tutorial:scripting – FieldTrip

Creating a clean analysis scriptIntroduction
This tutorial is intended to provide some guidelines and suggestions how to set up a chain of analysis steps that makes most efficient use of your (and your computer's) time and is in accordance to the FieldTrip philosophy. Some MATLAB basics regarding the making of your own function will also be introduced. The idea of batching is introduced. Finally some practical tips regarding computer memory usage are given as well as an example.

The examples are about preprocessing of the data, but it does not provide detailed information about it. If you are interested in how to preprocess your data, you can check for example, this tutorial.

Background
The analysis of an experiment typically involves a lot of repetition as similar analysis steps are taken for every condition and for every subject. Also, the same steps are often repeated with only slightly different settings (e.g. filters, timings). Because of this we should program our own functions around the FieldTrip functions. FieldTrip functions are **not** intended to be just typed into MATLAB's command window. If you do, you are guaranteed to lose record of preceding steps, repeat yourself unnecessarily, or unknowingly change settings between subjects or conditions. Another '**no-no**' is the practice of collecting all your steps in one large m-file and copy-pasting parts in the command window. Besides becoming easily cluttered with previous tries, different filter settings, etc., it does not create a clear continuity between steps, and most importantly, does not permit batching. Batching is the ultimate aim of any analysis pipeline. It means that in the end most of your analysis steps can be repeated over all subjects and/or conditions with a single command.

Subject m-files
As stated before, by making our own function around FieldTrip functions we can in a later stage easily repeat them, e.g. over multiple subjects. However, every subject or condition will commonly have different filenames, different variables, different filter-settings, different trials that have to be rejected, etc. A good idea, therefore, is to first **write all your subject-specific details in a separate m-file**. You can choose to have one m-file per subject, or one in which you combine all subjects. In the current example we will use the first option:

```
% Subject01.m
```

```
% ensure that we don't mix up subjects
clear subjectdata

% define the filenames, parameters and other information that is subject specific
subjectdata.subjectdir        = 'Subject01';
subjectdata.datadir           = 'mtw05a_1200hz_20090819_04_600Hz.ds';
subjectdata.subjectnr         = '01';
subjectdata.MRI               = '01_mri';
subjectdata.badtrials         = [1 3]; % subject made a mistake on the first and third
```

```
trial

% more information can be added to this script when needed
...
```

Save this as *Subject01.m* in a personal folder that you will need to add to the MATLAB path. Using the command line you can now simply retrieve this personal data by calling

```
Subject01
```

or from any script by using

```
eval('Subject01')
```

. This will return the structure

```
subjectdata
```

containing all the fields we have specified. We can now use this structure as input for our own functions, giving us a flexible way of combining generic functions and subject-specific settings. In addition, you could use this file to add further comments such as

```
% subject made a mistake on the first trial
```

Making your own analysis functions

As already said, FieldTrip is most efficiently used by calling its functions within your own functions. To make a function in MATLAB write something in the style of:

```
function output = MyOwnFunction(input)

% MyOwnFunction takes the square root of the input
%
% the first few lines with comments are displayed as help

output = sqrt(input);
```

Make sure you save the filename identical as the function name, i.e. MyOwnFunction, and to save it in your personal folder dedicated to your own functions and scripts.

Do not save your own functions/scripts in the FieldTrip folder! This will not help you to organize your own functions - the number of functions will grow exponentially. Also, it makes it harder to update your FieldTrip folder.

Having saved your function in a folder of your MATLAB path you can, from within any script or from the command line, use your function. In our example

```
MyOwnFunction(4)
```

will give you the answer

```
2
```

To put the answer in a variable for storage or future use you need to call something like

```
output = MyOwnFunction(4)
```

This is the way most FieldTrip functions work: you provide the parameters together with data as the input and the function will return the results as the output.

It is often convenient to save intermediate results to disk. For instance you can type

```
save('firstoutput','output');
```

to save the output to *firstoutput.mat* in the directory you are in. Let's say you defined an output folder as in the first paragraph

```
subjectdata.subjectdir = 'Subject01';
```

you can program a generic solution to save all analysis steps of every subject in their own output folder:

```
save([subjectdata.subjectdir filesep 'firstoutput'],'output');
```

In this way all your functions (i.e. analysis steps) can read the output of the previous step as .mat files based upon their subject number.

