The Journal of Open Source Software

MultiSim: A Python Toolbox for Simulating MEG/EEG Datasets with Known "Ground Truth" Effects

Alex Lepauvre [1,2¶], **Qian Chu** [1,3,4,5], **Lucia Melloni** [1,6,7], **and Peter Zeidman** [8]

1 Neural Circuits, Consciousness and Cognition Research Group, Max Planck Institute for Empirical Aesthetics, Frankfurt am Main, Germany **2** Donders Institute for Brain, Cognition and Behaviour, Radboud University Nijmegen, Nijmegen, The Netherlands **3** Max Planck – University of Toronto Centre for Neural Science and Technology **4** Krembil Brain Institute, Toronto Western Hospital, University Health Network, Toronto, ON, Canada **5** Institute of Biomedical Engineering, University of Toronto, Toronto, ON, Canada **6** Department of Neurology, New York University Grossman School of Medicine, New York, NY, USA **7** Predictive Brain Department, Research Center One Health Ruhr, University Alliance Ruhr, Faculty of Psychology, Ruhr University Bochum, Bochum, Germany **8** Wellcome Centre for Human Neuroimaging, Institute of Neurology, University College London, London, UK ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

In MEG/EEG research, validating analysis pipelines is hampered by the lack of ground-truth neural signals in real data. SimMEG fills this gap by generating realistic, time-locked multivariate effects of known magnitude that you can inject into simulated sensor data. You can then run any pipeline—e.g. decoding, sensor-level statistics, or source estimation—against these datasets to benchmark sensitivity and specificity.

Key benefits include:

- Testing whether your pipeline reliably detects effects of a chosen size.
- Providing demonstrable, reproducible benchmarks for reviewers or collaborators.
- Offering a controlled teaching environment for newcomers.

Below, we describe the rationale (Statement of needs), and the data-generation method (Methods), a hands-on example (Results), and potential extensions (Discussion).

Statement of needs

MEG and EEG analysis pipelines—spanning from preprocessing and artifact rejection to sensor- or source-level statistics and multivariate decoding—struggle with three fundamental issues: the very high dimensionality of the data, low signal-to-noise ratios (SNR), and the fact that in real recordings we never know the "ground truth" about when or where genuine neural effects occur. As a result, it is difficult to quantify how sensitive a given pipeline really is, or how often it produces false positives.

SimMEG addresses this gap by enabling simulation of multivariate data with full control over the simulated effects. Users can define any experimental design matrix (conditions and contrasts), the parameters of the data set (number of trials, channels, participants and spatial covariance of the sensors), specify exactly when and how large each effect should be, to simulate entire subject cohorts. Users can then run your custom analysis pipeline on these synthetic datasets and directly measure its ability to detect the planted effects—and its proclivity for false alarms—under realistic noise and covariance conditions. Importantly, this toolbox can be used to determine the ideal number of trials per subject and the number of subjects, given the parameters of their recording system and the predetermined size of the effect under investigation.

Method

Generative model

For each subject s, we simulate epoched data $\mathbf{Y}_s \in \mathbb{R}^{N_{\text{samples}} \times n_{\text{feat}}}$ according to the general linear model:

$$\mathbf{Y}_s = \mathbf{X}\mathbf{B}_s + \mathbf{1}\boldsymbol{\beta}_{0,s}^{\top} + \boldsymbol{\varepsilon}_s,$$

where:

- $\mathbf{X}$ is the full design matrix with $n_{samples} * n_{trials}$ rows and $n_{samples} * n_{conditions}$ columns
- $\mathbf{B}$ is a matrix with $n_{samples} * n_{conditions}$ rows and $n_{features}$ columns
- $N_{\text{samples}} = n_{\text{epochs}} \times n_t$ is the total number of time samples across all trials, and $\mathbf{1}\boldsymbol{\beta}_{0,s}^{\top}$ is a subject-specific intercept term. The matrix $\mathbf{X}$ is the **full design matrix**, with one regressor for each combination of experimental condition and time point.

The noise term is drawn from a multivariate normal distribution:

$$\boldsymbol{\varepsilon}_s \sim \mathcal{N}(0, \sigma^2)$$

where $\sigma =$ `noise_std` and $=$ `spat_cov` denotes the spatial covariance of the sensors (default: identity).

Constructing the regression coefficients

Each experimental effect is defined by:

- an experimental condition (column index c),
- a temporal window $[t_{\text{on}}, t_{\text{off}}]$,
- and a desired multivariate effect size d (interpreted as a Cohen-style d').

We first create a rectangular temporal mask:

$$m_t = \begin{cases} 1 & \text{if } t_{\text{on}} \leq t \leq t_{\text{off}}, \\ 0 & \text{otherwise} \end{cases}$$

If a causal kernel h is provided, the temporal profile is convolved and rescaled to unit energy:

$$\widetilde{\mathbf{m}} = \frac{\mathbf{m} * h}{\|\mathbf{m} * h\|_2}$$

A uniform spatial pattern is used: $\mathbf{v} = \mathbf{1} \in \mathbb{R}^{n_{\text{feat}}}$. Its length under the inverse spatial covariance is:

$$L = \sqrt{\mathbf{v}^{\top -1}\mathbf{v}} = \sqrt{\text{tr}(^{-1})}$$

We then scale the amplitude for subject s as:

$$a_s = \mathcal{N}(d \cdot \sigma / L,\ \texttt{intersub\_noise\_std}^2)$$

Finally, the corresponding rows in the coefficient matrix $\mathbf{B}_s$ are populated as:

$$\beta_{c,t:s} = a_s \cdot \widetilde{m}_t \cdot \mathbf{v}^\top$$

All other entries of $\mathbf{B}_s$ remain zero.

From effect size to decoding accuracy

Because $\|\widetilde{\mathbf{m}}\|_2 = 1$, the effective Mahalanobis distance between classes at each time point is:

$$d'_t = \frac{a_s}{\sigma} \cdot L = d$$

This guarantees that the discriminability between class centroids at each time point matches the desired d'. Under standard assumptions of equal-covariance Gaussian classes, the theoretical decoding accuracy is:

$$P_{\text{correct}} = \Phi\left(\frac{d}{2}\right)$$

Thus, users can simulate data with known decoding difficulty: e.g. $d = 0.2, 0.5, 1.0$ yields ~60%, 69%, and 84% expected accuracy respectively, independent of the number or covariance of features.

Code Quality and Documentation

SimMEG is hosted on GitHub. Examples and API documentation are available on the platform XXX. We provide installation guides, algorithm introductions, and examples of using the package with Jupyter Notebook [REF]. The package is available on Linux, macOS and Windows for Python $>=$3.12 It can be installed with pip install simMEG. To ensure high code quality, all implementations adhere to the PEP8 code style [REF], enforced by ruff [REF], the code formatter black and the static analyzer prospector. The documentation is provided through docstrings using the NumPy conventions and build using Sphinx.

Acknowledgements

References