Practical: MVPA using the ADAM toolbox

By Johannes Fahrenfort

In this practical you will learn how to:

1. Work with the ADAM toolbox, learn how to plot and compare ERP and MVPA results from of EEG and MEG data
2. Perform a group-level decoding analysis on EEG data, both for raw EEG data and for time frequency data (TFR)
3. Learn how to plot and interpret temporal generalization matrices
4. Compute a single subject (first-level) decoding analysis on EEG data
5. Compute a single-subject (first-level) analysis of TFR data and play around with the scripts/data

Install the required toolboxes (including the ADAM toolbox)

To be able to work with the ADAM toolbox, you have to have a working version of EEGLAB, FieldTrip and ADAM on Matlab. Follow the instructions below:

STEP 1:

For this practical, we created an easy to download file which contains all required toolboxes, here is the link:
https://www.dropbox.com/s/h64mu9m6xv6vnwl/matlab_toolboxes.zip?dl=1

This file might be too large to fit on your account if you are not working on your own laptop. If you run into memory issues the teacher has a USB stick containing all the required files which you can use without having to download.

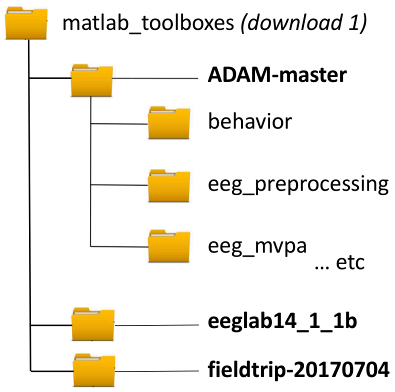
You can also download each of the toolboxes separately from the source links (below), but it is probably easier to just use the zip file from the linked zip file above as this contains everything you need in a single download:

- FieldTrip, we tested using: ftp://ftp.fieldtriptoolbox.org/pub/fieldtrip/fieldtrip-20170711.zip
- A recent release of EEGLAB, we tested using: ftp://sccn.ucsd.edu/pub/daily/eeglab14_1_1b.zip
- The latest version of the ADAM toolbox: https://github.com/fahrenfort/ADAM/archive/master.zip

If you already had a previous version of ADAM installed, please replace it with the latest version that you can download from the link above.

STEP 2:

Unzip the file(s) you just downloaded, which should create a folder called 'matlab_toolboxes' containing all three toolboxes, and should have the following organization:



You can put this folder anywhere you want (e.g. 'C:\matlab_toolboxes' on Windows PC, or '/Users/JJF/matlab_toolboxes' on a Mac), as long as you know where you put it.

STEP 3:

Next, we need to make sure that Matlab knows how to find the toolboxes. Go into the 'matlab_toolboxes' folder above, and then into the folder of the ADAM toolbox and finally the 'install' directory. Here you will find a `startup.m` file. Open this file by double clicking on it. You should see something like:

```
%------------------------ toolboxes ------------------------%
% path definitions
ft_path = 'C:\matlab_toolboxes\fieldtrip-20170704';
eeglab_path = 'C:\matlab_toolboxes\eeglab14_1_1b';
adam_path = 'C:\matlab_toolboxes\ADAM-master';
```

Now replace those paths to point to the three toolboxes from step 2.
If you already have EEGLAB (and/or FieldTrip) you can also point to their existing locations on your computer.

STEP 4

Once you have correctly specified the folder paths to all three toolboxes, run `startup.m` by clicking on the green 'Run' icon ▶ in your toolbar (at the top of your Matlab window, Editor tab). If all goes well, you should see:

```
FIELDTRIP IS ALIVE
EEGLAB IS ALIVE
ADAM IS ALIVE
```