We suggest that you store a single variable per file. This will in general make it possible to more easily only read what is necessary. Furthermore, if you give the files a clear and consistent name, you can easily delete the files (intermediate results) that are not needed anymore. Note that you can sort in the file manager on filename, as well as on creation date. The latter is convenient to quickly get an overview of the most recent files after you notice yet another bug in your analysis script :). For one subject a full analysis of the content of your data directory could then look something like this:

```
subject01.eeg
subject01_rawdata.mat
subject01_avg_cond1.mat
subject01_avg_cond2.mat
subject01_avg_cond3.mat
subject01_avg_cond4.mat
```

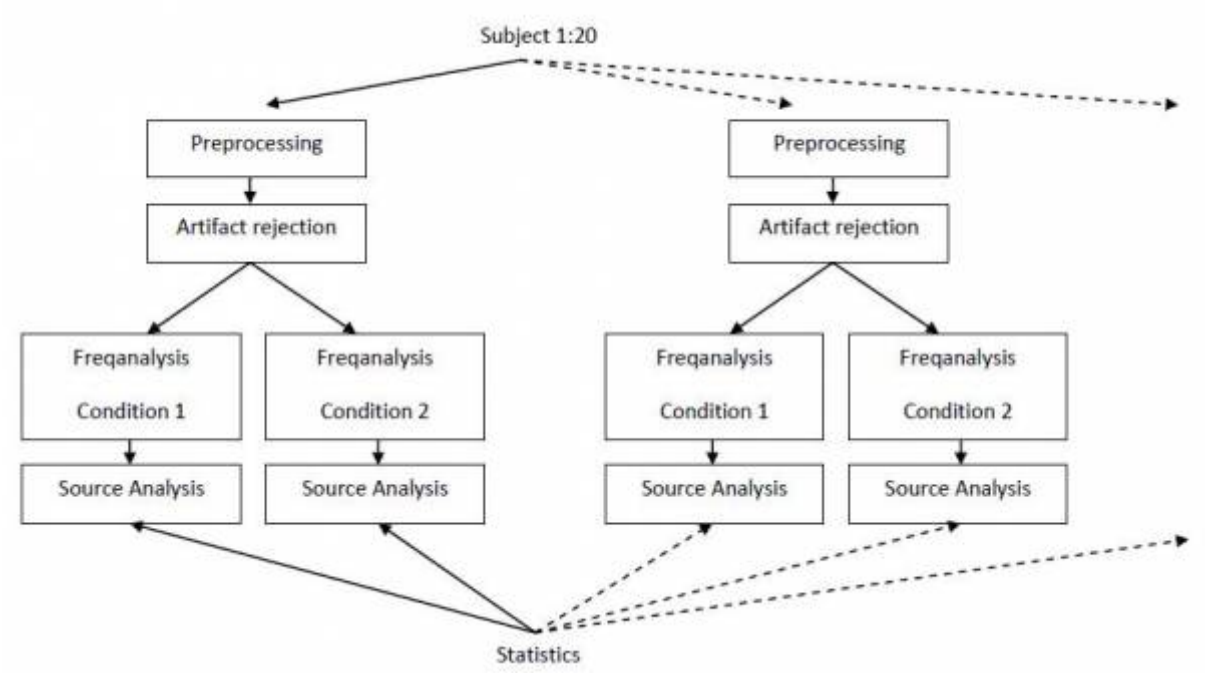
```
subject01_rawdata_filtered.mat
subject01_avg_cond1_filtered.mat
subject01_avg_cond2_filtered.mat
subject01_avg_cond3_filtered.mat
subject01_avg_cond4_filtered.mat
...
subject02.eeg
subject02_rawdata.mat
subject02_avg_cond1.mat
...
```

Along the way, you will most likely expand on the subject-specific information. For instance, in the first step you used ft_databrowser to select some unusual artifacts in one subject, which you could write (automatically) in your subject m-file:

```
subjectdata.visualartifacts = [
  160611,162906
  473717,492076
  604850,606076
  702196,703615
  736261,738205
  850361,852159
  887956,895200
  959974,972785
  1096344,1099772 ];
```

Batching

In the end we'll end up with a collection of several functions, either depending on the output of previous functions (e.g. preprocessing or artifact rejection) while others could in principle be called in parallel (e.g. averaging per condition or per subject). This could result in an analysis pipeline such as this (simplified) one:



This will allow us to automate most of the steps that do not require manual labor (in this example that would be the visual inspection of the data to reject artifacts). This is called *batching*. Large datasets will often require quite some processing time and it will therefore often be the case that a batch will be run overnight.

