The Journal of Open Source Software

Reducing the efforts to create reproducible analysis code with FieldTrip

Mats W. J. van Es [1,2,3], **Eelke Spaak** [1], **Jan-Mathijs Schoffelen** [1], and **Robert Oostenveld** [1,4]

1 Donders Institute for Brain, Cognition and Behaviour, Radboud University Nijmegen, The Netherlands **2** Oxford Centre for Human Brain Activity, Department of Psychiatry, University of Oxford, United Kingdom **3** Wellcome Centre for Integrative Neuroimaging, University of Oxford, Oxford, United Kingdom **4** NatMEG, Karolinska Institutet, Stockholm, Sweden

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Claudia Solis-Lemus

Reviewers:

- @gflofst
- @ashahide

Submitted: 10 March 2023
Published: unpublished

Summary

FieldTrip `@Oostenveld2011` is a `@MATLAB2020` toolbox for the analysis of electroencephalography (EEG) and magnetoencephelography (MEG) data. Typically, a researcher will create an analysis pipeline by scripting a sequence of high level FieldTrip functions. Depending on researcher coding style, readability and reproducibility of the custom written analysis pipeline is variable. `reproducescript` is a new functionality in the toolbox that allows complete reproduction of MATLAB-based scripts with little extra efforts on behalf of the user. Starting from the researchers' idiosyncratic pipeline scripts, this new functionality allows researchers to automatically create and publish analysis pipeline scripts in a standardized format, along with all relevant intermediate data and final results. This approach may prove useful as a general framework for increasing scientific reproducibility, applicable well beyond the FieldTrip toolbox.

Statement of Need

Unsound scientific practices have led to a replication crisis in psychological science in recent years ([`@ OpenScienceCollaboration2015; @ Simmons2011`]), and it is unlikely that cognitive neuroscience is an exception ([`@Button2013; @Gilmore2017; @Szucs2017`]). One way to combat this crisis is through increasing methodological transparency ([`@Gilmore2017; @Gleeson2017; @Zwaan2017`]), but the increased sophistication of experimental designs and analysis methods results in data analysis getting so complex that the methods sections of manuscripts in most journals are too short to represent the analysis in sufficient detail. Therefore, researchers are increasingly encouraged to share their data and analysis pipelines along with their published results `@Gleeson2017`. However, analysis scripts are often written by researchers without formal training in computer science, resulting in the quality and readability of these analysis scripts to be highly dependent on individual coding expertise and style. Even though the computational outcomes and interpretation of the results can be correct, the inconsistent style and quality of analysis scripts make reviewing the details of the analysis difficult for other researchers, even those directly involved in the study. The quality of analysis scripts might thus compromise the reproducibility of obtained results. The purpose of `reproducescript` is to automatically create analysis pipeline scripts in a standardized format, along with all relevant intermediate data, that are executable, readable, and therefore fully reproducible, and that can directly be shared with peers.

State of the field

A number of strategies have been proposed to enhance the reproducibility of analysis pipelines and scientific results. One option to improve reproducibility and efficiency through reuse of code is through automation using pipeline systems (e.g. Taverna, Galaxy, LONI, PSOM, Nipype, Brainlife; [@Afgan2018; @Bellec2012; @Gorgolewski2011; @Oinn2004; @Pestilli2017; @Rex2003] or batch scripts (e.g. SPM's matlabbatch @Ashburner2020). Generally, these provide the researcher with tools to construct an analysis pipeline, manage the execution of the steps in the pipeline and, to a varying degree, handle data.

Some drawbacks of pipeline systems in general are the following: they require the researcher to learn how the pipeline software works on top of learning the analysis itself; the execution requires extra software to be installed, or requires moving the execution from a local computer to an online (cluster or cloud-based) system; they do not allow interactive analysis steps; and the flexibility of pipeline systems is limited. For example, MNE-Python, a widely used Python package for the analysis of electrophysiology data, has recently implemented the MNE-BIDS-Pipeline, which produces a standardized analysis pipeline. However, the analysis steps that are incorporated in this pipeline, as well as their order, is considerably limited compared to the full breadth of the MNE-Python package. Furthermore, many researchers use MATLAB for analysing their data, which is incompatible with this Python based software package.

Example

A detailed document with three examples that build up in complexity can be found on bioRxiv as Reducing the efforts to create reproducible analysis code with FieldTrip. These examples have also been incorporated on the FieldTrip website, see the example 1, example 2, and example 3 on the corresponding GitHub pages. Below we list the core usage of the functionality, as well as how it is implemented. Note that more detailed information on the structure of the FieldTrip toolbox can be found at Introduction to the FieldTrip toolbox and Toolbox architecture and organization of the source code.

The FieldTrip toolbox is not a program with a user interface where you can click around in, but rather a collection of functions. Each FieldTrip function implements a specific algorithm, for which specific parameters can be specified. These parameters on how the function behaves is passed as a configuration structure, for example:

```
cfg1                        = [];
cfg1.dataset                = 'Subject01.ds';
cfg1.trialdef.eventtype     = 'backpanel trigger';
cfg1.trialdef.eventvalue    = 3; % the value of the stimulus trigger for fully incongru
cfg1.trialdef.prestim       = 1;
cfg1.trialdef.poststim      = 2;
```

Users mainly use the high-level functions as the main building blocks in their analysis scripts. These functions are executed with the configuration structure (`cfg`) and in most cases with a data structure as inputs. For example:

```
cfg1               = ft_definetrial(cfg1);
dataPreprocessed   = ft_preprocessing(cfg1);

cfg2         = [];
dataTimelock = ft_timelockanalysis(cfg2, dataPreprocessed);
```

When using FieldTrip, the analysis protocol is the MATLAB script, in which you call the different FieldTrip functions. Such a script (or set of scripts) can be considered as an analysis protocol, since in them you are defining all the steps that you are taking during the analysis.

The high-level functions (which take a cfg argument) mainly do data bookkeeping and subsequently pass the data over to the algorithms in low-level functions. There are a number of features in the bookeeping that are always the same, hence these are shared over all high-level functions using the so-called `ft_preamble` and `ft_postamble` functions, which are called at the start and end of every high-level function, respectively.

`ft_preamble` ensures that the MATLAB path is set up correctly, that the notification system is initialized, that errors can be more easily debugged, that input data is read and analysis provenance tracked. `ft_postamble` takes care of e.g. debugging, updating of analysis provenance, and saving the output data.

The new functionality we propose, called *reproducescript*, is enabled by the user making a small addition to the configuration structure `cfg`. This results in a number of low level functions in the pre- and postamble scripts being called which automatically create analysis pipeline scripts in a standardized format, along with all relevant input, intermediate, and output data. Together, these represent a non-ambiguous, standardized, fully reproducible, and readable version of the original analysis pipeline. Moreover, this functionality is enabled without much effort from the researcher, namely by embedding the analysis pipeline in the wrapper below.

```
global ft_default
ft_default = [];

% enable reproducescript by specifying a directory
ft_default.reproducescript = 'reproduce/';

% the original analysis pipeline with calls (high level) FieldTrip functions should be w

% disable reproducescript
ft_default.reproducescript = [];
```

`ft_default` is the structure in which global configuration defaults are stored; it is used throughout all FieldTrip functions and global options at the start of the function are merged with the user-supplied options in the `cfg` structure specific to the function.

The directory containing the reproducible analysis pipeline is structured as below. The standardized script is in `script.m`, which is shown below the directory structure. All the data files are saved with a unique identifier to which is referred in `script.m`, and `hashes.mat` contains MD5 hashes for bookkeeping all input and output files. It furthermore allows any researcher to check the integrity of all the intermediate and final result files of the pipeline.

directory:

```
reproduce/
    script.m
    hashes.mat
    unique_identifier1_ft_preprocessing_input.mat
    unique_identifier1_ft_preprocessing_output.mat
    ...
```

`script.m`:

```
%%

cfg = [];
cfg.dataset = 'Subject01.ds';
cfg.trialdef.eventtype = 'backpanel trigger';
cfg.trialdef.eventvalue = 3;
cfg.trialdef.prestim = 1;
cfg.trialdef.poststim = 2;
```

```
cfg.showlogo = 'yes';
cfg.tracktimeinfo = 'yes';
cfg.trackmeminfo = 'yes';
cfg.datafile = 'Subject01.ds/Subject01.meg4';
cfg.headerfile = 'Subject01.ds/Subject01.res4';
cfg.dataformat = 'ctf_meg4';
cfg.headerformat = 'ctf_res4';
cfg.trialfun = 'ft_trialfun_general';
cfg.representation = [];
cfg.trl = 'reproduce/20221128T140217_ft_preprocessing_largecfginput_trl.mat';
cfg.outputfile = { 'reproduce/20221128T140217_ft_preprocessing_output_data.mat' };
ft_preprocessing(cfg);

%%

% a new input variable is entering the pipeline here: 20221128T140224_ft_timelockanalysi

cfg = [];
cfg.showlogo = 'yes';
cfg.tracktimeinfo = 'yes';
cfg.trackmeminfo = 'yes';
cfg.inputfile = { 'reproduce/20221128T140224_ft_timelockanalysis_input_data.mat' };
cfg.outputfile = { 'reproduce/20221128T140232_ft_timelockanalysis_output_timelock.mat' }
ft_timelockanalysis(cfg);
```

Because here we used *reproducescript* for a simple pipeline containing only three function calls, the standardized script does not look much different. For more complex analysis pipelines the differences with the original scripts tend to be larger. We refer the reader to the extended examples mentioned above for further details.

Acknowledgements

The authors would like to thank Lau Andersen for publishing his original data and analysis scripts in @Andersen2018 and his help in executing the pipeline.

- Author MVE is supported by The Netherlands Organisation for Scientific Research (NWO Vidi: 864.14.011), and Wellcome Trust (215573/Z/19/Z).
- Author ES is supported by The Netherlands Organisation for Scientific Research (NWO Veni: 016.Veni.198).
- Author JMS is supported by The Netherlands Organisation for Scientific Research (NWO Vidi: 864.14.011).
- Author RO is supported by

References