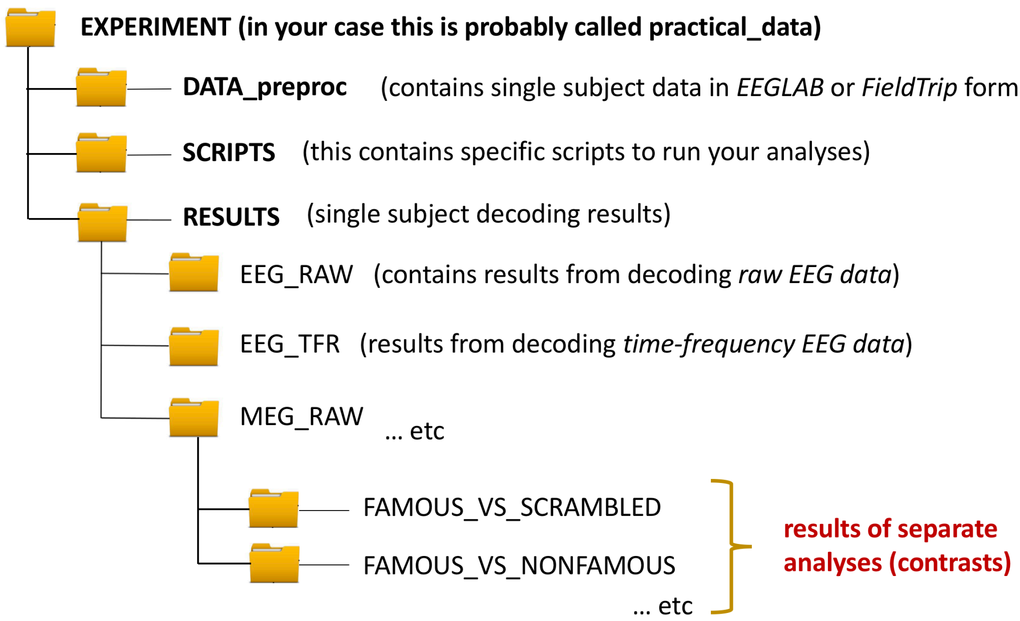
In the Command Window (along with some other messages), after which you should be able to run all aspects of the tutorial. For the most part, the tutorial relies only on the ADAM toolbox, only the first level (single subject) analyses require EEGLAB and/or FieldTrip.

NOTE:
After you ran `startup.m`, Matlab should now be able to find all the required functions by setting paths to the toolboxes. However, these path settings are lost when you close Matlab. For now, this is not a problem, because this is only one practical. However, if you would want Matlab to *always* be able to find the toolboxes, you should replace Matlab's default `startup.m` on your computer with the one you just modified. You can find the default `startup.m` file by typing `userpath` in the Matlab command window. This returns the location of the default `startup.m` file that Matlab runs when you start Matlab. When you replace this `startup.m` with the one above, it will be executed every time you run Matlab. If for whatever reason you run into path conflicts and want to reset the search paths to their defaults, you can type `restoredefaultpath` in your command window, but note that you will lose all existing paths in your settings.

PART I: Working with the ADAM toolbox

The next thing you have to download is the actual data and results that you will be working with during this practical. These you can find here:
https://www.dropbox.com/s/9pgaxlv91f75oo5/practical_data.zip?dl=1
Note that this is a rather large data file (~3.1 GB). It may not fit on your network drive, e.g. if you are not using your own laptop but one of the computers from the practical room. Try to save it locally, i.e. C:\ drive. If you cannot save to C:\ due to IT restrictions or lack of disk space, please ask the teacher for a USB stick to work from.

After successful unzip, the folder structure should be something as follows:


EXPERIMENT (in your case this is probably called practical_data)
DATA_preproc (contains single subject data in *EEGLAB* or *FieldTrip* format)
SCRIPTS (this contains specific scripts to run your analyses)
RESULTS (single subject decoding results)
EEG_RAW (contains results from decoding *raw EEG data*)
EEG_TFR (results from decoding *time-frequency EEG data*)
MEG_RAW
... etc
FAMOUS_VS_SCRAMBLED
FAMOUS_VS_NONFAMOUS
... etc
results of separate analyses (contrasts)

This is the way in which your data/scripts/results are best organized when using the ADAM toolbox. The **DATA_preproc** folder contains the pre-processed and epoched data (see EEGLAB or FieldTrip manual and/or previous lecture). To save space, this folder only contains the EEG and MEG data from the first subject, but the results do contain analyses from all subjects. Note that here, these are `.mat` files that contain the data in FieldTrip format, but you also put standard EEGLAB files here (`.set/.fdt` pairs), the ADAM toolbox is able to analyze those too, as long as they are epoched (segmented). You can look around in the folders to see what they contain.

The **RESULTS** folder contains the outcome of specific analyses that you run, for example when classifying from the EEG whether subjects are viewing faces or scrambled faces, or when classifying whether they were viewing famous faces or nonfamous faces. Because it is too time consuming to run these decoding analyses within the timeslot of the practical, we have already done this for you for all subjects. However, you will also learn how to do this for a single subject.

The **SCRIPTS** folder contains the custom scripts used to analyze and plot the data. These scripts all start with `run_` so that it is easy to recognize that these scripts were used to run analyses. This folder contains a couple of files. I use the the keyword 'RAW' to indicate that the file contains a script to perform a decoding analysis of raw EEG data (so without performing time-frequency analysis). I use TFR to indicate that it performs a decoding analysis on time-frequency data. I use the keyword 'firstlevels' this means that the script performs an analysis on all the single subjects. The scripts in the SCRIPTS folder were used to run the analyses that you see in the RESULTS folder. You do not have to inspect them now, but you can have a look at them in a later stage to find out how to script these analyses. This folder also contains a file called `run_practical.m`. In this practical, we are going to work from this file.

But before we start, some more quick background information about how the ADAM toolbox works. The toolbox has a couple of main functions that it uses. All user functions start with the prefix `adam_` so they are easily recognized. In part I of this practical, we are going to be doing a group analysis. The single subject (also called first-level) analyses have already been performed and are contained in the RESULTS folder. In this part of the practical you will mostly be using three functions:

adam_compute_group_ERP.m *(does group ERP analysis)*
adam_compute_group_MVPA.m *(group MVPA analysis)*
adam_plot_MVPA.m *(visualizes group results)*

The ADAM functions generally use a syntax like this:

result `= adam_function_name(`**cfg, filepath_or_data_variable**`);`

filepath_or_variable contains the path to the data or a variable name containing the data, while **cfg** (short for configuration) is a struct that contains the parameters that specify how to run the analysis. The idea to put these parameters in a struct was borrowed from the FieldTrip toolbox, but ADAM is *not* FieldTrip. You will see how to use the **cfg** struct once you start working through the `run_practical.m` script file.

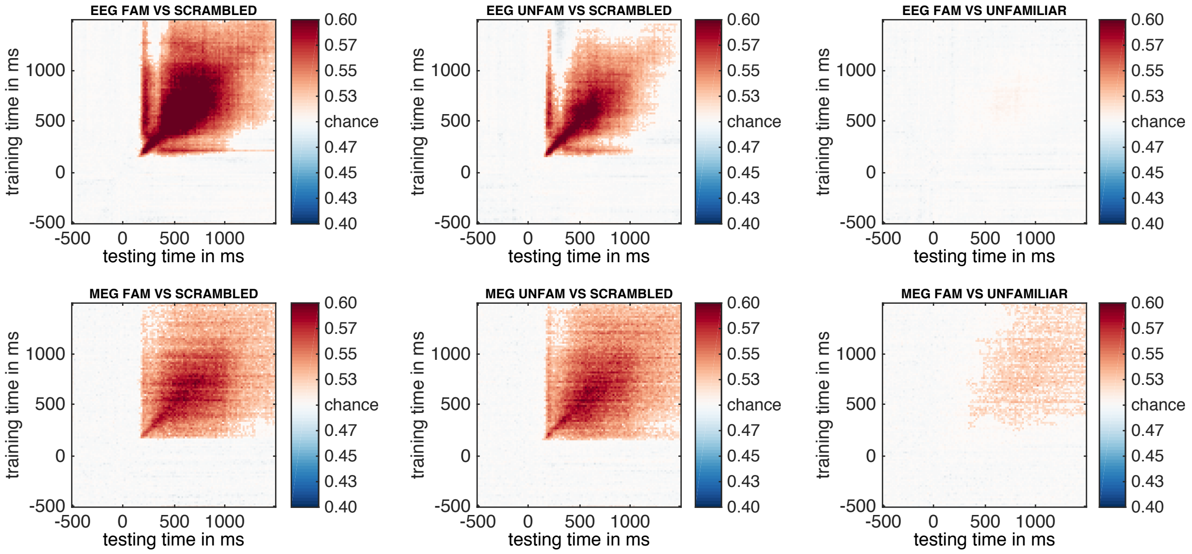
The `adam_compute_group` functions read the results from individual subject analyses that are contained in the RESULTS folder. They perform a group analysis on these data and return a **stats** struct that contains the outcome of the group analysis. This **stats** struct can also be an array of structs (multiple analyses), which can be input into `adam_plot_MVPA.m` for visualizing.

Open the `run_practical.m` file. The file contains 'cells' that highlighted in yellow when you put your cursor inside them. All code in a highlighted cell (yellow area) will execute when press CTRL+Enter (PC) or CMD(⌘)+Enter (Mac). Go through the `run_practical.m` script cell by cell. Read all text carefully, make sure you understand how the **cfg** structure is created, and what parameter settings are are contained in **cfg**.

Once you have finished PART I and II, return to this document

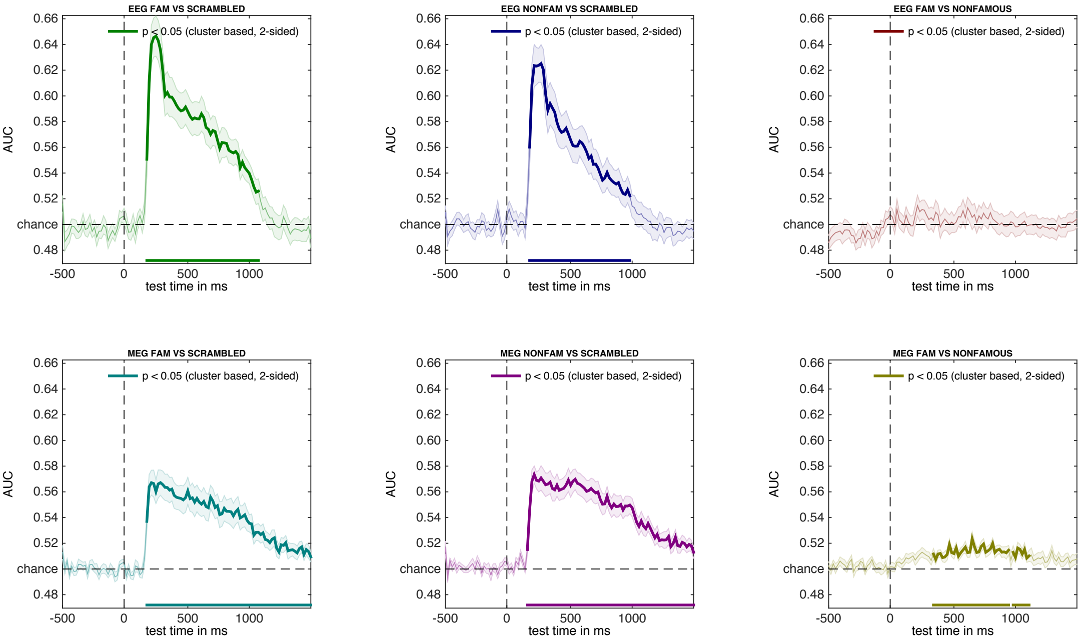
PART II: TEMPORAL GENERALIZATION, Summary

Only have a look at the results below *after* you have finished part II. Below is a summary of these results, pointing out the things you *should* have noticed when you went through part II of the practical.



The top row contains the EEG results, the bottom row are the MEG results. What you should have concluded from this graph is that EEG in this task produced stronger decoding accuracy (especially early on), but that MEG produced more stable representations (as you can also see in the FAMOUS vs NONFAMOUS comparison).

Next, you looked at what happened when you selected a limited window of data on which the classifier was trained, to inspect how well this window generalized to other time points. You can see what this looked like below:



As you can see, when training on the 200-300 ms interval and testing on all other time points, EEG shows an early peak that quickly levels off, whereas MEG shows a stable / less variable signal over time. This confirms what we already suspected based on the temporal generalization plots. Before you continue on to part III in Matlab, first read the short explanation about setting up first level analyses below.

PART III: first level analysis

For first level (single subject) analyses, there is only one core user function:

adam_MVPA_firstlevel.m
(does all first level single subject analyses)

Allows you to run a complete first level analysis (compute all single subject results) using one line of code, in combination with a **cfg** variable to specify the parameters of the analysis. First, you need to specify the subjects that you want to analyze, like below (here only the first 6 subjects for illustration purposes):

```
filenames = {
                'S01_ds000117_EEG' 'S01_ds000117_MEG_grad' ...
                'S02_ds000117_EEG' 'S02_ds000117_MEG_grad' ...
                'S03_ds000117_EEG' 'S03_ds000117_MEG_grad' ...
                'S04_ds000117_EEG' 'S04_ds000117_MEG_grad' ...
                'S05_ds000117_EEG' 'S05_ds000117_MEG_grad' ...
                'S06_ds000117_EEG' 'S06_ds000117_MEG_grad' ...
};
eeg_filenames = file_list_restrict(filenames,'EEG'); % restricts to only the EEG files
meg_filenames = file_list_restrict(filenames,'MEG'); % restricts to only the MEG files
```

The function `file_list_restrict` is a simple function that allows you to select files from your full file list based on a part of the file name. This can be useful in cases like this, where you have separate EEG and MEG files, or when you have files coming from different experimental sessions etc. You can now enter these file names into the cfg struct like this:

```
cfg.filenames = eeg_filenames;
```

As mentioned before, these files can either be in EEGLAB format (as long as the data is epoched) or FieldTrip format (you can use a struct with arbitrary name, either generic or timelock format that is saved inside a .mat file). You should not add an extension to the file name, ADAM will first look for a file that ends in .set, and if it canot find that it will look for a file that ends in .mat. The path to the files can be defined in the **cfg** using:

```
cfg.datadir = 'C:\practical_data\DATA_preproc';
```

Next, you also need to define the classes that you want to perform classification on (the conditions that you want to compare). Many experiments use factorial designs. For the current experiment, the factorial design looked like this:

	Famous	Nonfamous	Scrambled
First presentation	5	13	17
Immediate repeat	6	14	18
Delayed repeat	7	15	19

Factor
'stimulus type'

Factor
'stimulus repetition'

The numbers you see in the table are the event codes that were used in the experiment to denote the various conditions. It is good practice to start your script by defining all the event codes in the experiment, using code like below:

```
% event specifications: stimulus type
famous_faces = [5 6 7];           % specifies ALL famous faces
nonfamous_faces = [13 14 15];     % specifies ALL nonfamous faces
scrambled_faces = [17 18 19];     % specifies ALL scrambled faces

% event specification: repetition type
first_presentation = [5 13 17];  % specifies ALL initial presentations
immediate_repeat = [6 14 18];     % specifies ALL immediate repeats
delayed_repeat = [7 15 19];       % specifies ALL delayed repeats
```

Next, you can easily set up an analysis using these event code specifications, without making silly errors that mess up your analysis. For example, to classify all famous faces against all nonfamous faces, you would write:

```
cfg.class_spec{1} = cond_string(famous_faces);
cfg.class_spec{2} = cond_string(nonfamous_faces);
```

cond_string is an ADAM function that creates strings from event code specifications, because ADAM requires class specifications to be strings. Thus, the above class definition is effectively the same as:

```
cfg.class_spec{1} = '5,6,7';
cfg.class_spec{2} = '13,14,15';
```

Next, when you pass the above **cfg** to adam_MVPA_firstlevel, it will classify the activity across the 64 electrodes for each train-test sample in a trial as either coming from a famous face or from an nonfamous face, and compute average classification accuracy for each of these samples. You can run adam_MVPA_firstlevel using:

```
adam_MVPA_firstlevel(cfg);
```

ADAM forces you to think about whether your design is *balanced,* as it automatically enforces *event balancing* and *class balancing* during first level analysis. Note that ADAM currently enforces event balancing *within* stimulus classes by *undersampling* (throwing out trials). For example, if an analysis uses first presentations, immediate repeats and delayed repeats of famous faces to define the 'famous faces' class, and if there are 300 first presentations of famous faces, but only 50 immediate repeats and 50 delayed repeats, ADAM lowers the trial count of the first presentations to match with the others (so the 300 first presentations of famous faces would be lowered by randomly selecting 50 of those, to match with the immediate repeats and delayed repeats of famous faces). Between classes, ADAM applies *oversampling*. So if your analyses decodes famous faces versus scrambled faces, but there are only 100 famous faces, but 300 scrambled faces, ADAM synthetically generates another 200 famous faces in the training set so that the training set is fully balanced. See the slides about balancing of today's lecture if it is unclear why ADAM balances event codes *within* and *between* classes by design. To keep things simple, we did not analyze the immediate repeat and delayed repeat in the analyses we ran for you. We only used the 'first presentation' of each stimulus type for analysis (codes 5, 13 and 17). This can be done by specifying:

```
cfg.class_spec{1} = cond_string(famous_faces,first_presentation);
cfg.class_spec{2} = cond_string(nonfamous_faces,first_presentation);
```

cond_string will now take only vales that belong to both event code specifications, so in this case the above is equivalent to:

```
cfg.class_spec{1} = '5';
cfg.class_spec{2} = '13';
```

It is also possible to define different event codes for training and testing by separating them using a semicolon. For example, to train on first repeats, but test on second repeats use:

```
cfg.class_spec{1} = ...
cond_string(famous_faces,first_presentation,';',famous_faces,immediate_repeat);
cfg.class_spec{2} = ...
cond_string(nonfamous_faces,first_presentation,';',nonfamous_faces,immediate_repeat);
```

Note that you have to put quotation marks around the semi-colon because ADAM takes strings as event code definitions, and this is how the cond_string is able to tell apart the training and testing event codes. In fact, the above is equivalent to:

```
cfg.class_spec{1} = '6;7';
cfg.class_spec{2} = '14;15';
```

Do you understand what happens if you train on one type of stimulus and test on another?

OK, that is all you need to know about class specifications in ADAM. Now go through part III, IV and V of `run_practical` Good luck!