The worst that can happen is that the next morning you'll see some red lines in your MATLAB command window just because of a small mistake in one of the first subjects. Therefore, you might want to try using the

```
try-catch
```

option in MATLAB. Whenever something goes wrong between the

```
try
```

and

```
catch
```

it will jump to the catch after which it will just continue. E.g.:

```
for i = 1:number_of_subjects
  try
        my_preprocessing_function(i)
        % my_old_freqanalysis_function(i)
        my_freqanalysis_function(i)
        my_sourceanalysis_function(i)
  catch
        disp([‘Something was wrong with Subject’ int2str(i) ‘! Continuing with next in
line’]);
```

```
  end
end
```

Example batches

The following function will load the data as specified in Subject01.m, uses the databrowser for visual inspection of artifacts, rejects those trials containing artifacts and then saves the data in a separate folder as “01_preproc_dataM.mat”. You can simply call it by “do_preprocess_MM(‘Subject01’);”

```
function do_preproces_MM(Subjectm)

cfg = [];
if nargin == 0
        disp('Not enough input arguments');
        return;
end
eval(Subjectm);
outputdir = 'AnalysisM';

%%% define trials
cfg.dataset             = [subjectdata.subjectdir filesep subjectdata.datadir];
cfg.trialdef.eventtype  = 'frontpanel trigger';
cfg.trialdef.prestim    = 1.5;
cfg.trialdef.poststim   = 1.5;
%cfg.continuous         = 'no';
cfg.lpfilter            = 'no';
cfg.continuous          = 'yes';
cfg.trialfun            = 'motormirror_trialfun';   % located in \Scripts
cfg.channel             = 'MEG';
cfg.layout              = 'EEG1020.lay';
cfg                     = ft_definetrial(cfg);

%%% if there are visual artifacts already in subject m-file use those. They will show up
in databrowser
try
        cfg.artfctdef.eog.artifact = subjectdata.visualartifacts;
catch
end

%%% visual detection of jumps etc
cfg.continuous  = 'yes';
cfg.blocksize   = 20;
cfg.eventfile   = [];
cfg.viewmode    = 'butterfly';
cfg             = ft_databrowser(cfg);
```

```
%%% enter visually detected artifacts in subject m-file;
fid = fopen([subjectdata.mfiledir filesep Subjectm '.m'],'At');
fprintf(fid,'\n%s\n',['%%% Entered @ ' datestr(now)]);
fprintf(fid,'%s',['subjectdata.visualartifacts = [ ' ]);
if isempty(cfg.artfctdef.visual.artifact) == 0
        for i = 1 : size(cfg.artfctdef.visual.artifact,1)
                fprintf(fid,'%u%s%u%s',cfg.artfctdef.visual.artifact(i,1),'
',cfg.artfctdef.visual.artifact(i,2),';');
        end
end

fprintf(fid,'%s\n',[ ' ]; ']);
fclose all;

%%% reject artifacts
cfg.artfctdef.reject = 'complete';
cfg = ft_rejectartifact(cfg);

%%% make directory, if needed, to save all analysis data
if exist(outputdir) == 0
        mkdir(outputdir)
end

%%% Preprocess and SAVE
dataM = ft_preprocessing(cfg);
save([outputdir filesep subjectdata.subjectnr '_preproc_dataM'],'dataM','-V7.3')
clear all;
```

Summary and suggested further readings

This tutorial explained how to write your own functions and how to do batching in order to increase the efficiency of your analysis. If you are interested in further issues on memory usage and speed of the analysis, you can check this and this tutorials.

The following FAQs are related to issues with using MATLAB: