldtrip contains collection of functions to detect the artifacts automatically.

Although the automatic artifact detection algorithm works efficiently for well-known artifacts in well-behaved data, you should **not** use the automatic detection functions as your default method.

The available functions for automatic artifact detection are:

- **ft_artifact_clip**
- **ft_artifact_ecg**
- **ft_artifact_threshold**
- **ft_artifact_eog**
- **ft_artifact_jump**
- **ft_artifact_muscle**
- **ft_artifact_zvalue**

eog, jump and muscle detection functions are all just wrappers around **ft_artifact_zvalue** where the filter and padding options are set to reasonable defaults.

Removing artifacts from the data

If you use manual or automatic detection of time segments that contain an artifact, you usually would proceed to reject those segments from subsequent analysis with **ft_rejectartifact**

Subtracting spatial/temporal/spectral aspects of data reflecting artifacts

In this type of artifact detection and rejection, spatial/temporal/spectral aspects of the data that contain artifacts are identified and removed from the data set. For example, certain spectral components such as line noise, are subtracted from the data.

Using ICA-

Another commonly used approach is to make a linear decomposition of the data using methods such as ICA (independent component analysis) or PCA (principal component analysis). These methods consist of applying a set of spatial filters to the data, after which the data is no longer represented at the level of recorded (scalp) channels, but as a set of virtual channels or components.

If you use ICA to detect artifacts, you usually would proceed with projecting the decomposed data (excluding the artifact components) back to the sensor level. This is done with **ft_rejectcomponent**.

Further I will discuss more about ICA and how to detect and remove the artifacts.

Visual artifact rejection

Introduction

First we have to load the preprocessing data in matlab

```
load PreprocData dataFIC
```

for accurate measurement we should have clean or denoised data.

One of the factors that is difficult to control are the presence of artifacts in the data. These artifact are physiological or can result from the acquisition electronics. The strongest physiological artifacts stem from eye blinks, eye movements and head movements. Muscle artifact from swallowing and neck contraction can be a problem as well. Artifacts related to the electronics are 'SQUID jumps' or spikes seen in several channels. To start with, it is best to avoid those artifacts during the recording. You can instruct the subject not to blink during the trial, but instead give him some well-defined time between the trials in which he is allowed to blink. But of course there will always be some artifacts in the raw data.

keep in mind that it is a subjective decision to reject certain trials and keep other trials. Some data are bad and some data are good.

time-frequency analysis of power in the gamma band it is important to reject all trials with muscle artifacts, but for a ERF analysis it is more important to reject trials with drifts and eye artifacts.

The functions that are available for visual artifact detection are

- **ft_rejectvisual**
- **ft_databrowser**

already I have discussed above.

Procedure-

The following steps are taken to do visual artifact rejection

- Read the data into MATLAB using **ft_definetrial** and **ft_preprocessing**, as explained in the previous tutorial
- Visual inspection of the trials and rejection of artifacts using **ft_rejectvisual**
- Alternatively: use **ft_databrowser** and mark the artifacts manually by interactively paging trial by trial

Manual artifact rejection - display one trial at a time

The configuration option cfg.method provides the possibility of browsing through the data channel by channel (cfg.method= 'channel'), trial by trial (cfg.method = 'trial') .

The scaling of the plots is automatically adjusted according to the maximum amplitude over all channels. The scaling can be set using cfg.alim. For EOG/EEG channels cfg.alim=5e-5 (50 micro Volt) is a useful scale and for the MEG channels cfg.alim=1e-12 (10 fT/cm).

To browse through the data trial by trial while viewing all channels write

This is for MEG channels

```
cfg          = [];
cfg.method   = 'trial';
cfg.alim     = 1e-12;
dummy        = ft_rejectvisual(cfg,dataFIC);
```

the input is raw data with 152 channels and 87 trials

the call to "ft_selectdata" took 1 seconds

showing the data per trial, all channels at once

trial 1 marked as GOOD

trial 1 marked as BAD

trial 1 marked as GOOD

trial 1 marked as BAD

trial 2 marked as GOOD

trial 2 marked as GOOD

trial 3 marked as GOOD

trial 3 marked as BAD

trial 3 marked as GOOD

trial 4 marked as GOOD

trial 4 marked as GOOD

trial 5 marked as GOOD

trial 5 marked as GOOD

trial 6 marked as GOOD

trial 6 marked as GOOD

trial 7 marked as GOOD

trial 7 marked as GOOD

trial 8 marked as GOOD

trial 8 marked as GOOD

trial 9 marked as GOOD

trial 9 marked as GOOD

trial 10 marked as GOOD

trial 10 marked as GOOD

trial 11 marked as GOOD

channel MLC11 marked as BAD

trial 11 marked as GOOD

channel MLC11 marked as GOOD

trial 11 marked as GOOD

channel MLC11 marked as BAD

trial 11 marked as GOOD

channel MLC11 marked as GOOD

trial 11 marked as GOOD

channel MLC11 marked as BAD

channel MLC11 marked as GOOD

channel MLC11 marked as BAD

channel MLC11 marked as GOOD

trial 11 marked as GOOD

channel MLC11 marked as BAD

channel MLF44 marked as BAD

trial 11 marked as GOOD

channel MLF44 marked as GOOD

channel MLF44 marked as BAD

trial 11 marked as GOOD

channel MLF44 marked as GOOD

trial 11 marked as GOOD

trial 12 marked as GOOD

trial 12 marked as GOOD

trial 12 marked as BAD

trial 12 marked as GOOD

trial 13 marked as GOOD

trial 13 marked as GOOD

trial 13 marked as GOOD

channel MLC12 marked as BAD

trial 13 marked as GOOD

channel MLC12 marked as GOOD

channel MLC12 marked as BAD

trial 13 marked as GOOD

channel MLC12 marked as GOOD

trial 13 marked as GOOD

channel MLC12 marked as BAD

channel MLC12 marked as GOOD

trial 13 marked as GOOD

channel MLC12 marked as BAD

channel MLC12 marked as GOOD

trial 13 marked as GOOD

channel MLC12 marked as BAD

channel MLC12 marked as GOOD

channel MLC12 marked as BAD

trial 13 marked as GOOD

channel MLC12 marked as GOOD

channel MLC12 marked as BAD

trial 13 marked as GOOD

channel MLC12 marked as GOOD

unknown key pressed

unknown key pressed

unknown key pressed

trial 13 marked as GOOD

trial 13 marked as GOOD

trial 14 marked as GOOD

trial 14 marked as GOOD

trial 15 marked as GOOD

trial 15 marked as GOOD

trial 16 marked as GOOD

trial 17 marked as GOOD

trial 18 marked as GOOD

trial 18 marked as GOOD

trial 19 marked as GOOD

trial 19 marked as GOOD

trial 20 marked as GOOD

trial 20 marked as GOOD

trial 21 marked as BAD

trial 21 marked as BAD

trial 21 marked as BAD

trial 21 marked as BAD

trial 21 marked as BAD

trial 21 marked as BAD

trial 21 marked as BAD

trial 21 marked as BAD

trial 21 marked as BAD

trial 21 marked as BAD

trial 21 marked as BAD

trial 21 marked as BAD

trial 21 marked as GOOD

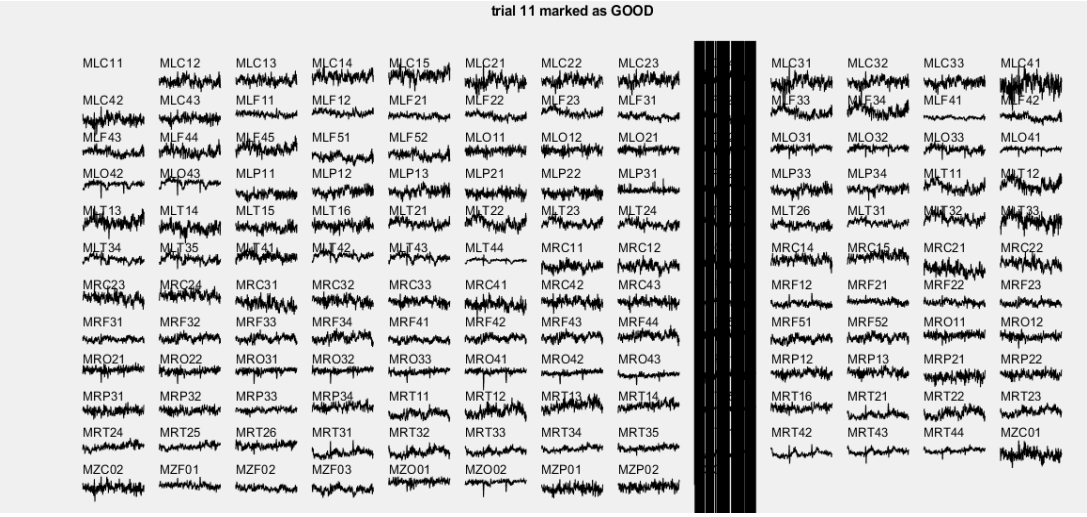
trial 21 marked as GOOD

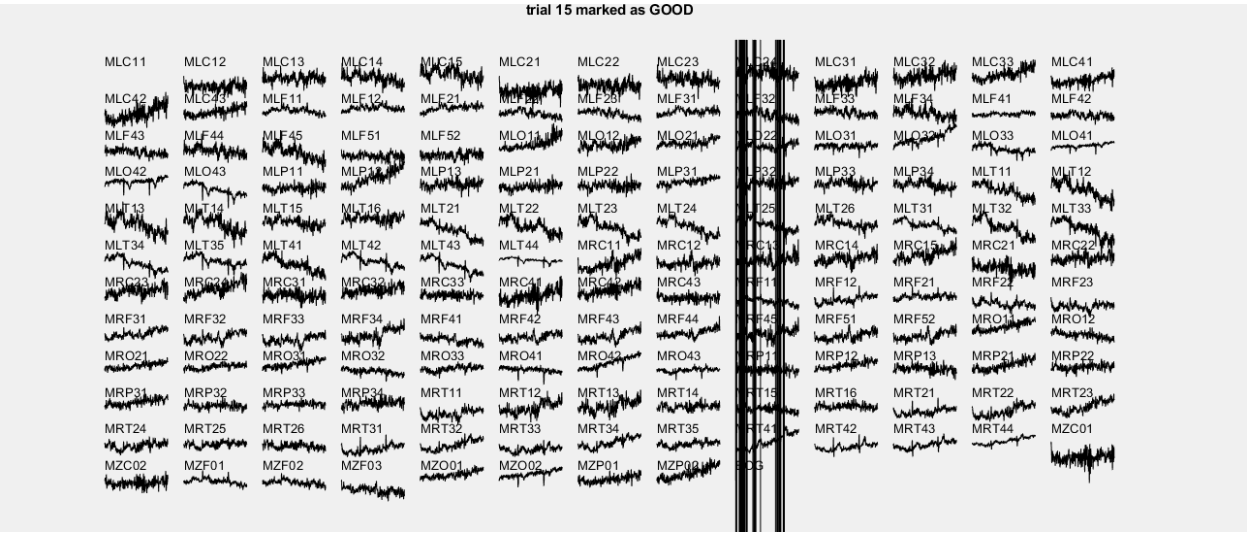
trial 21 marked as GOOD

trial 21 marked as BAD

trial 21 marked as BAD

trial 21 marked as GOOD





If your dataset contains MEG and EEG channels (like this dataset), the MEG and EEG channels are scaled differently when using only cfg.alim (the EEG channels show up as big black bars on the screen). One of the reasons to record EOG, EMG or ECG is to check these channels while identifying eye, muscle and heart artifacts. The following code can be used to scale MEG and EEG channels both properly.

In trial 15 notice the slower drift observed over a larger group of sensors. This is most likely due to a head movement.

```
cfg          = [];
cfg.method   = 'trial';
cfg.alim     = 1e-12;
cfg.megscale = 1;
cfg.eogscale = 5e-8;
dummy        = ft_rejectvisual(cfg,dataFIC);
```

cfg = [];

cfg.method = 'trial';

cfg.alim = 1e-12;

cfg.megscale = 1;

cfg.eogscale = 5e-8;

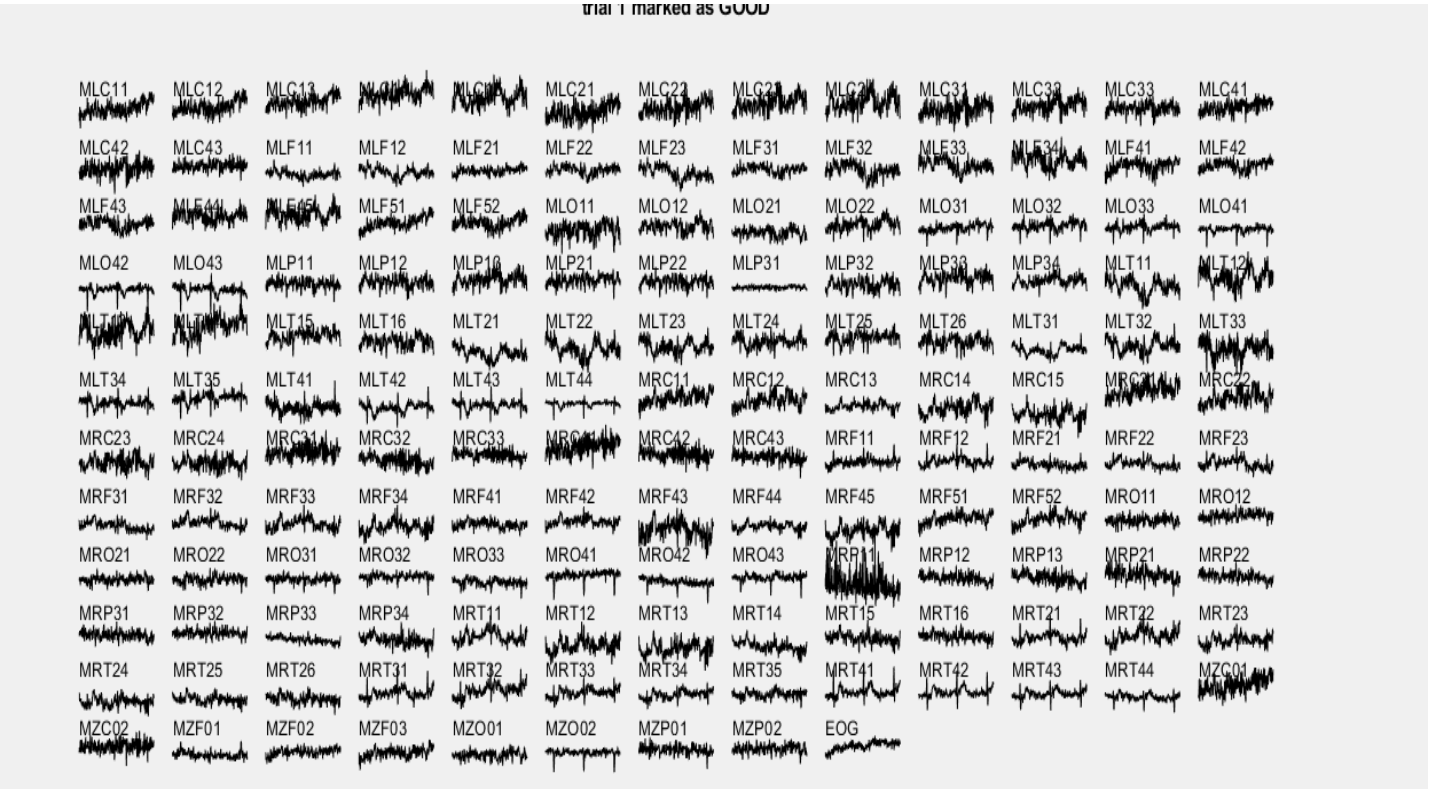
dummy = ft_rejectvisual(cfg,dataFIC);

the input is raw data with 152 channels and 87 trials

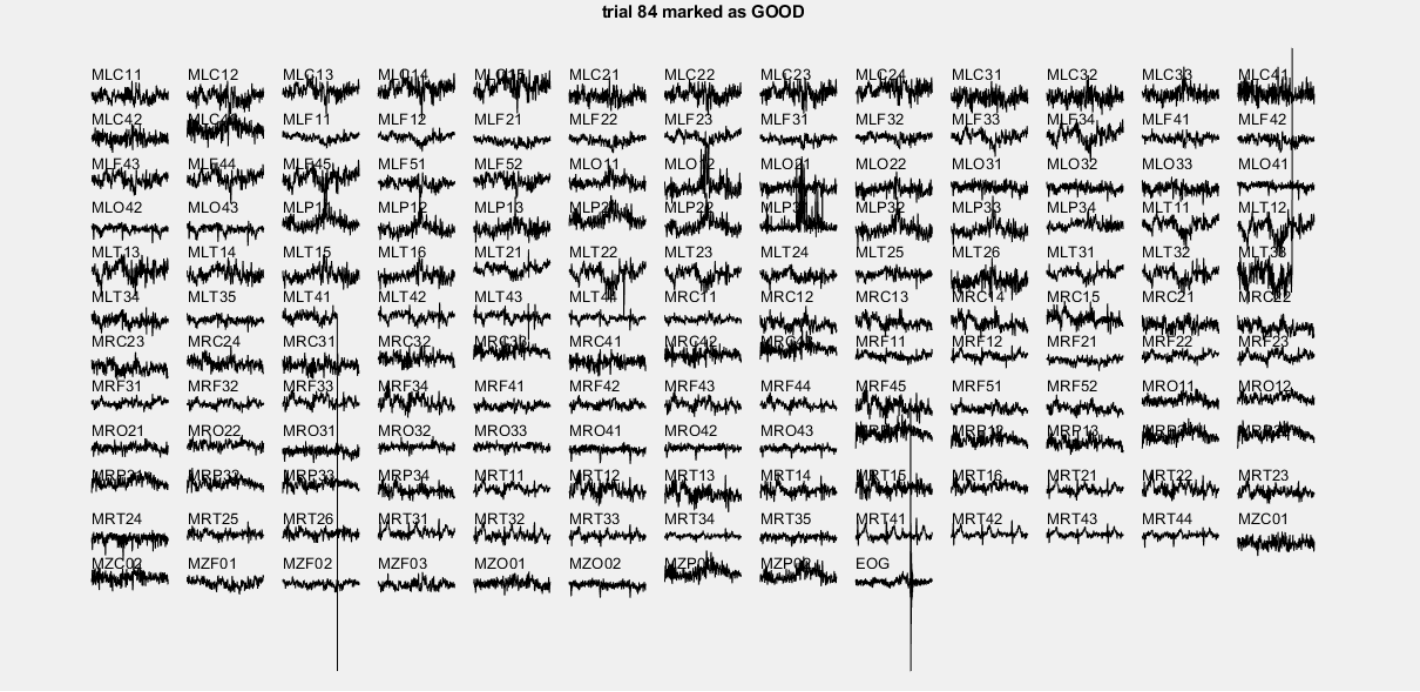
the call to "ft_selectdata" took 0 seconds

showing the scaled per trial, all channels at once

trial 1 marked as GOOD



Trial 84 shows an artifact which is caused by the electronics. Notice the jump in sensor MLT41.



If you would like to keep track of which trials you reject, keep in mind that the trial numbers change when you call **ft_rejectvisual** more than once.

There are 87 trials in your data and first you reject trial 15, 36 and 39. Then trial number 87 becomes trial number 84. Later when you also want to reject trials 42, 43, 45 ,49, 50, 81, 82 and 84 you should be very careful and subtract 3 from all the old trial numbers. If you would like to know which trials you rejected, it is best to call ft_rejectvisual only once.

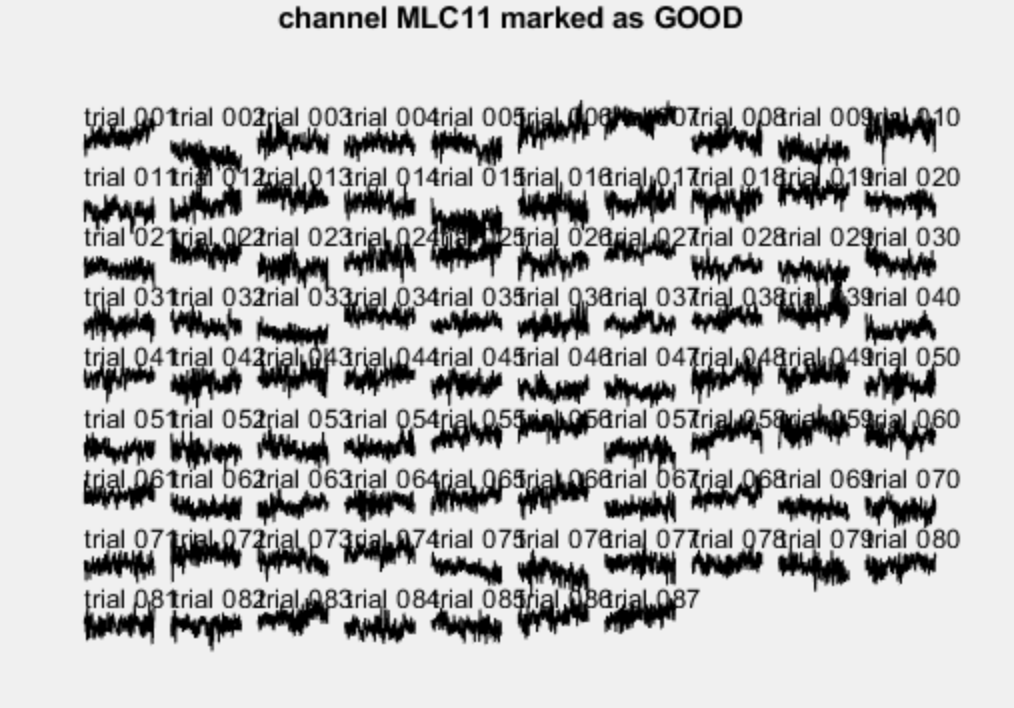
Manual artifact rejection - display one channel at a time

It can also be convenient to view data from one channel at a time. This can be particularly relevant for the EOG channel.

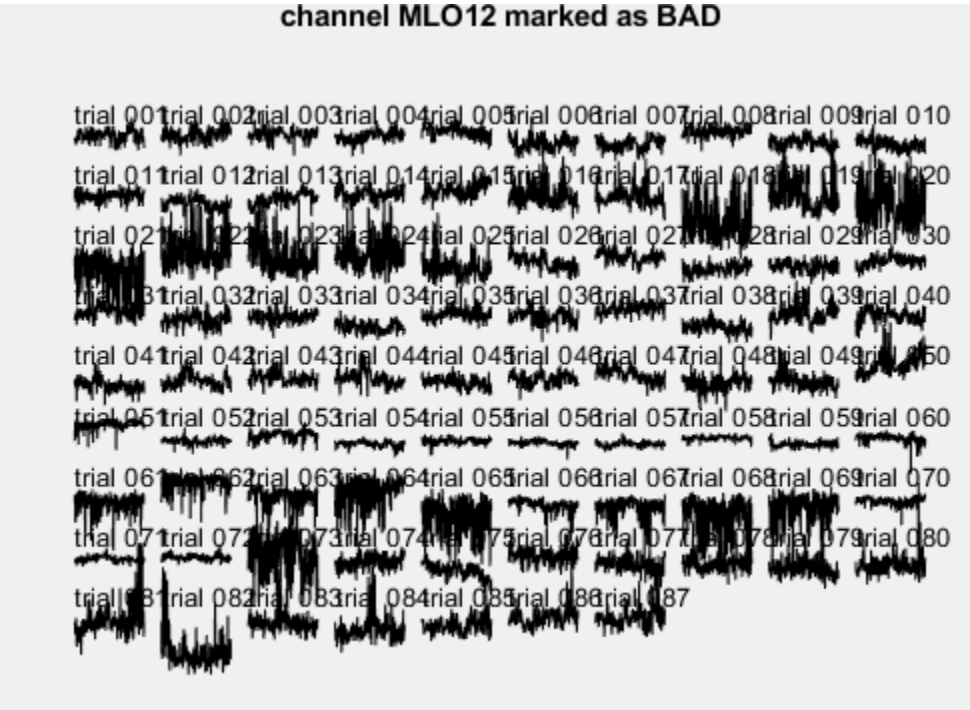
```
cfg          = [];
cfg.method   = 'channel';
cfg.alim     = 1e-12;
cfg.megscale = 1;
cfg.eogscale = 5e-8;
dummy        = ft_rejectvisual(cfg,dataFIC);
```

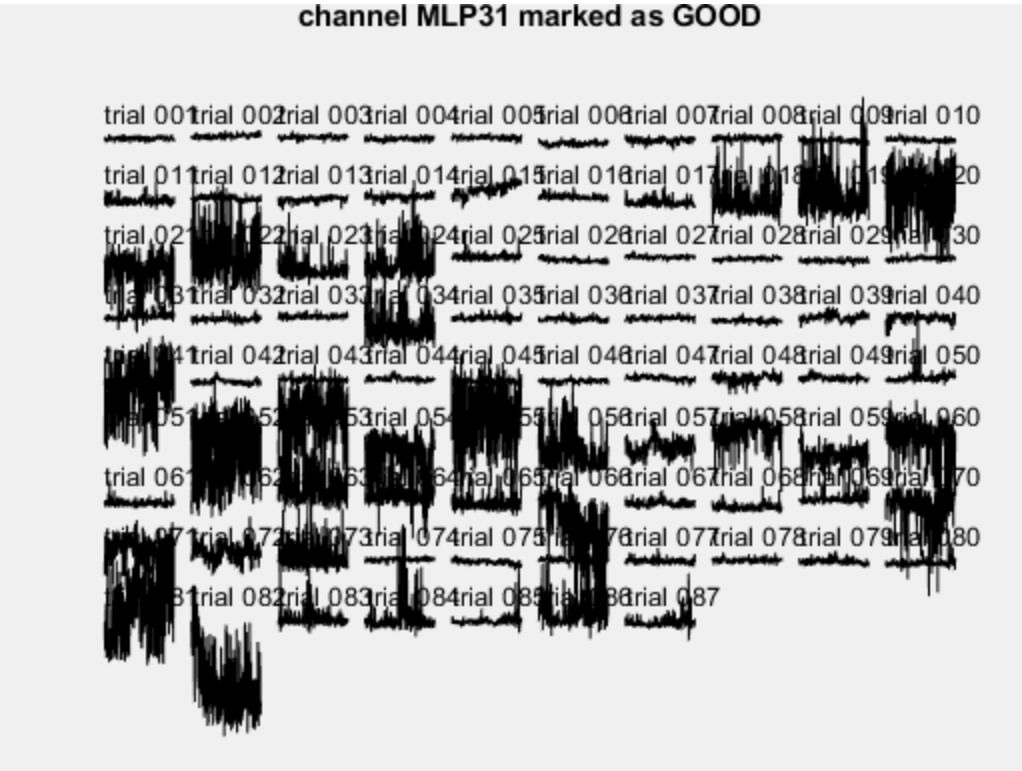
showing the scaled data per channel, all trials at once

channel MLC11 marked as GOOD



Click through the data using the > button. While clicking through all the trials you see that channels MLO12 and MLP31 contain a lot of artifacts (see the figure below). They should be marked as ‘bad’. After pressing the ‘quit’ button the channels marked ‘bad’ are now removed from the data structure.





Manual artifact rejection - display a summary

To produce an overview of the data choose the cfg.method 'summary

```
cfg          = [];
cfg.method   = 'summary';
cfg.alim     = 1e-12;
dummy        = ft_rejectvisual(cfg,dataFIC);
```

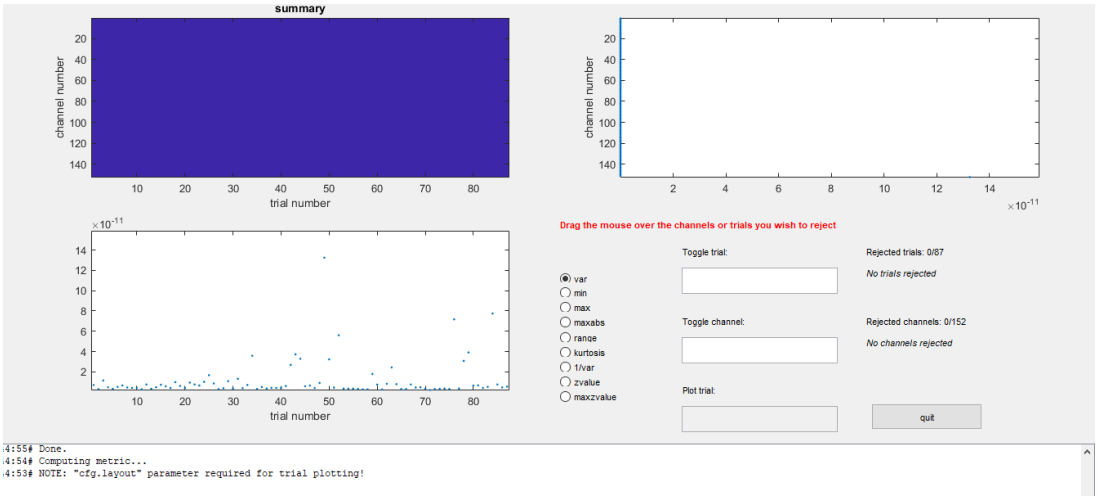
the input is raw data with 152 channels and 87 trials

the call to "ft_selectdata" took 0 seconds

showing a summary of the data for all channels and trials

computing metric [-|]computing metric [----------/]computing metric [----------

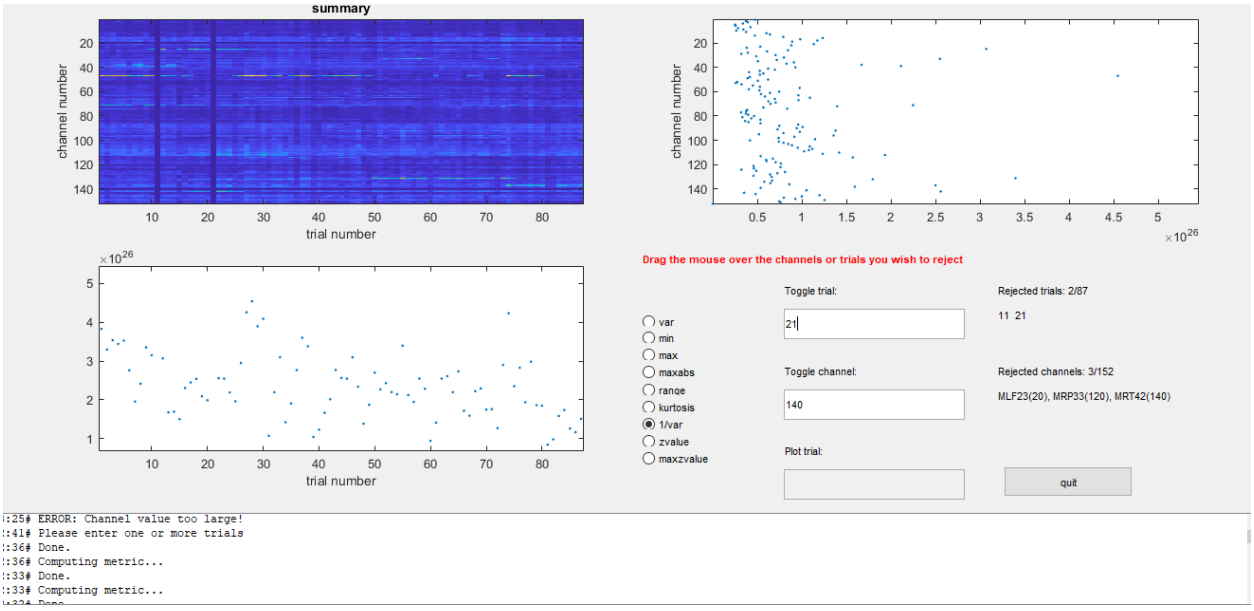
This gives you a plot with the variance for each channel and trial.



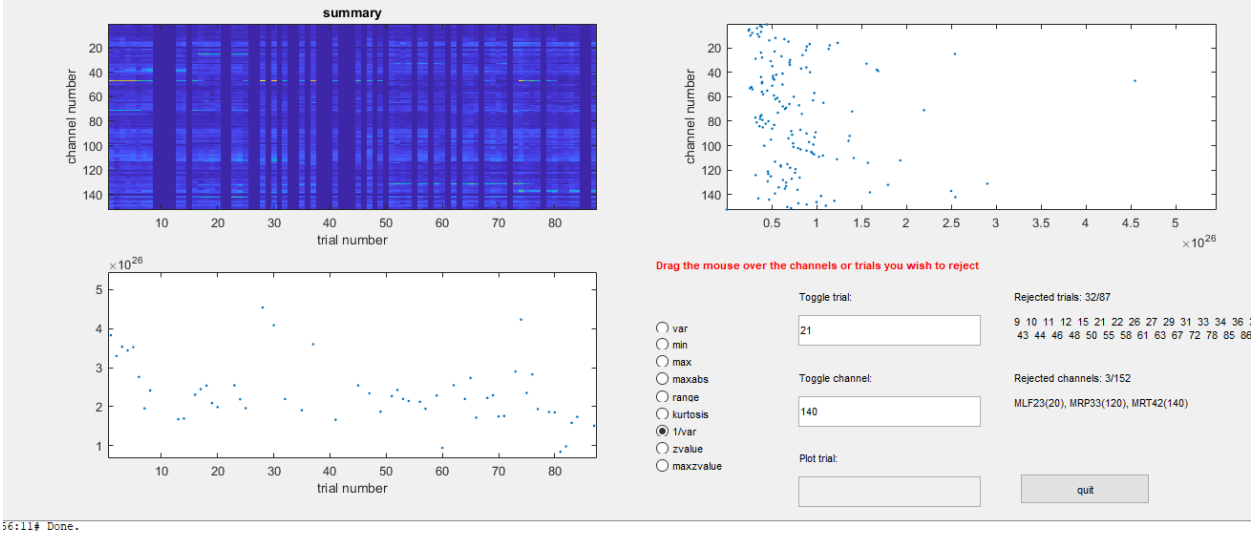
You should note that there is one channel which has a very high variance. That is the EOG channel, which contains numbers in uV which are of a very different order of magnitude than all MEG channels in T. Toggling the EOG channel will also change the figure with the maximal variance per trial (second row, left) a lot. Then you only see the variance in each trial in the MEG channels.

The command window allows for toggling trials and channels on and off.

Before pressing the 'Quit' button, you can always toggle the channels/trials back on, by using the edit boxes 'Toggle trial' or 'Toggle channel'.



After drading the mouse on channel.rejecting the trail and channel according to the requirement




```
56:11# Done.
56:10# Computing metric...
55:59# Done.
55:58# Computing metric...
53:25# ERROR: Channel value too large!
52:41# Please enter one or more trials
```

After quitting, the trials/channels will be rejected from the data set and the command line output appears as follow

55 trials marked as GOOD, 32 trials marked as BAD

149 channels marked as GOOD, 3 channels marked as BAD

the following channels were removed: MLF23, MRP33, MRT42

the following trials were removed: 9, 10, 11, 12, 15, 21, 22, 26, 27, 29, 31, 33, 34, 36, 38, 39, 40, 42, 43, 44, 46, 48, 50, 55, 58, 61, 63, 67, 72, 78, 85, 86

the call to "ft_selectdata" took 0 seconds

the call to "ft_rejectvisual" took 910 seconds

This operation could be repeated for each of the metrics, by clicking on the different radio buttons ‘var’, ‘min’, ‘max’, etc.

he summary mode in **ft_rejectvisual** has been primarily designed to visually screen for artefacts in channels of a consistent type, i.e. in this example only for the axial MEG gradiometers.

You can use the following options in **ft_rejectvisual** to apply a scaling to the channels prior to visualisation: *cfg.eegscale, cfg.eogscale, cfg.ecgscale, cfg.emgscale, cfg.megscale, cfg.gradscale* and *cfg.magscale*

You can also call **ft_rejectvisual** multiple times, once for every type of channels in your data. If you use *cfg.keepchannel='yes'*, channels will not be removed from the data on the subsequent calls. For example:

```
cfg = [];
cfg.method = 'summary'
cfg.keepchannel = 'yes';

cfg.channel = 'MEGMAG';
clean1  = ft_rejectvisual(cfg, orig);

cfg.channel = 'MEGGRAD';
clean2  = ft_rejectvisual(cfg, clean1);

cfg.channel = 'EEG';
clean3  = ft_rejectvisual(cfg, clean2);
```

You can repeat this for the initially congruent (IC) condition. To detect all the artifacts use **ft_rejectvisual** with all 3 methods (trial, channel and summary) like in the examples above.

```
clear all
load PreprocData dataIC

cfg          = [];
cfg.method   = 'trial'; % also try cfg.method = 'channel' and cfg.method = 'summary'
cfg.alim     = 1e-12;
cfg.megscale = 1;
cfg.eogscale = 5e-8;
dummy        = ft_rejectvisual(cfg,dataIC);
```

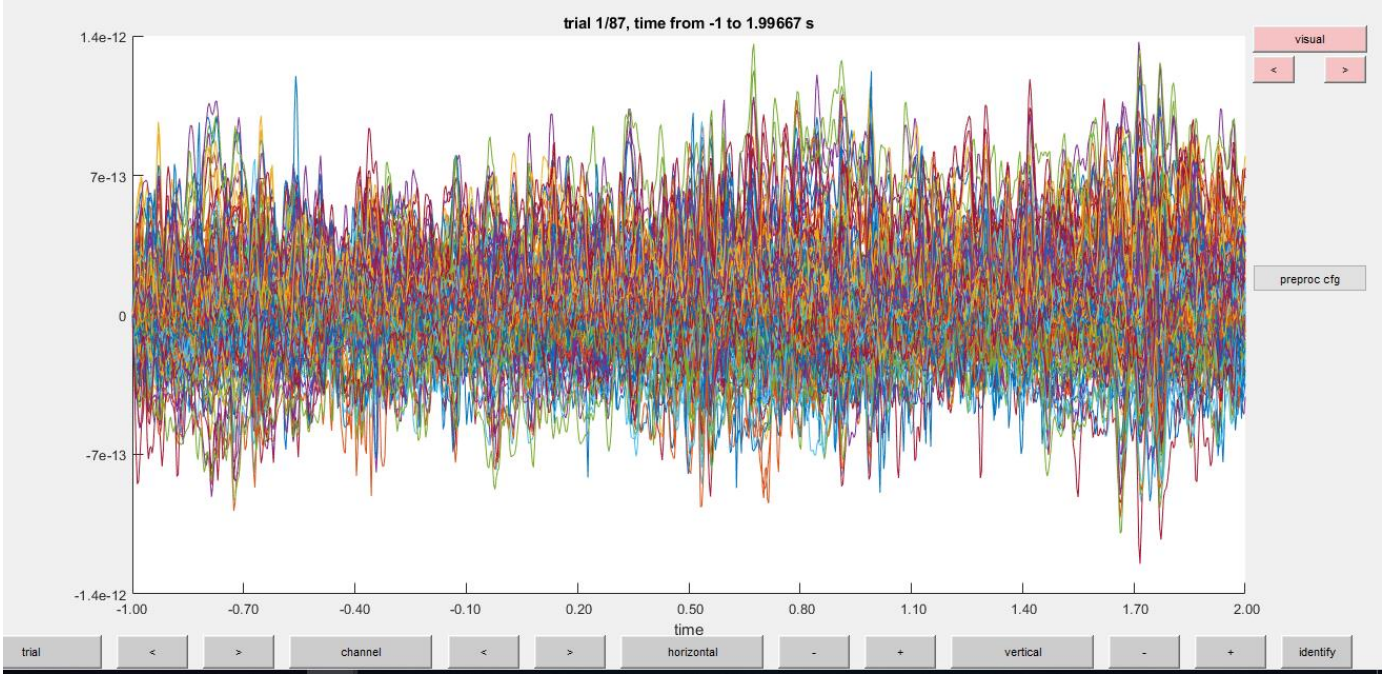
Trials 1, 2, 3, 4, 14, 15, 16, 17, 20, 35, 39, 40, 47, 78, 79, 80, 86 contain various artifacts, classify these as 'BAD'. Also reject the channels MLO12 and MLP31.

That already done previously.

Use ft_databrowser to mark the artifacts manually

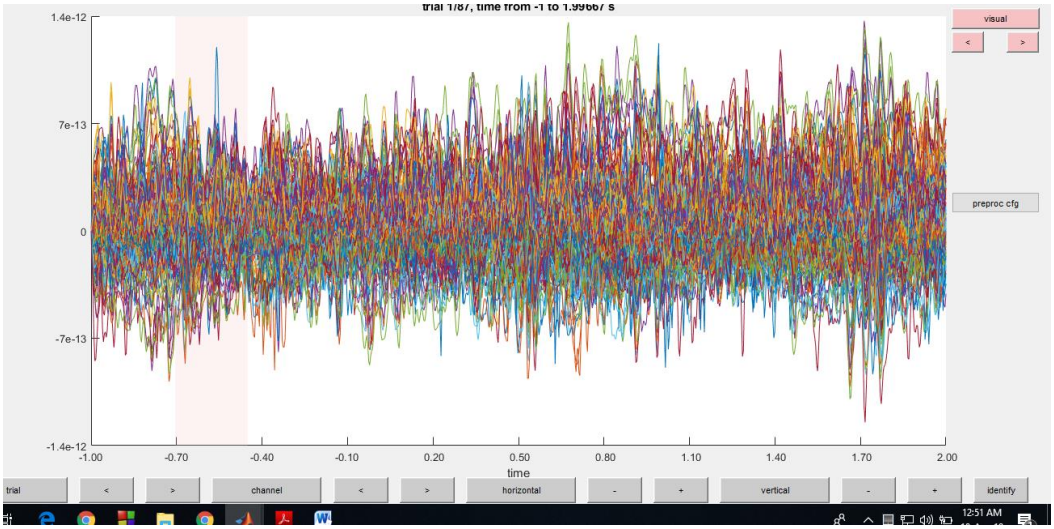
n alternative way to remove artifacts is to page through the butterfly plots of the single trials, by using the ft_databrowser function. Call the function like

```
% first select only the MEG channels
cfg = [];
cfg.channel = 'MEG';
data = ft_preprocessing(cfg,dataFIC);
% open the browser and page through the trials
cfg=[];
cfg.channel = 'MEG';
artf=ft_databrowser(cfg,dataFIC);
```

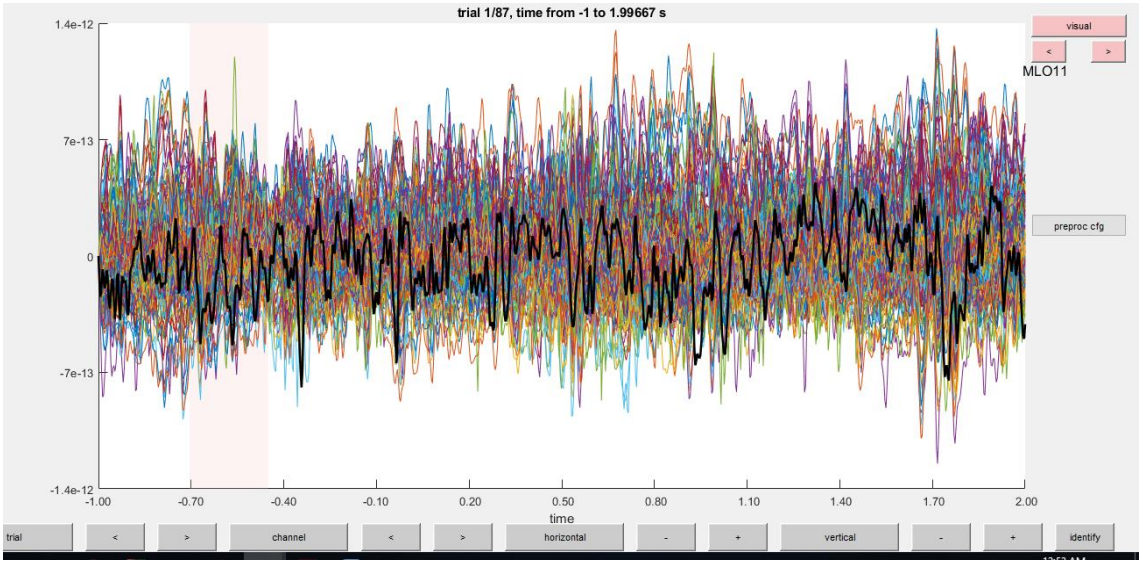


The resulting variable contains the file

artf.artfctdef.visual.artifact = [begartf endartf]

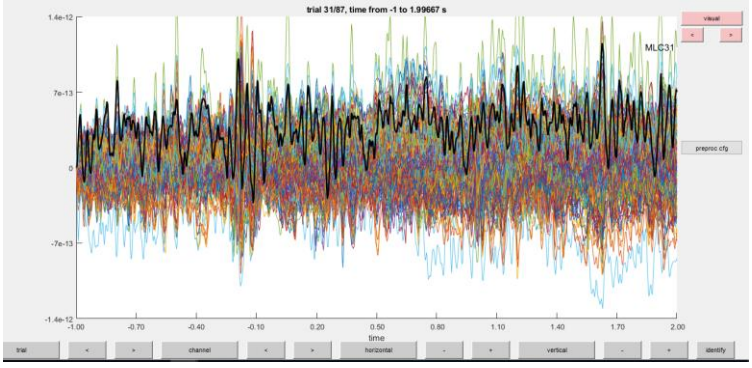


The artifact is detected and dragged by mouse clearly shown in graph.



I understood that this graph is summation of all the channel data in first trail.

We can change the trail and can detect the artifacts manually and select it and remove it.



Summary

Per condition, the following trials contain artifacts:

- FIC: 15, 36, 39, 42, 43, 49, 50, 81, 82, 84
- IC: 1, 2, 3, 4, 14, 15, 16, 17, 20, 35, 39, 40, 47, 78, 79, 80, 86
- FC: 2, 3, 4, 30, 39, 40, 41, 45, 46, 47, 51, 53, 59, 77, 85
- Channels MLO12 and MLP31 are removed because of artifacts.

Automatic artifact rejection

Automatic artifact rejection in FieldTrip is a sophisticated and complicated approach to artifact rejection

Introduction

Before further analysis in any of the other tutorials, it is best to have artifact free data. Within FieldTrip you can choose to do visual/manual or automatic artifact detection and rejection.

Background

For a successful analysis of EEG or MEG signals, “clean” data is required. That means that you should try to reduce variance in the data due to factors unrelated to your experimental conditions.

Procedure

The following steps are used to detect artifacts in FieldTrip’s automatic artifact rejection (see figure below)

1. Defining segments of interest using ft_definetrial
2. Detecting artifacts using ft_artifact_zvalue, this consists of
    - Reading the data (with padding) from disk
    - Filtering the data
    - Z-transforming the filtered data and averaging it over channels
    - Threshold the accumulated z-score

Artifact definition a two column array with the onset and offset sample number of every detected artifact.

I. Reading the data (with padding) from disk

Physiological data is often very large and takes system memory. For this reason it is most efficient to read only the data you need from disk instead of reading in all the data first and selecting interesting segments later.

Read the data and determine the artifacts.

Try to remove the artifacts from the data.

as in all the cases where we use trial- and filter-padding more data is needed, these cannot be applied and you will become much more sensitive to e.g. filter issues. This will also be an issue when data is not recorded continuously.

II. Filtering the data

We can optimize filter settings for EOG and SQUID jump artifacts.

III. Z-transforming the filtered data and averaging it over channels

Artifact detection approach-

1)they occur sparsely in time, i.e. not all the time.

2) they are reflected by larger amplitudes than the brain data (possibly or especially in a particular frequency band).

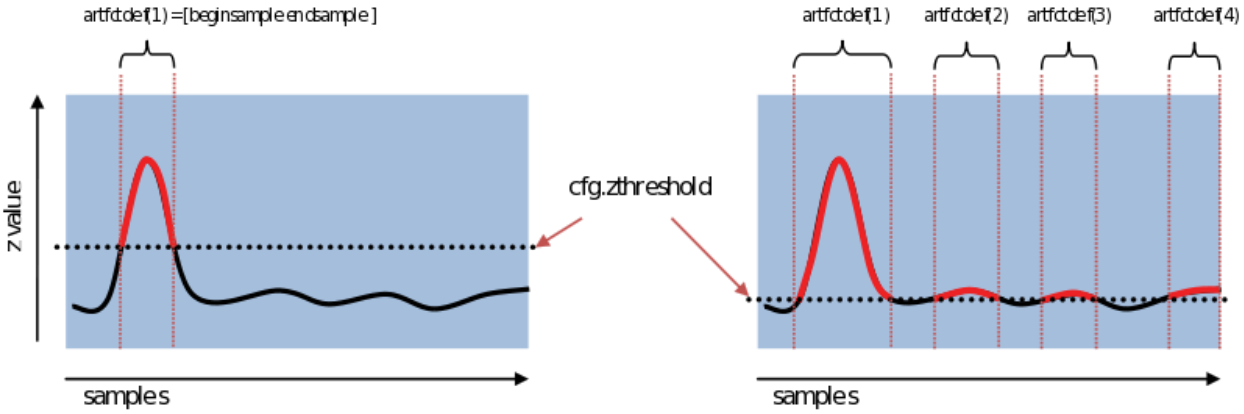
3) they occur over multiple channels/electrodes.

After Filtering ,transformed

1. Per channel/electrode the amplitude of the signal over time is calculated (the Hilbert envelope)
2. Per channel/electrode its mean and standard deviation is calculated (over all samples)
3. Per channel/electrode every timepoint is z-normalized (mean subtracted and divided by standard deviation)

4. Per timepoint these z-values are averaged. Since an artifact might occur on any and often on more than one electrode (think of eyeblinks and muscle artifacts), averaging z-values over channels/electrodes allows evidence for an artifact to accumulate. *This results in one timecourse representing standardized deviations from the mean of all channels.*

IV. Thresholding the accumulated z-score



Setting a higher or lower z-value threshold will make the detection of artifacts more conservative or liberal.

Padding

Artifact padding

(''cfg.artfctdef.xxx.artpadding'') is used

Trial padding

(''cfg.artfctdef.xxx.trlpadding'') is used

Filter padding

(''cfg.artfctdef.xxx.fltpadding'') is used

Artifact rejection

First we need to define our trial

```
cfg                     = [];
cfg.dataset             = 'ArtifactMEG.ds';
cfg.headerformat        = 'ctf_ds';
cfg.dataformat          = 'ctf_ds';
cfg.trialdef.eventtype = 'trial';
cfg                     = ft_definetrial(cfg);
trl                     = cfg.trl(1:50,:); % we'll only use the first 50 trials for
this example
```

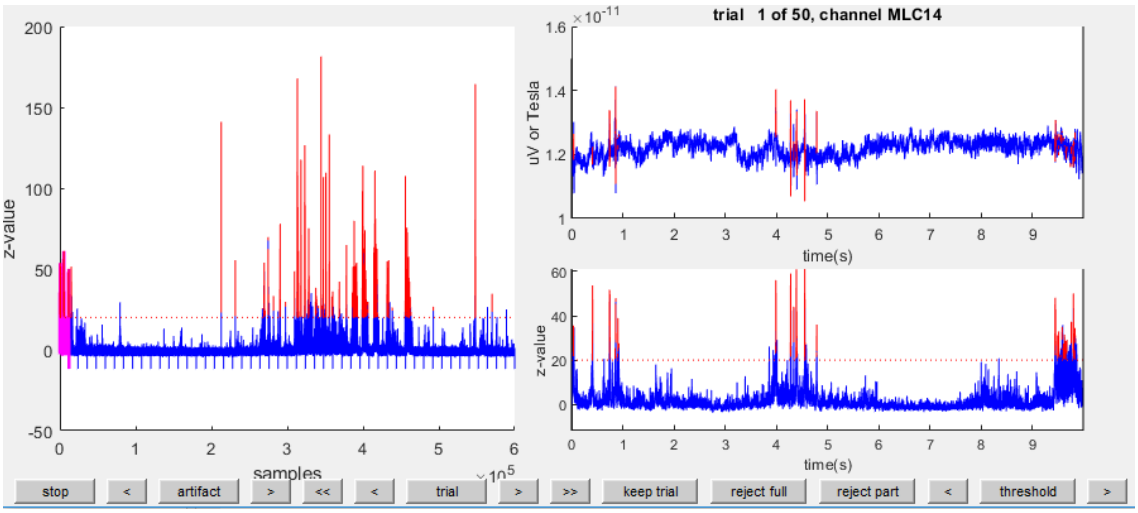
getRawCTFBalanceCoefs : Sensor Q11 appears in ds.res4.scrr, but not in ds.res4.chanNames

Jump artifact detection

For detecting jump artifacts, begin with the following parameter

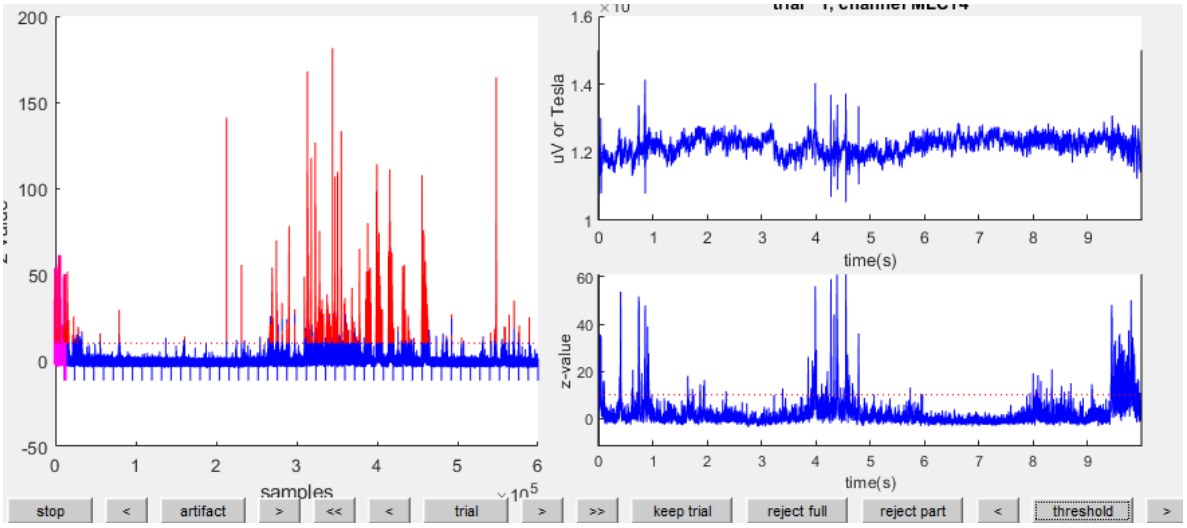
```
% jump
cfg = [];
cfg.trl = trl;
cfg.datafile = 'ArtifactMEG.ds';
cfg.headerfile = 'ArtifactMEG.ds';
cfg.continuous = 'yes';

% channel selection, cutoff and padding
cfg.artfctdef.zvalue.channel = 'MEG';
cfg.artfctdef.zvalue.cutoff = 20;
cfg.artfctdef.zvalue.trlpadding = 0;
cfg.artfctdef.zvalue.artpadding = 0;
cfg.artfctdef.zvalue.fltpadding = 0;

% algorithmic parameters
cfg.artfctdef.zvalue.cumulative = 'yes';
cfg.artfctdef.zvalue.medianfilter = 'yes';
cfg.artfctdef.zvalue.medianfiltord = 9;
cfg.artfctdef.zvalue.absdiff = 'yes';

% make the process interactive
cfg.artfctdef.zvalue.interactive = 'yes';

[cfg, artifact_jump] = ft_artifact_zvalue(cfg);
```

Interactive figure (below) of ft_artifact_zvalue. The left panel shows the z-score of the processed data, along with the threshold. Suprathreshold data points are marked in red. The lower right panel shows the z-score of the processed data for a particular trial, and the upper right panel shows the unprocessed data of the channel that contributed most to the (cumulated) z-score. You can browse through the trials using the buttons at the bottom of the figure. Also, you can adjust the threshold, and manually keep/reject trials.



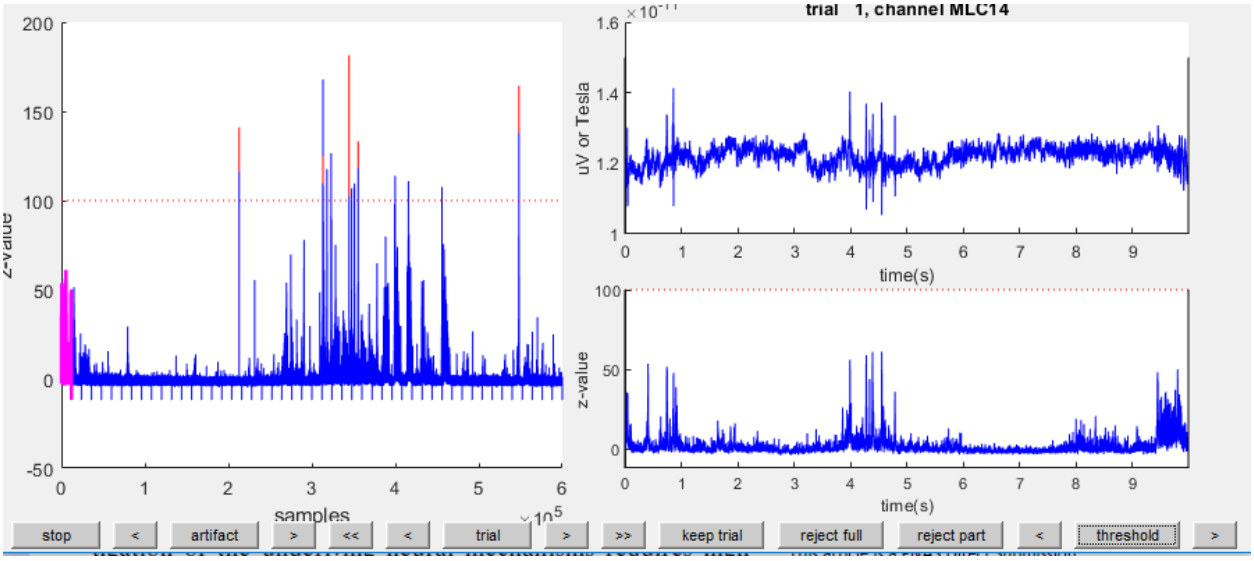
See how the graph is changing due to changing the z valve ,it reject the upper part.

Z value =10

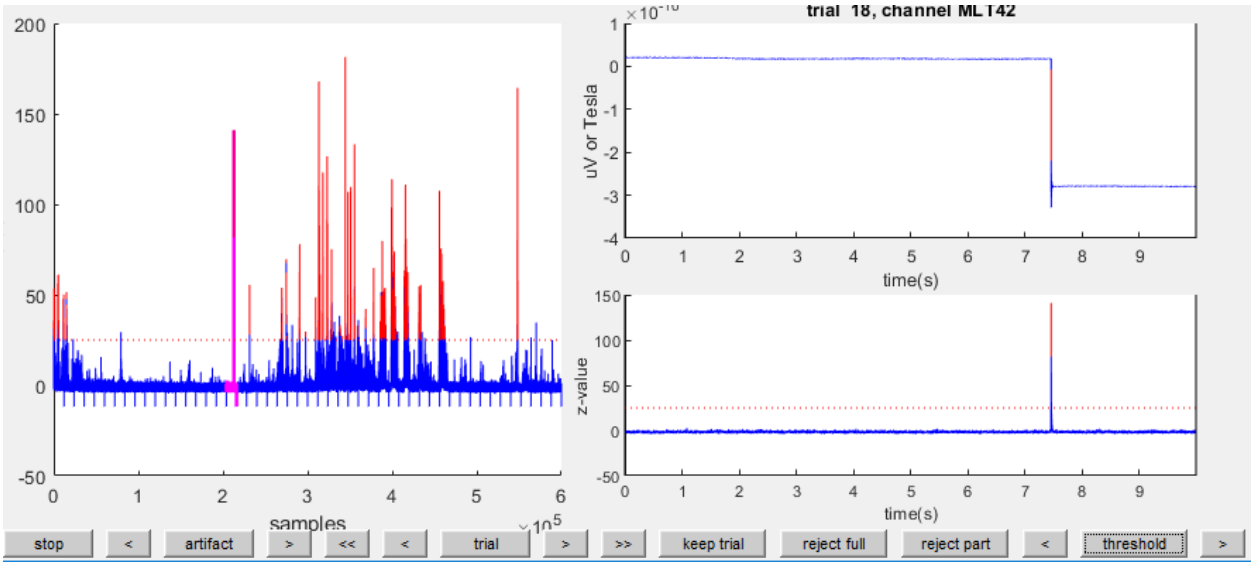


a lot of data points are detected to be a 'jump' artifact, although they seem more often 'muscular' in nature.

Z _value is 100 .



A typical SQUID bump can be seen in trail 18.

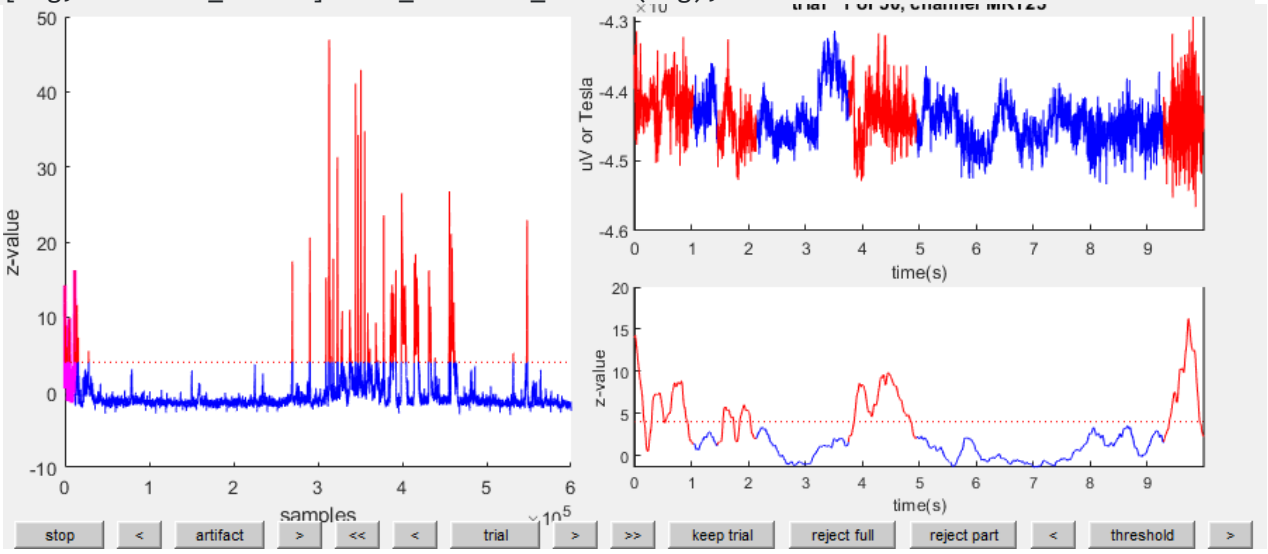


Detection of muscle artifacts

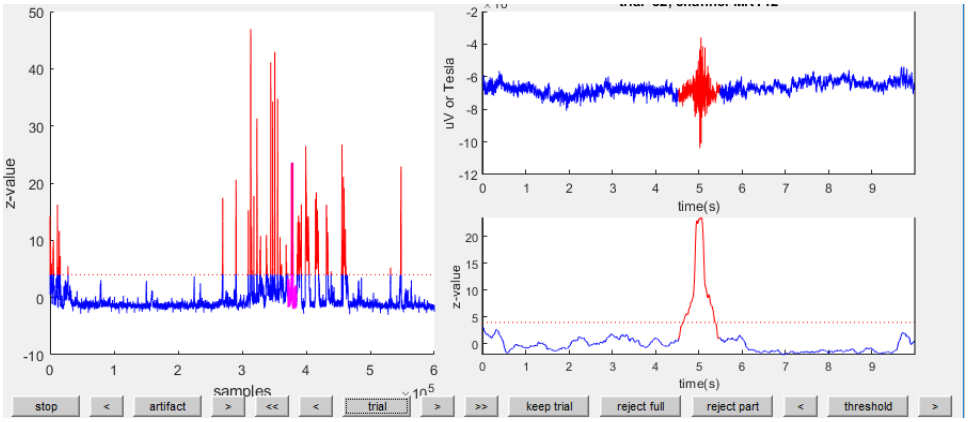
The same way as **ft_artifact_zvalue** is used to detect jump artifacts, it can be used to detect muscle artifacts.

```
% muscle
cfg            = [];
cfg.trl        = trl;
cfg.datafile   = 'ArtifactMEG.ds';
cfg.headerfile = 'ArtifactMEG.ds';
cfg.continuous = 'yes';
```

```
% channel selection, cutoff and padding
cfg.artfctdef.zvalue.channel      = 'MRT*';
cfg.artfctdef.zvalue.cutoff       = 4;
cfg.artfctdef.zvalue.trlpadding   = 0;
cfg.artfctdef.zvalue.fltpadding   = 0;
cfg.artfctdef.zvalue.artpadding   = 0.1;

% algorithmic parameters
cfg.artfctdef.zvalue.bpfilter     = 'yes';
cfg.artfctdef.zvalue.bpfreq       = [110 140];
cfg.artfctdef.zvalue.bpfiltord    = 9;
cfg.artfctdef.zvalue.bpfilttype   = 'but';
cfg.artfctdef.zvalue.hilbert      = 'yes';
cfg.artfctdef.zvalue.boxcar       = 0.2;

% make the process interactive
cfg.artfctdef.zvalue.interactive = 'yes';

[cfg, artifact_muscle] = ft_artifact_zvalue(cfg);
```



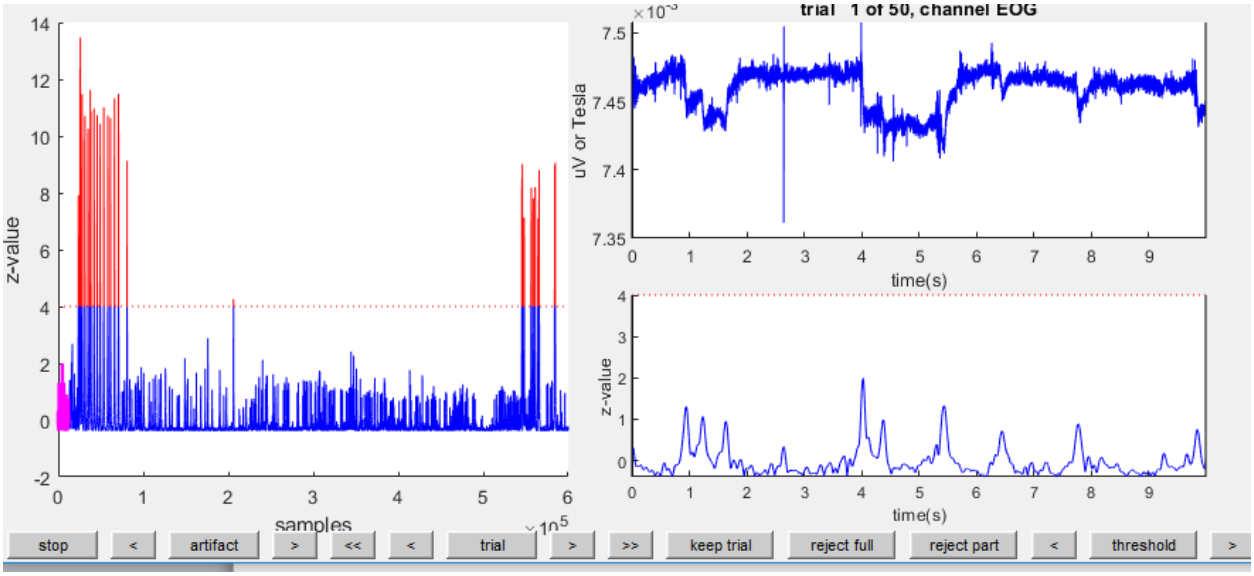
A typical muscle artifact can be observed on channel MRT12, trial 32.



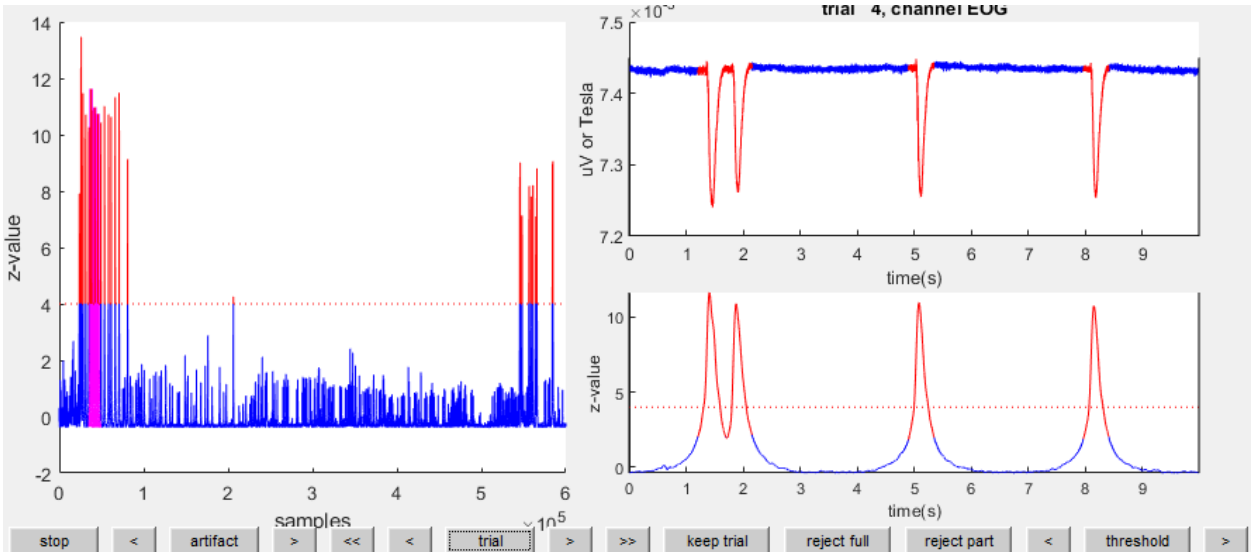
Detection of EOG artifacts

The same way as **ft_artifact_zvalue** is used to detect jump artifacts,it can be used to detect eye blinks artifacts (EOG). Note that only the EOG is scanned in the eye artifacts case, which will take less time than scanning all MEG channels, which was needed for jump and muscle artifacts.

```
% EOG
 cfg            = [];
 cfg.trl        = trl;
 cfg.datafile   = 'ArtifactMEG.ds';
 cfg.headerfile = 'ArtifactMEG.ds';
 cfg.continuous = 'yes';

 % channel selection, cutoff and padding
 cfg.artfctdef.zvalue.channel     = 'EOG';
 cfg.artfctdef.zvalue.cutoff      = 4;
 cfg.artfctdef.zvalue.trlpadding  = 0;
 cfg.artfctdef.zvalue.artpadding  = 0.1;
 cfg.artfctdef.zvalue.fltpadding  = 0;

 % algorithmic parameters
 cfg.artfctdef.zvalue.bpfilter   = 'yes';
 cfg.artfctdef.zvalue.bpfilttype = 'but';
 cfg.artfctdef.zvalue.bpfreq     = [2 15];
 cfg.artfctdef.zvalue.bpfiltord  = 4;
 cfg.artfctdef.zvalue.hilbert    = 'yes';

 % feedback
 cfg.artfctdef.zvalue.interactive = 'yes';

 [cfg, artifact_EOG] = ft_artifact_zvalue(cfg);
```



Typical eyeblink artifacts can be observed in trial 4. Note that the trial considers artifacts to precede and last longer than the threshold crossing, by padding data with 0.1 seconds on the left and 0.1 sec on the right



We are applying band pass filter to reject the unwanted signal.

Use independent component analysis (ICA) to remove ECG artifacts

Description

this script demonstrates how you can use ICA for cleaning the ECG artifacts from your MEG data. It consists of four steps:

1. preparing MEG data for running an ICA
2. decomposition of the MEG data
3. identifying the components that reflect heart artifacts
4. removing those components and backprojecting the data

Example dataset

```
% ft_preprocessing of example dataset
cfg = [];
cfg.dataset = 'ArtifactRemoval.ds';
cfg.trialdef.eventtype = 'trial';
cfg = ft_definetrial(cfg);
```

we have to remove all the artifacts like ,muscle artifacts, eog artifacts and jumps in data.

```
cfg = ft_artifact_jump(cfg);
cfg = ft_rejectartifact(cfg);
cfg.trl([3 11 23],:) = []; % quick removal of trials with muscle artifacts, works only for this dataset!
```

```
detected 56 artifacts

the call to "ft_artifact_zvalue" took 66 seconds

detected  56 jump artifacts

rejected    6 trials completely

rejected    0 trials partially

filled parts of    0 trials with nans

filled parts of    0 trials with the specified value

resulting  38 trials
```

now process the data,eeg data is processed by meg ,ica

```
cfg.channel            = {'MEG', 'EEG058'}; % channel 'EEG058' contains the ECG
recording
cfg.continuous         = 'yes';
data = ft_preprocessing(cfg);

% split the ECG and MEG datasets, since ICA will be performed on MEG data but not on
ECG channel
% 1 - ECG dataset
cfg              = [];
cfg.channel      = {'EEG'};
ecg              = ft_selectdata(cfg, data);
ecg.label{:}     = 'ECG'; % for clarity and consistency rename the label of the ECG
channel
% 2 - MEG dataset
cfg              = [];
cfg.channel      = {'MEG'};
data              = ft_selectdata(cfg, data);
```

processing channel { 'MLC11' 'MLC12' 'MLC13' 'MLC14' 'MLC15' 'MLC16' 'MLC17' 'MLC21' 'MLC22' 'MLC23' 'MLC24' 'MLC25' 'MLC31' 'MLC32' 'MLC41' 'MLC42' 'MLC51' 'MLC52' 'MLC53' 'MLC54' 'MLC55' 'MLC61' 'MLC62' 'MLC63' 'MLF11' 'MLF12' 'MLF13' 'MLF14' 'MLF21' 'MLF22' 'MLF23' 'MLF24' 'MLF25' 'MLF31' 'MLF32' 'MLF33' 'MLF34' 'MLF35' 'MLF41' 'MLF42' 'MLF43' 'MLF44' 'MLF45' 'MLF46' 'MLF51' 'MLF52' 'MLF53' 'MLF54' 'MLF55' 'MLF56' 'MLF61' 'MLF62' 'MLF63' 'MLF64' 'MLF65' 'MLF66' 'MLF67' 'MLO11' 'MLO12' 'MLO13' 'MLO14' 'MLO21' 'MLO22' 'MLO23' 'MLO24' 'MLO31' 'MLO32' 'MLO33' 'MLO34' 'MLO41' 'MLO42' 'MLO43' 'MLO44' 'MLO51' 'MLO52' 'MLO53' 'MLP11' 'MLP12' 'MLP21' 'MLP22' 'MLP23' 'MLP31' 'MLP32' 'MLP33' 'MLP34' 'MLP35' 'MLP41' 'MLP42' 'MLP43' 'MLP44' 'MLP45' 'MLP51' 'MLP52' 'MLP53' 'MLP54' 'MLP55' 'MLP56' 'MLP57' 'MLT11' 'MLT12' 'MLT13' 'MLT14' 'MLT15' 'MLT16' 'MLT21' 'MLT22' 'MLT23' 'MLT24' 'MLT25' 'MLT26' 'MLT27' 'MLT31' 'MLT32' 'MLT33' 'MLT34' 'MLT35' 'MLT36' 'MLT41' 'MLT42' 'MLT43' 'MLT44' 'MLT45' 'MLT46' 'MLT47' 'MLT51' 'MLT52' 'MLT53' 'MLT54' 'MLT55' 'MLT56' 'MLT57' 'MRC11' 'MRC12' 'MRC13' 'MRC14' 'MRC15' 'MRC16' 'MRC17' 'MRC21' 'MRC22' 'MRC23' 'MRC24' 'MRC25' 'MRC31' 'MRC32' 'MRC41' 'MRC42' 'MRC51' 'MRC52' 'MRC53' 'MRC54' 'MRC55' 'MRC61' 'MRC62' 'MRC63' 'MRF11' 'MRF12' 'MRF13' 'MRF14' 'MRF21' 'MRF22' 'MRF23' 'MRF24' 'MRF25' 'MRF31' 'MRF32' 'MRF33' 'MRF34' 'MRF35' 'MRF41' 'MRF42' 'MRF43' 'MRF44' 'MRF45' 'MRF46' 'MRF51' 'MRF52' 'MRF53' 'MRF54' 'MRF55' 'MRF56' 'MRF61' 'MRF62' 'MRF63' 'MRF64' 'MRF65' 'MRF66' 'MRF67' 'MRO11' 'MRO12' 'MRO13' 'MRO14' 'MRO21' 'MRO22' 'MRO23' 'MRO24' 'MRO31' 'MRO32' 'MRO33' 'MRO34' 'MRO41' 'MRO42' 'MRO43' 'MRO44' 'MRO51' 'MRO52' 'MRO53' 'MRP11' 'MRP12' 'MRP21' 'MRP22' 'MRP23' 'MRP31' 'MRP32' 'MRP33' 'MRP34' 'MRP35' 'MRP41' 'MRP42' 'MRP43' 'MRP44' 'MRP45' 'MRP51' 'MRP52' 'MRP53' 'MRP54' 'MRP55' 'MRP56' 'MRP57' 'MRT11' 'MRT12' 'MRT13' 'MRT14' 'MRT15' 'MRT16' 'MRT21' 'MRT22' 'MRT23' 'MRT24' 'MRT25' 'MRT26' 'MRT27' 'MRT31' 'MRT32' 'MRT33' 'MRT34' 'MRT35' 'MRT36' 'MRT37' 'MRT41' 'MRT42' 'MRT43' 'MRT44' 'MRT45' 'MRT46' 'MRT47' 'MRT51' 'MRT52' 'MRT53' 'MRT54' 'MRT55' 'MRT56' 'MRT57' 'MZC01' 'MZC02' 'MZC03' 'MZC04' 'MZF01' 'MZF02' 'MZF03' 'MZO01' 'MZO02' 'MZO03' 'MZP01' 'EEG058' }

reading and preprocessing

reading and preprocessing trial 35 from 35

the call to "ft_preprocessing" took 13 seconds

the call to "ft_selectdata" took 3 seconds

the call to "ft_selectdata" took 2 second

Finally, you should downsample your data before continuing, otherwise ICA decomposition will take too long.

Downsampling (or subsampling) is the process of reducing the sampling rate of a signal. This is usually done to reduce the data rate or the size of the data.

```
data_orig = data; %save the original data for later use
cfg            = [];
cfg.resamplefs = 150;
cfg.detrend    = 'no';
data           = ft_resampledata(cfg, data);
```

the input is raw data with 274 channels and 35 trials

the call to "ft_selectdata" took 1 seconds

resampling data

resampling data in trial 35 from 35

original sampling rate = 1200 Hz

new sampling rate = 150 Hz

the call to "ft_resampledata" took 34 seconds

Code

Now we can identify the ECG artifacts wit MEG data

he coherence between the components and the ECG channel, and the spatial topographies, it is possible to determine which components are responsible for the ECG artifact in the MEG channels. Those components can then be removed from the original data.

```
% read the already preprocessed MEG data from a MATLAB file
```

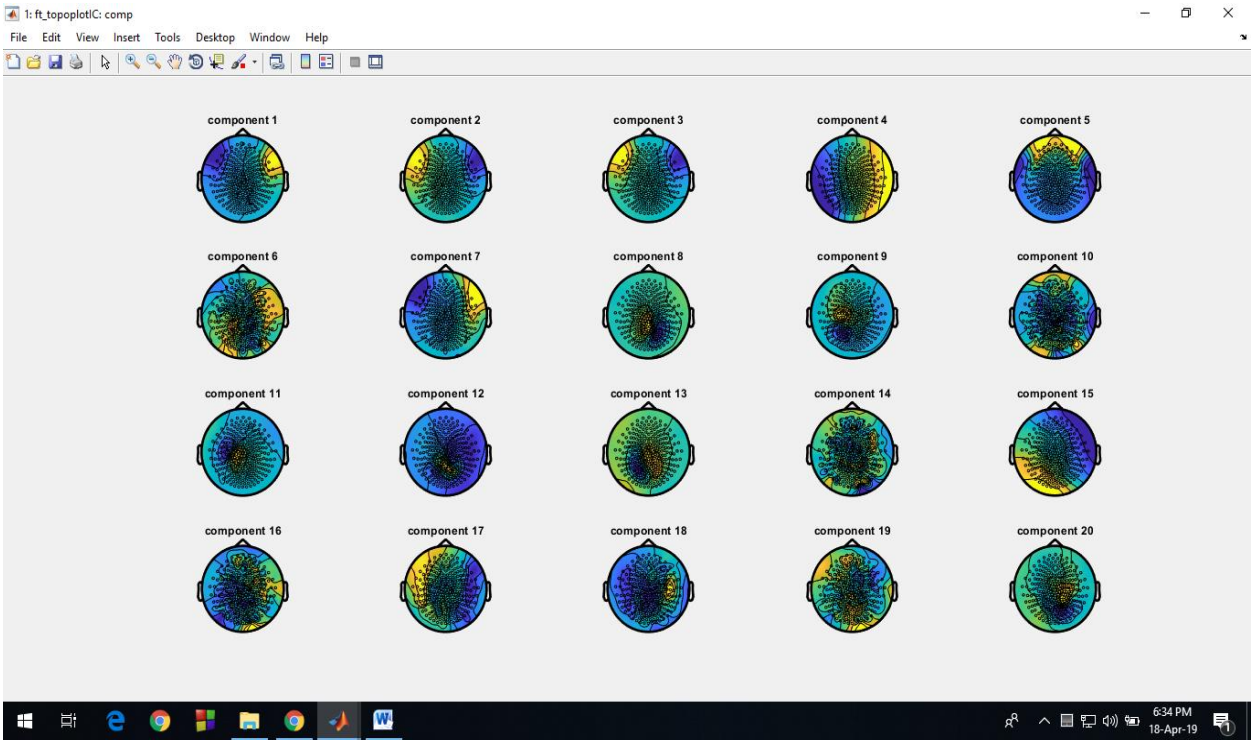
```
cfg             = [];
cfg.method      = 'runica';
comp            = ft_componentanalysis(cfg, data);
```

Sorting components in descending order of mean projected variance ...

also applying the unmixing matrix to the grad structure

the call to "ft_componentanalysis" took 357 seconds

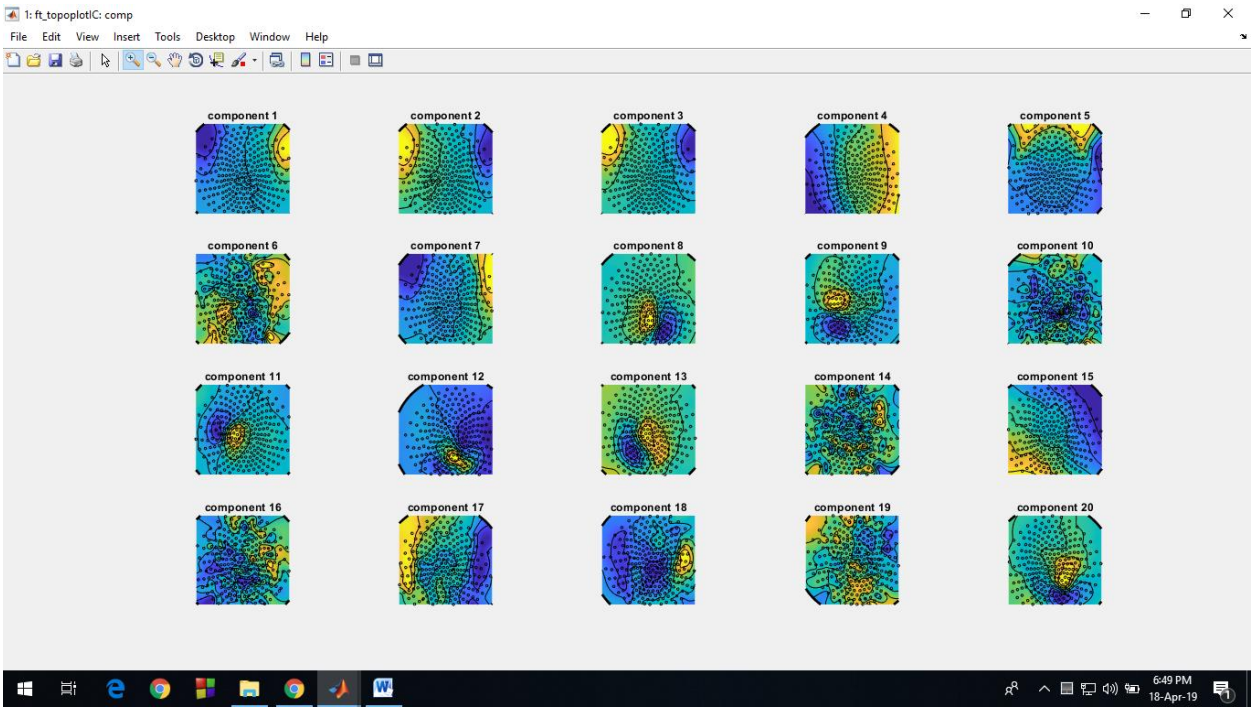
component analysis has been done no analysye the topography of the component



you will get the ECG components within the first 20 because the heartbeat is a very regular and very salient signal. You can almost always expect to get two ECG components, and they should look similar to each other, but slightly rotated.

components 4 and 17 show a regular signal, typical for the heartbeat.

I have zoom out see inside the topography .we can easly identify the similarity and nonsimilarity among all the topographic plot.




```
cfg           = [];
cfg.channel   = [2:5 15:18]; % components to be plotted
cfg.viewmode = 'component';
cfg.layout    = 'CTF275.lay'; % specify the layout file that should be used for
plotting
ft_databrowser(cfg, comp)
```

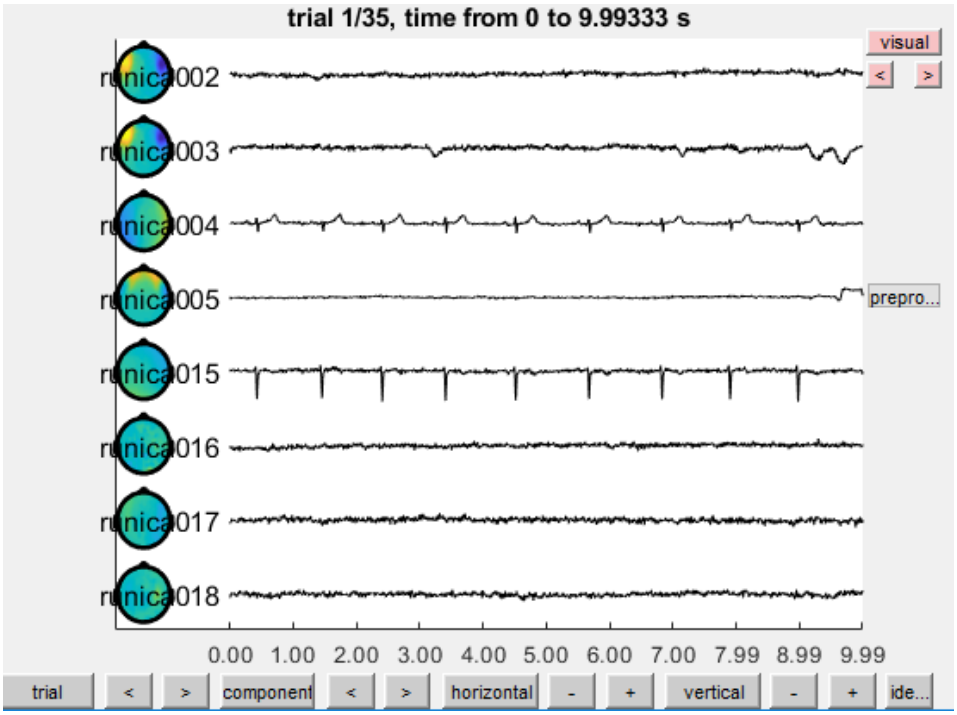
detected 0 visual artifacts

the call to "ft_prepare_layout" took 0 seconds

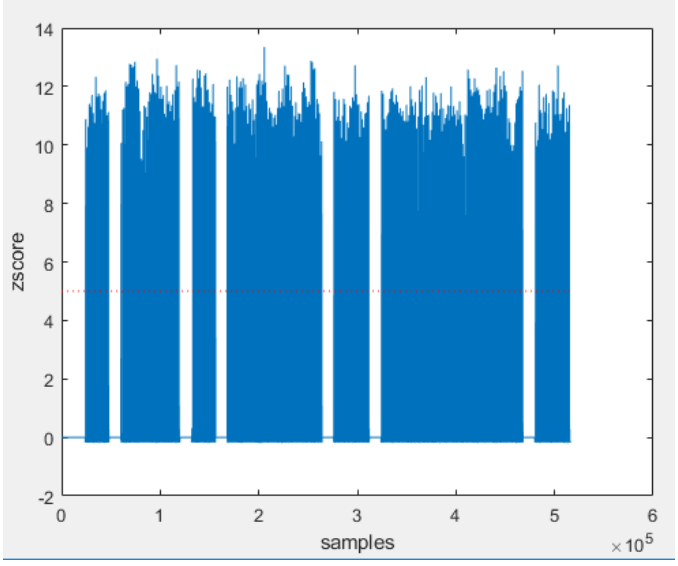
the call to "ft_databrowser" took 10 seconds

the below graph shows that

However, given that you measured the heartbeat on a separate channel, you can use this information to extract the two components of interest. Two possible ways of doing this is by using time lock data, and frequency data.



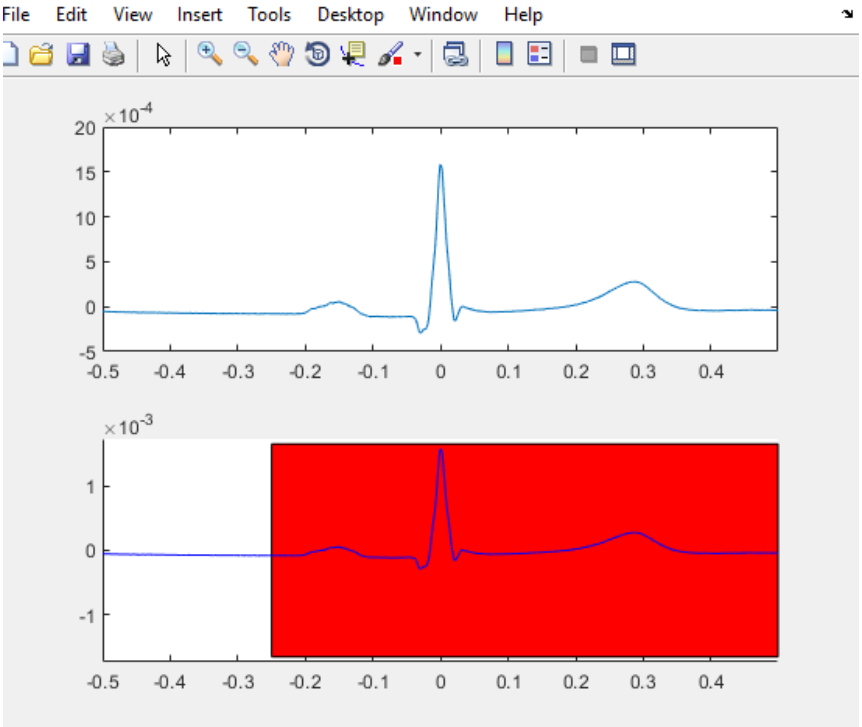

```
% go back to the raw data on disk and detect the peaks in the ECG channel, i.e. the
QRS-complex
cfg                         = [];
cfg.trl                     = data_orig.cfg.previous.trl;
cfg.dataset                 = data_orig.cfg.previous.dataset;
cfg.continuous              = 'yes';
cfg.artfctdef.ecg.pretim    = 0.25;
cfg.artfctdef.ecg.psttim    = 0.50-1/1200;
cfg.channel                 = {'ECG'};
cfg.artfctdef.ecg.inspect   = {'ECG'};
[cfg, artifact]             = ft_artifact_ecg(cfg, ecg);
```



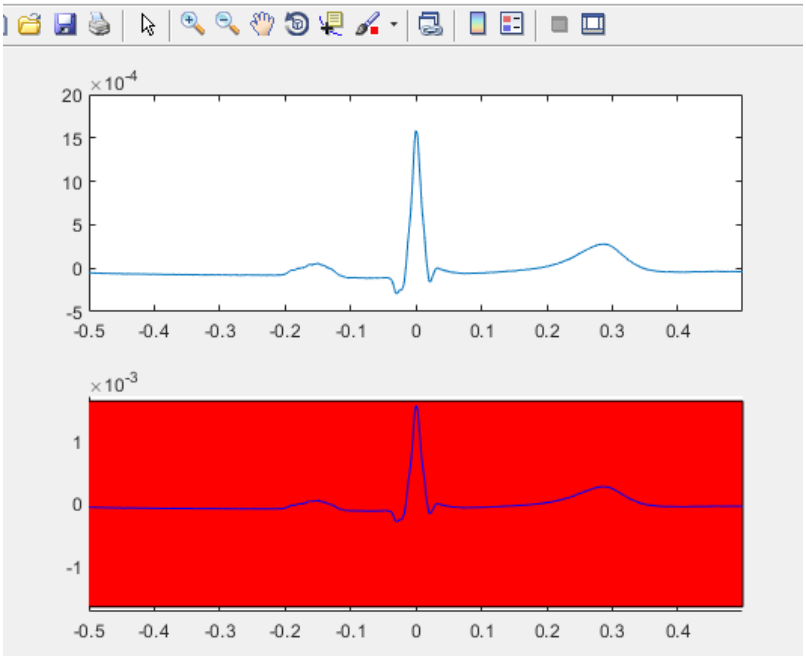
For keep the current value yes

current pre-peak interval = 0.250

ECG of the data



Keep the current value no



Keep changing the z value manually we can reject the artifact shown as dotted line above figure.

The visual graph shows that artifacts are very less because we have already removed the artifacts in visual and automatic artifacts data . graph does not have jump.

```
% preproces the data around the QRS-complex, i.e. read the segments of raw data
containing the ECG artifact
cfg            = [];
cfg.dataset    = data_orig.cfg.previous.dataset;
cfg.continuous = 'yes';
cfg.padding    = 10;
cfg.dftfilter  = 'yes';
cfg.demean     = 'yes';
cfg.trl        = [artifact zeros(size(artifact,1),1)];
cfg.channel    = {'MEG'};
data_ecg       = ft_preprocessing(cfg);
cfg.channel    = {'EEG058'};
ecg            = ft_preprocessing(cfg);
ecg.channel{:} = 'ECG'; % renaming is purely for clarity and consistency

% resample to speed up the decomposition and frequency analysis, especially usefull
for 1200Hz MEG data
cfg            = [];
cfg.resamplefs = 300;
cfg.detrend    = 'no';
data_ecg       = ft_resampledata(cfg, data_ecg);
ecg            = ft_resampledata(cfg, ecg);

% decompose the ECG-locked datasegments into components, using the previously found
(un)mixing matrix
cfg           = [];
cfg.unmixing  = comp.unmixing;
cfg.topolabel = comp.topolabel;
comp_ecg      = ft_componentanalysis(cfg, data_ecg);

% append the ecg channel to the data structure;
comp_ecg      = ft_appenddata([], ecg, comp_ecg);

% average the components timelocked to the QRS-complex
cfg           = [];
timelock      = ft_timelockanalysis(cfg, comp_ecg);
```

the call to "ft_preprocessing" took 787 seconds

processing channel { 'EEG058' }

reading and preprocessing

reading and preprocessing trial 333 from 333

the call to "ft_preprocessing" took 76 seconds

```
the input is raw data with 274 channels and 333 trials

the call to "ft_selectdata" took 20 seconds

resampling data

resampling data in trial 333 from 333


original sampling rate = 1200 Hz

new sampling rate = 300 Hz

the call to "ft_resampledata" took 1452 seconds

the input is raw data with 1 channels and 333 trials

original sampling rate = 1200 Hz

new sampling rate = 300 Hz

the call to "ft_resampledata" took 98 seconds

the input is raw data with 274 channels and 333 trials

the call to "ft_selectdata" took 2 seconds

baseline correcting data

scaling data with 1 over 0.000000

not concatenating data

starting decomposition using predetermined unmixing matrix

also applying the unmixing matrix to the grad structure

the call to "ft_componentanalysis" took 1031 seconds

the call to "ft_selectdata" took 6 seconds

the call to "ft_timelockanalysis" took 364 seconds
```

```
% look at the timelocked/averaged components and compare them with the ECG
figure
subplot(2,1,1); plot(timelock.time, timelock.avg(1,:))
subplot(2,1,2); plot(timelock.time, timelock.avg(2:end,:))
figure
subplot(2,1,1); plot(timelock.time, timelock.avg(1,:))
subplot(2,1,2); imagesc(timelock.avg(2:end,:));
```

we detect which components correlate more with the time course of the heartbeat.

Analyse the frequency versus time plot.

Figure1

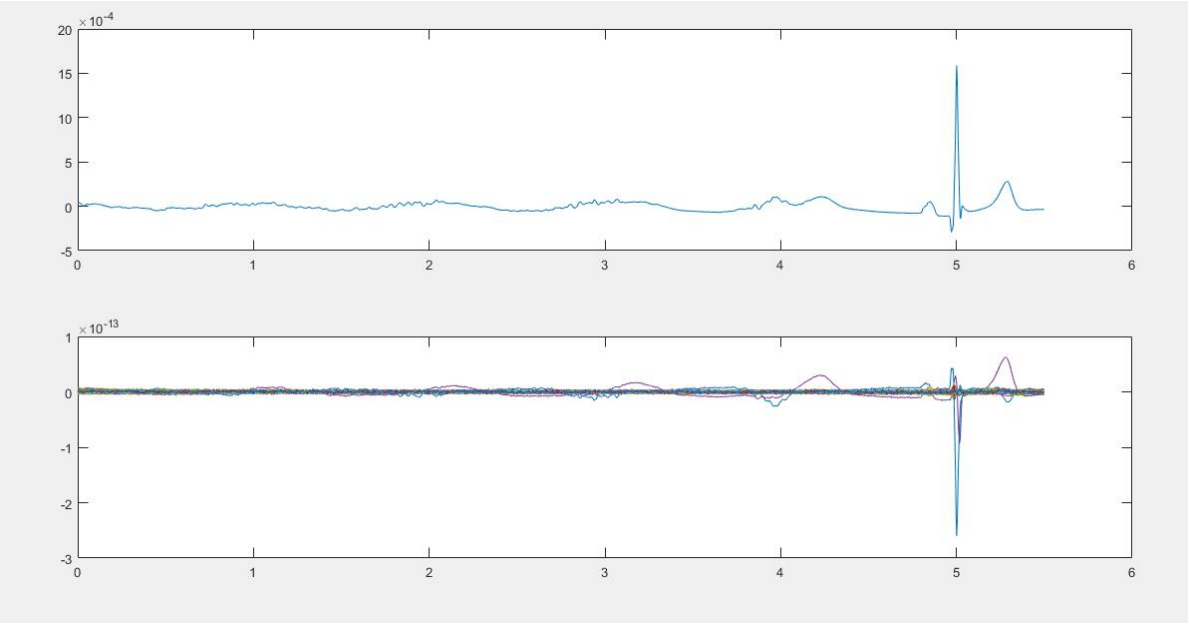
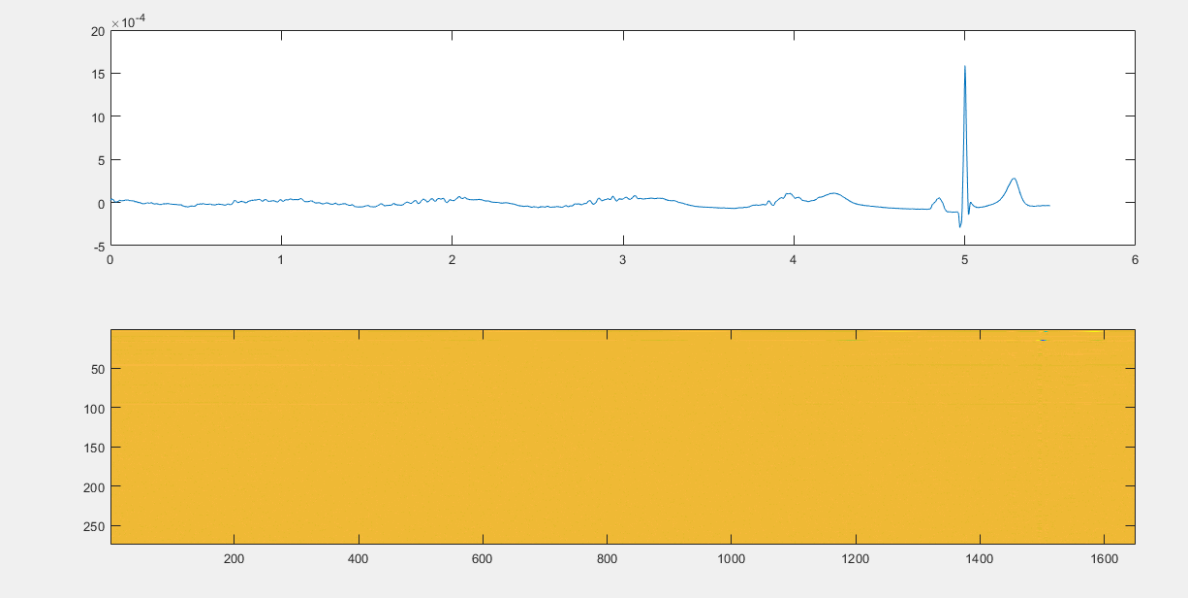
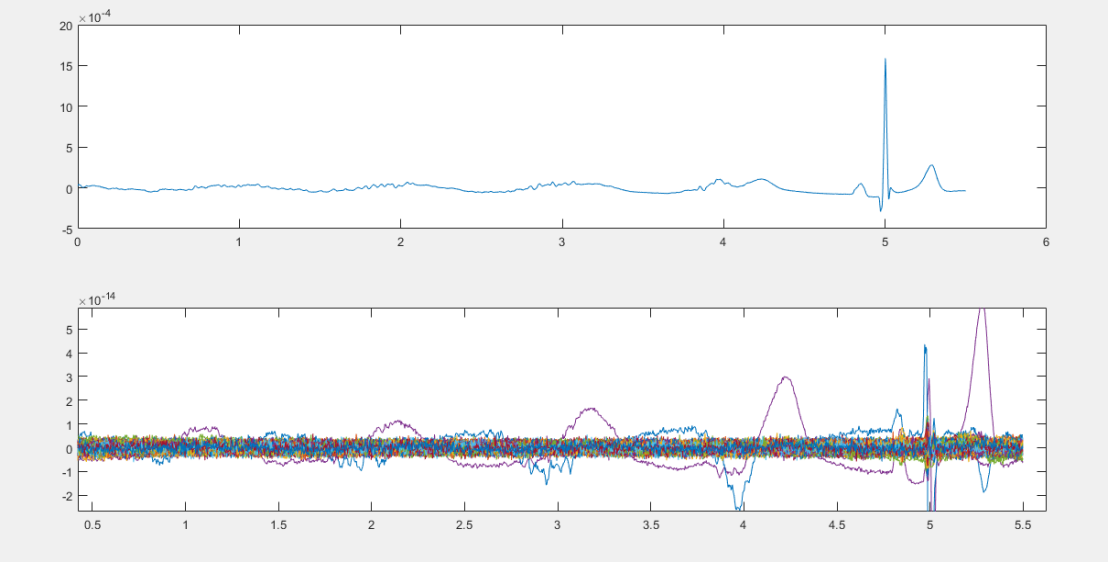
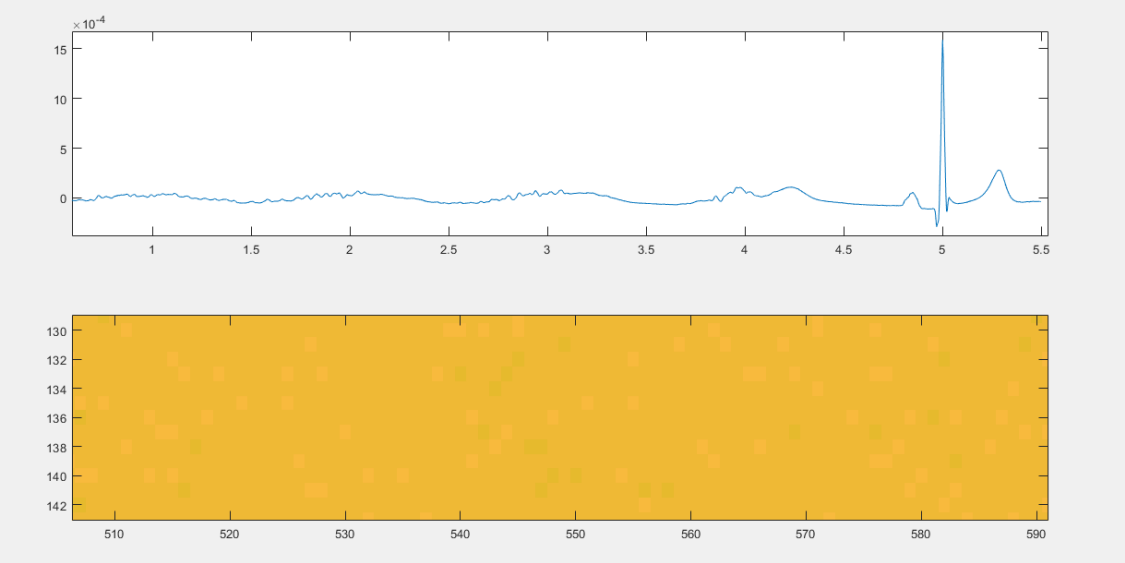


Figure2







By zooming into the second subplot of the second graph you can see which numbers these components have. In this case it is indeed components 4 and 17.

```
% compute a frequency decomposition of all components and the ECG
cfg            = [];
cfg.method     = 'mtmfft';
cfg.output     = 'fourier';
cfg.foilim     = [0 100];
cfg.taper      = 'hanning';
cfg.pad        = 'maxperlen';
freq           = ft_freqanalysis(cfg, comp_ecg);

% compute coherence between all components and the ECG
cfg            = [];
cfg.channelcmb = {'all' 'ECG'};
cfg.jackknife  = 'no';
```

```
cfg.method     = 'coh';
fdcomp         = ft_connectivityanalysis(cfg, freq);

% look at the coherence spectrum between all components and the ECG
figure;
subplot(2,1,1); plot(fdcomp.freq, abs(fdcomp.cohspctrm));
subplot(2,1,2); imagesc(abs(fdcomp.cohspctrm));
```

**** this figure I did not get I run the the above code the error was after execusion.

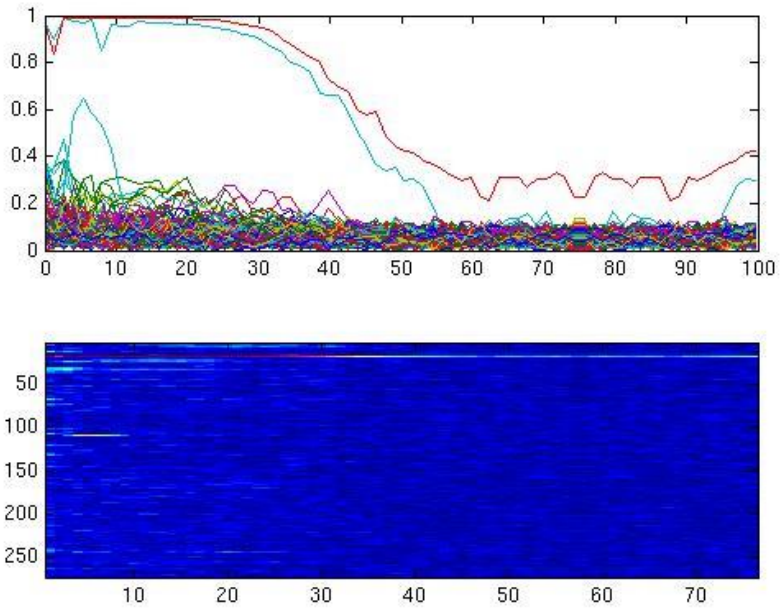
The error was Index in position 2 exceeds array bounds.

Error in ft_channelcombination (line 173)

collect = [datachannel(indx(:,1)) datachannel(indx(:,2))];

Error in ft_connectivityanalysis (line 174)

cfg.channelcmb = ft_channelcombination(cfg.channelcmb(:, 1:2), tmpchan, 1);



you now should select the components that explain the ECG artifact, and remove them from your data. The resulting dataset will contain the measured brain activity, with the variance attributable to the heartbeat removed out.

```
% decompose the original data as it was prior to downsampling to 150Hz
cfg           = [];
cfg.unmixing  = comp.unmixing;
cfg.topolabel = comp.topolabel;
comp_orig     = ft_componentanalysis(cfg, data_orig);

% the original data can now be reconstructed, excluding those components
cfg           = [];
cfg.component = [4 17];
data_clean    = ft_rejectcomponent(cfg, comp_orig,data_orig);
```

ft_connectivityanalysis

ft_connectivityanalysis computes various measures of connectivity between

MEG/EEG channels or between source-level signals.

Use as

stat = ft_connectivityanalysis(cfg, data)

stat = ft_connectivityanalysis(cfg, timelock)

stat = ft_connectivityanalysis(cfg, freq)

stat = ft_connectivityanalysis(cfg, source)

where the first input argument is a configuration structure (see below)

and the second argument is the output of FT_PREPROCESSING,

FT_TIMELOCKANLAYSIS, FT_FREQANALYSIS, FT_MVARANALYSIS or FT_SOURCEANALYSIS.

The different connectivity metrics are supported only for specific

datatypes (see below).

The configuration structure has to contain

cfg.method = string, can be

'amplcorr', amplitude correlation, support for freq and source data

'coh', coherence, support for freq, freqmvar and source data.
For partial coherence also specify cfg.partchannel, see below.
For imaginary part of coherency or coherency also specify
cfg.complex, see below.

'csd', cross-spectral density matrix, can also calculate partial
csds - if cfg.partchannel is specified, support for freq
and freqmvar data

'dtf', directed transfer function, support for freq and
freqmvar data

'granger', granger causality, support for freq and freqmvar data

'pdc', partial directed coherence, support for freq and
freqmvar data

'plv', phase-locking value, support for freq and freqmvar data

'powcorr', power correlation, support for freq and source data

'powcorr_ortho', power correlation with single trial
orthogonalisation, support for source data

'ppc' pairwise phase consistency

'psi', phaseslope index, support for freq and freqmvar data

'wpli', weighted phase lag index (signed one,
still have to take absolute value to get indication of
strength of interaction. Note: measure has positive

bias. Use wpli_debiased to avoid this.

'wpli_debiased' debiased weighted phase lag index
(estimates squared wpli)

'wppc' weighted pairwise phase consistency

'corr' Pearson correlation, support for timelock or raw data

Additional configuration options are

cfg.channel = Nx1 cell-array containing a list of channels which are used for the subsequent computations. This only has an effect when the input data is univariate. See FT_CHANNELSELECTION

cfg.channelcmb = Nx2 cell-array containing the channel combinations on which to compute the connectivity. This only has an effect when the input data is univariate. See FT_CHANNELCOMBINATION

cfg.trials = Nx1 vector specifying which trials to include for the computation. This only has an effect when the input data contains repetitions.

cfg.feedback = string, specifying the feedback presented to the user. Default is 'none'. See FT_PROGRESS

For specific methods the configuration can also contain

cfg.partchannel = cell-array containing a list of channels that need to be partialized out, support for method 'coh', 'csd', 'plv'

cfg.complex = 'abs' (default), 'angle', 'complex', 'imag', 'real', '-logabs', support for method 'coh', 'csd', 'plv'

cfg.removemean = 'yes' (default), or 'no', support for method

'powcorr' and 'amplcorr'.

cfg.bandwidth = scalar, (default = Rayleigh frequency), needed for

'psi', half-bandwidth of the integration across frequencies (in Hz)

To facilitate data-handling and distributed computing you can use

cfg.inputfile = ...

cfg.outputfile = ...

If you specify one of these (or both) the input data will be read from a *.mat

file on disk and/or the output data will be written to a *.mat file. These mat

files should contain only a single variable, corresponding with the

input/output structure.

See also ft_preprocessing, ft_timelockanalysis, ft_freqanalysis,

ft_mvaranalysis, ft_sourceanalysis, ft_networkanalysis.

For the implemented methods, see also FT_CONNECTIVITY_CORR,

FT_CONNECTIVITY_GRANGER, FT_CONNECTIVITY_PPC, FT_CONNECTIVITY_WPLI,

FT_CONNECTIVITY_PDC, FT_CONNECTIVITY_DTF, FT_CONNECTIVITY_PSI

Use independent component analysis (ICA) to remove EOG artifacts

Description-

Steps to be followed-

1. decomposition of the MEG data
2. identifying the components that reflect eye artifacts
3. removing those components and backprojecting the data

Example dataset

```
% preprocessing of example dataset
cfg = [];
cfg.dataset             = 'ArtifactMEG.ds';
cfg.trialdef.eventtype = 'trial';
cfg = ft_definetrial(cfg);

cfg.channel             = 'MEG';
cfg.continuous          = 'yes';
data = ft_preprocessing(cfg);

% downsample the data to speed up the next step
cfg = [];
cfg.resamplefs = 300;
cfg.detrend    = 'no';
data = ft_resampledata(cfg, data);
```

processing channel { 'MLC11' 'MLC12' 'MLC13' 'MLC14' 'MLC15' 'MLC21' 'MLC22' 'MLC23' 'MLC24' 'MLC31' 'MLC32' 'MLC33' 'MLC41' 'MLC42' 'MLC43' 'MLF11' 'MLF12' 'MLF21' 'MLF22' 'MLF23' 'MLF31' 'MLF32' 'MLF33' 'MLF34' 'MLF41' 'MLF42' 'MLF43' 'MLF44' 'MLF45' 'MLF51' 'MLF52' 'MLO11' 'MLO12' 'MLO21' 'MLO22' 'MLO31' 'MLO32' 'MLO33' 'MLO41' 'MLO42' 'MLO43' 'MLP11' 'MLP12' 'MLP13' 'MLP21' 'MLP22' 'MLP31' 'MLP32' 'MLP33' 'MLP34' 'MLT11' 'MLT12' 'MLT13' 'MLT14' 'MLT15' 'MLT16' 'MLT21' 'MLT22' 'MLT23' 'MLT24' 'MLT25' 'MLT26' 'MLT31' 'MLT32' 'MLT33' 'MLT34' 'MLT35' 'MLT41' 'MLT42' 'MLT43' 'MLT44' 'MRC13' 'MRC14' 'MRC15' 'MRC22' 'MRC23' 'MRC24' 'MRC31' 'MRC32' 'MRC33' 'MRC41' 'MRC42' 'MRC43' 'MRF11' 'MRF21' 'MRF22' 'MRF23' 'MRF31' 'MRF33' 'MRF41' 'MRF42' 'MRF44' 'MRF45' 'MRO11' 'MRO12' 'MRO21' 'MRO22' 'MRO31' 'MRO32' 'MRO33' 'MRO41' 'MRO42' 'MRO43' 'MRP11' 'MRP12' 'MRP13' 'MRP21' 'MRP22' 'MRP31' 'MRP32' 'MRP33' 'MRP34' 'MRT11' 'MRT12' 'MRT13' 'MRT14' 'MRT15' 'MRT16' 'MRT21' 'MRT23' 'MRT24' 'MRT25' 'MRT26' 'MRT32' 'MRT33' 'MRT34' 'MRT35' 'MRT42' 'MRT43' 'MRT44' 'MZC01' 'MZC02' 'MZF01' 'MZF02' 'MZF03' 'MZO01' 'MZO02' 'MZP01' 'MZP02' }

reading and preprocessing

reading and preprocessing trial 76 from 76

the call to "ft_preprocessing" took 68 seconds

the input is raw data with 139 channels and 76 trials

the call to "ft_selectdata" took 4 seconds

resampling data

resampling data in trial 76 from 76

ICA decomposition-

Now start decomposing the component independently

```
% perform the independent component analysis (i.e., decompose the data)
cfg        = [];
cfg.method = 'runica'; % this is the default and uses the implementation from EEGLAB

comp = ft_componentanalysis(cfg, data);
```

it takes long time to load the data and decomposition.

the call to "ft_componentanalysis" took 619 seconds

it contains the spatial mixing matrix.

In principle you can continue analyzing the data on the component level by doing.

```
cfg = [];
  cfg = ...
  freq = ft_freqanalysis(cfg, comp);
```

or

```
  cfg = [];
  cfg = ...
  timelock = ft_timelockanalysis(cfg, comp);
```

In ft_timelockanalysis at line 204

the call to "ft_timelockanalysis" took 35 seconds

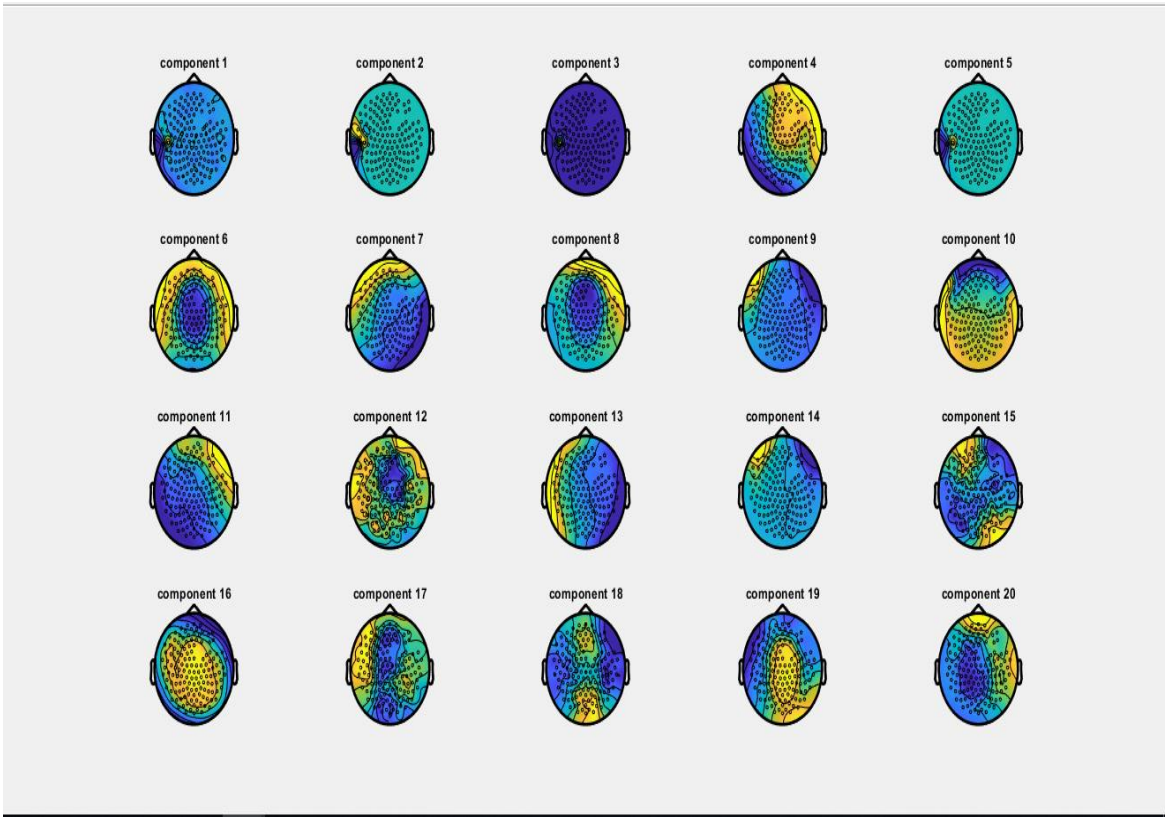
only remove the components that represent the artifacts.

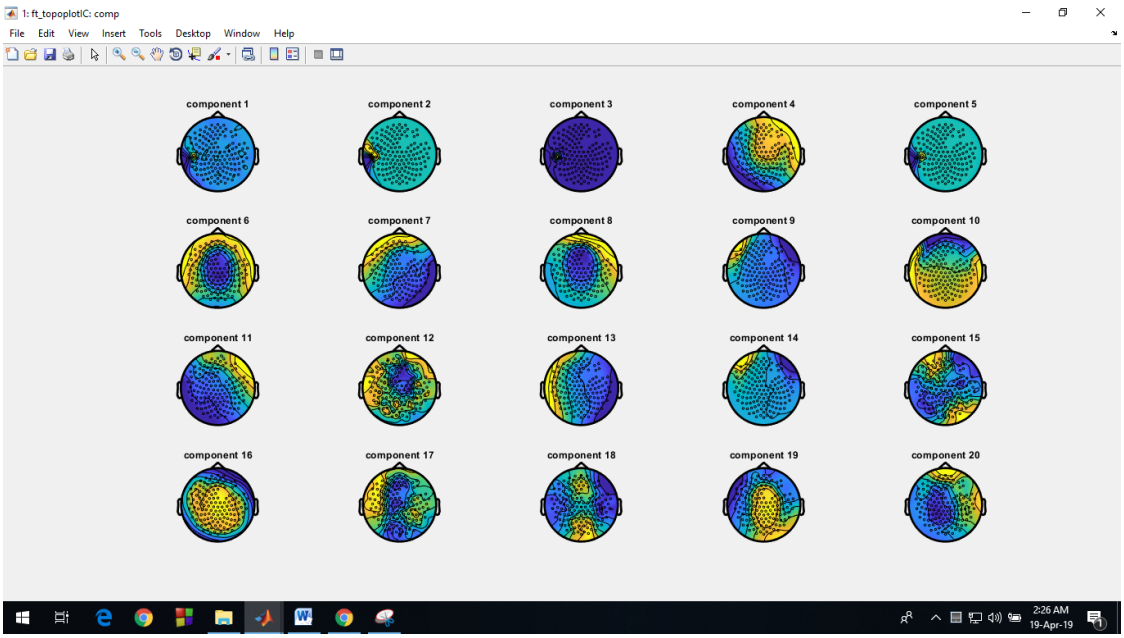
Identify the artifacts

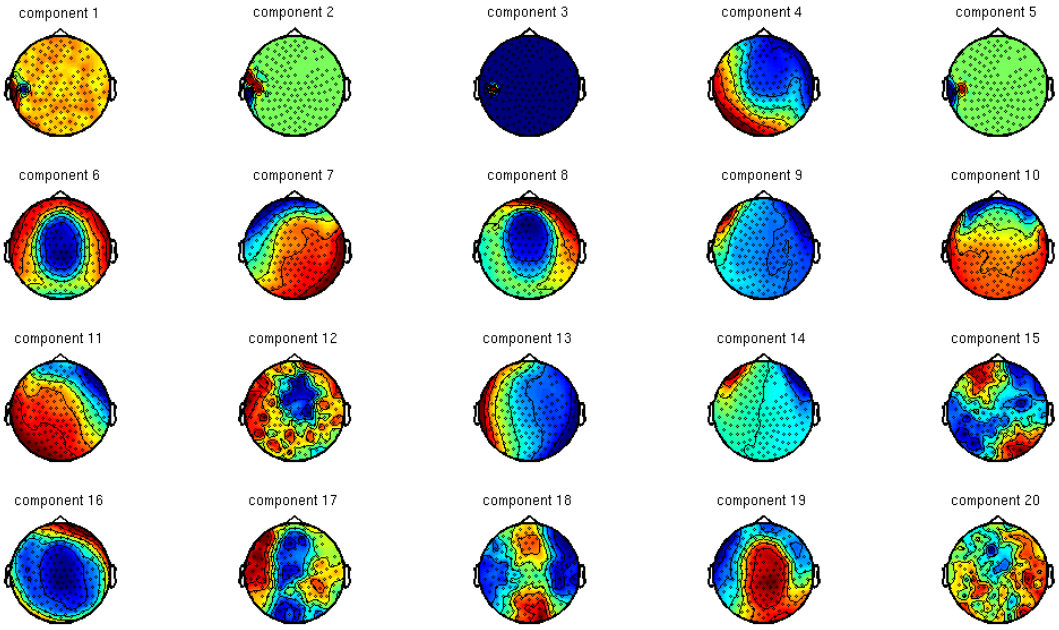
```
% plot the components for visual inspection
figure
cfg = [];
cfg.component = 1:20;       % specify the component(s) that should be plotted
cfg.layout    = 'CTF151.lay'; % specify the layout file that should be used for
plotting
cfg.comment   = 'no';
ft_topoplotIC(cfg, comp)
```

try to visualize the EOG artifact in decomposition

do not apply to another run of the ICA decomposition on the same data







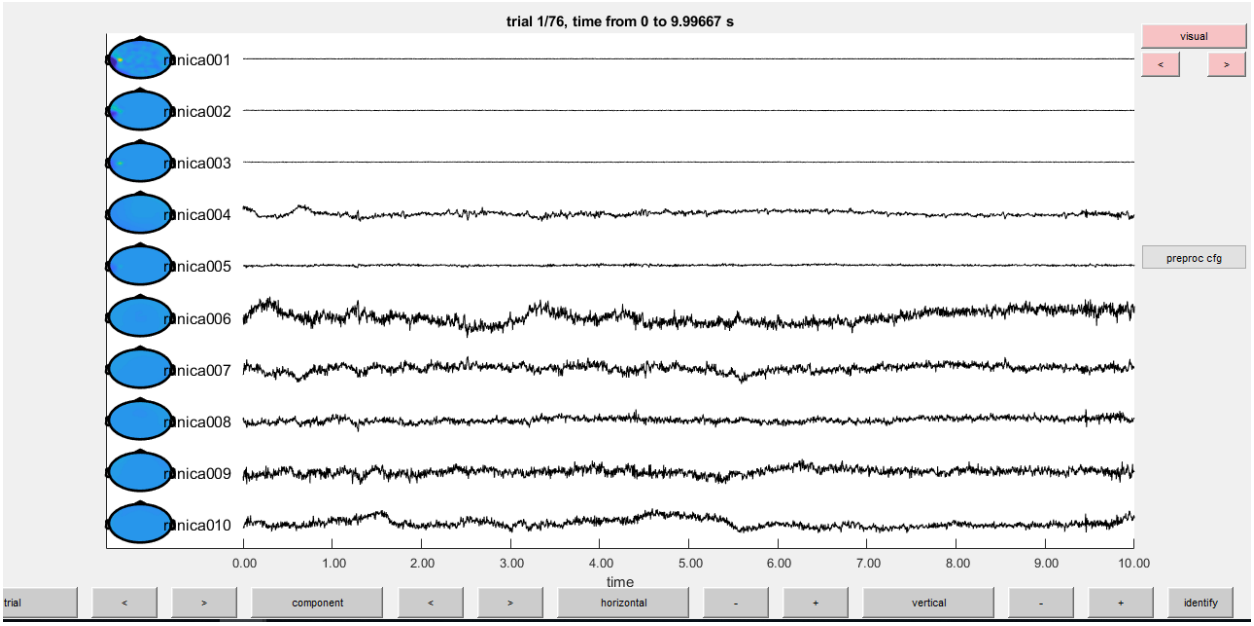
The spatial topography of the components aids in interpreting whether a component represents activity from the cortex, or non-cortical physiological activity (muscle, eyes, heart) or even non-physiological activity (line noise and other environmental noise).

easily spot the components that represent the eye movements: 9, 14 and 10.

Besides the spatial topography you should inspect the time course of the components, which gives additional information on separating the cortical from the non-cortical contributions to the data

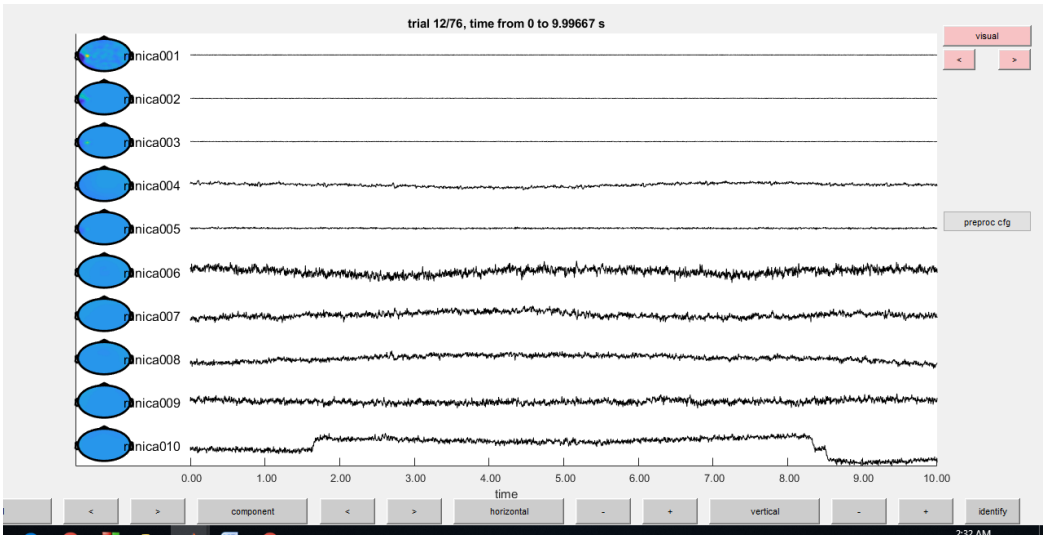
further time course component analysis-

```
cfg = [];
cfg.layout = 'CTF151.lay'; % specify the layout file that should be used for plotting
cfg.viewmode = 'component';
ft_databrowser(cfg, comp)
```



You can browse through the components and the trials. The EOG artifacts can be easily identified in the time course plots.

We can analyse the plots clearly and can inspect the artifacts.



Remove the artifacts

```
% remove the bad components and backproject the data
cfg = [];
cfg.component = [9 10 14 24]; % to be removed component(s)
data = ft_rejectcomponent(cfg, comp, data)
```

baseline correcting data

removing 4 components

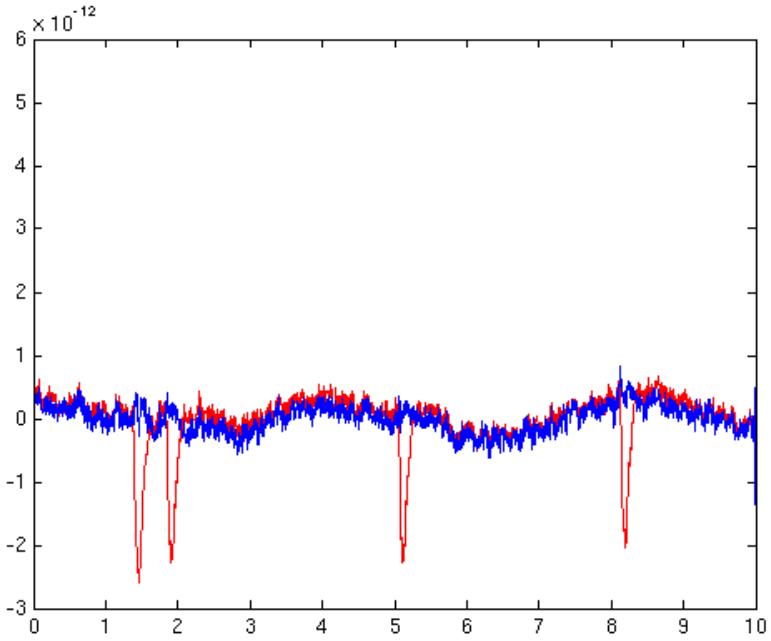
keeping 135 components

processing trials

processing trial 76 from 76

also applying the backprojection matrix to the grad structure

Compare the data before (red trace) and after (blue trace) the EOG removal - for example trial 4, channel MLF1


× 10^-12
6
5
4
3
2
1
0
-1
-2
-3
0
1
2
3
4
5
6
7
8
9
10