Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

neurodesign: Optimal Experimental Designs for Task fMRI

Joke Durnez
Stanford University and INRIA

Ross Blair
Stanford University

Russell A. Poldrack
Stanford University

Abstract

A recent stream of alarmist publications has questioned the validity of published neuroimaging findings. As a consequence, fMRI teams worldwide have been encouraged to increase their sample sizes to reach higher power and thus increase the positive predictive value of their findings. However, an often-overlooked factor influencing power is the experimental design: By choosing the appropriate experimental design, the statistical power of a study can be increased within subjects. By optimizing the order and timing of the stimuli, power can be gained at no extra cost.

To facilitate design optimization, we created a Python package and web-based tool called **neurodesign** to maximize the detection power or estimation efficiency within subjects, while controlling for psychological factors such as the predictability of the design. We implemented both a simulation-based optimization, as well as an optimization using the genetic algorithm, introduced by Wager and Nichols (2003) and further improved by Kao, Mandal, Lazar, and Stufken (2009), to optimize the experimental design. The toolbox **neurodesign** allows more complex experimental setups than existing toolboxes, while the graphical user interface (GUI) provides a more user-friendly experience. The toolbox is accessible online at `http://www.neuropowertools.org`.

Keywords: fMRI, experimental design, Python, GUI, statistical power.

1. Introduction

A recent stream of alarmist publications has questioned the validity of published neuroimaging findings (Eklund, Nichols, and Knutsson 2016; Ioannidis 2005; Open Science Collaboration 2015). At the core of the reproducibility crisis is the lack of power typically observed in neuroimaging (Button, Ioannidis, Mokrysz, Nosek, Flint, Robinson, and Munafò 2013), and more specifically, fMRI studies (Durnez, Moerkerke, and Nichols 2014). The signal measured in fMRI is known to be very noisy, while the hypothesized effects are small, such that a push for larger sample sizes promises a more powerful future for neuroimaging. Different power analysis strategies offer a way to optimize the sample size for a specific power level (Durnez *et al.* 2014; Mumford and Nichols 2008; Hayasaka, Peiffer, Hugenschmidt, and Laurienti 2007; Durnez, Degryse, Moerkerke, Seurinck, Sochat, Poldrack, and Nichols 2016). However, fMRI data are typically acquired and aggregated on two levels: within and between subjects. As such, increasing the power of an fMRI experiment can be achieved by increasing the number of subjects, but also via the within-subjects experimental design. This is especially true for smaller and more subtle effects, where the power curve is characterized by a slower increase, and thus the resulting power is more affected by the number of subjects and the number of time points. In addition to the duration of the experiment for each subject, the order and timing of different conditions within the experiment also influence the power of the resulting analyses.

The goal in task fMRI experiments is often one of two: detection or estimation. Detection refers to detecting the difference in brain activation between conditions or groups, while estimation relates to estimating the exact shape of the evoked fMRI response (called the hemodynamic response function, HRF). Ideally, the design of an fMRI experiment changes according to the specific research question asked. An optimal design with respect to these two distinct research questions are said to maximize the detection power or the estimation efficiency respectively. It is often argued that those two goals are opposite and an increase in detection power inevitably leads to a decrease of estimation efficiency. For example, when two trials of the same condition follow each other closely, the signal tends to accumulate linearly (Dale 1999), which makes it easier to detect. Therefore, the experiments often consist of blocks of the same condition. This type of design is called a blocked design. On the contrary, the accumulation (and saturation) of the measured signal conceals the shape of the HRF. To estimate the HRF, scientists often opt for an event-related design, where both the timing and order of conditions are randomized. However, Kao *et al.* (2009) show that the necessary trade-off between detection and estimation can be improved using certain optimization algorithms.

Another important aspect in an fMRI design is the psychological experience of the subject in the scanner. With a blocked design, the design becomes very predictable for subjects which can potentially bias the psychological function under investigation. To minimize predictability, Buračas and Boynton (2002) propose the use of m-sequences. However, the length of an m-sequence is restricted to $n = (Q_1)^l - 1$ with $Q + 1$ a prime, Q the total number of stimulus types, and l a positive nonzero integer. Recently Lin, Phao, and Kao (2007) proposed the use of a circulant (almost-)orthogonal array to expand the range of possible fMRI designs while ensuring complete independence between a trial and its successor. Very often, the best design is a combination of a maximal signal with low predictability. Therefore, Wager and Nichols (2003) suggest the use of a genetic algorithm to find an optimization between estimation efficiency, detection power and predictability. This algorithm optimizes a weighted average of different criteria, with the weights depending on the hypothesis and the expected outcome of the experiment. Subsequent work has further fine-tuned the algorithm and compared it with other approaches (Kao *et al.* 2009). However, in some cases the design requires more control than is offered in the genetic algorithm, in which case a simulation-based optimization is the only considerable option. In this paper, we present **neurodesign** for fMRI design optimization with different optimization algorithms, which is both available as a Python (van Rossum *et al.* 2011) module as well as a GUI web tool, available at `http://www.neuropowertools.org`. The paper is structured as follows: We start with a general description of the methodology in

Section 2. We show how designs can be compared and optimized using our Python module in Section 3. An overview of the GUI is given in Section 4. We compare our toolbox with other existing software in Section 5 and compare the designs from different optimizers in Section 6. Finally, we conclude and discuss the results in Sections 7 and 8.

2. Design optimization using the genetic algorithm

2.1. Statistical measures of design optimality

The signal measured using fMRI is the blood oxygen level dependent (BOLD) signal, which is a assumed to be related to the neural signal via convolution with a hemodynamic response function (HRF). We consider the general linear model (GLM) as the underlying model for the statistical objective as in Equation 1.

$$Y = X\beta + \epsilon, \epsilon \sim N(0, \sigma^2) \tag{1}$$

We denote Y as the measured signal. X represents the design matrix, β is the response amplitude for each column/condition in X and ϵ the error.

There are two types of design matrices: the convolved model and the finite impulse response (FIR) model where both are transformations of the matrix X_{base}, an $m \times t$ matrix, where m is the number of stimuli and t the number of measured time points. The values in X_{base} are 1 or 0: 1 when stimulus M is shown on time point T. The two transformations of X are shown in Figure 1. In the first model, the regressor X_{base} is convolved with the hemodynamic response function (Figure 1, panel 2) to represent the expected signal for a brain region related to the stimulus. The second model aims to estimate the exact shape of the HRF, by including regressors identical to the stimulus presentation, but each regressor with a certain temporal lag (Figure 1, panel 3).

Often, researchers are interested in specific hypotheses concerning particular combinations of parameters. The parameter of interest β can be estimated using the least squares estimator in Equation 2 and its variance in Equation 3.

$$\hat{\beta}_c = c^\top (X^\top X)^{-1} X^\top Y \tag{2}$$

$$\mathsf{VAR}(\hat{\beta}_c) = \sigma^2 c (X^\top X)^{-1} c^\top \tag{3}$$

with c the contrast vector of interest. To account for the specific character of fMRI data, we alter the model slightly. Because fMRI time series data exhibit substantial temporal autocorrelation, the errors in Equation 1 are not independent, $\epsilon \sim N(0, \sigma^2 V)$, where off-diagonal values represent the correlation between measurements at different time points. Furthermore a regressor, S, representing low-frequency noise components is added to the model (see Kao *et al.* 2009 for detailed derivations). The resulting variance of the estimator in defined in Equation 4.

$$\mathsf{VAR}(\hat{\beta}_c) = \sigma^2 c (X^\top W X)^{-1} c^\top \tag{4}$$

with $W = V - VS^\top(SVS^\top)^{-1}SV$. An optimal experimental design with respect to the estimator minimizes the variance of the estimator. We will therefore quantify the optimality of the design as the inverse of $c(X^\top WX)^{-1}c^\top$. Most often, an fMRI experiment has multiple

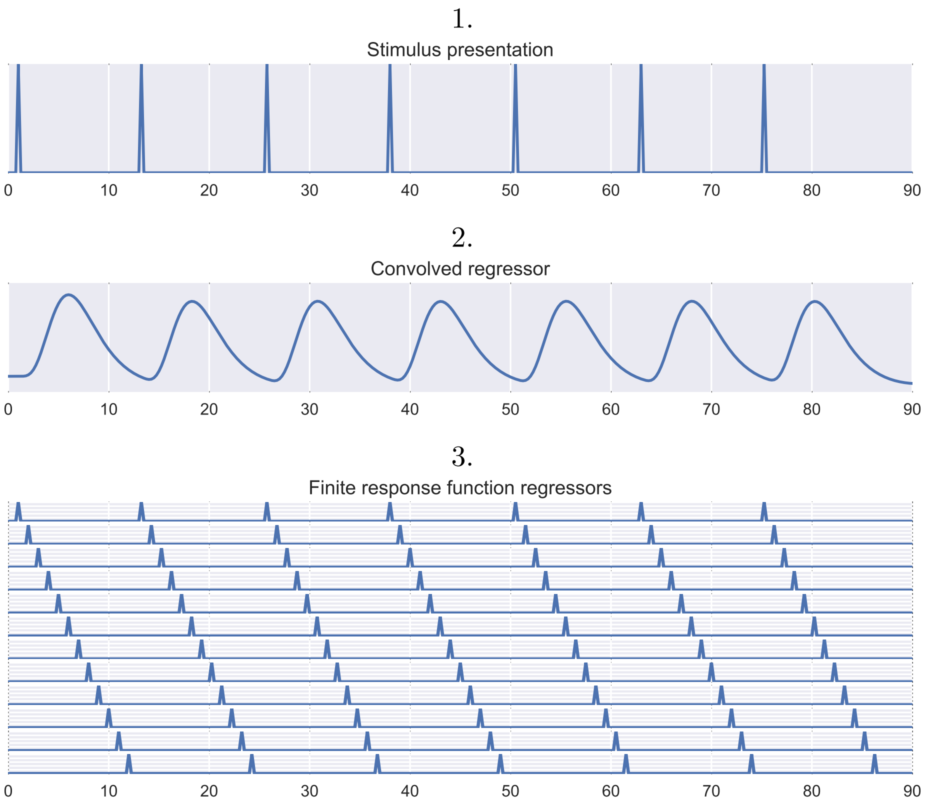


Figure 1: An experimental fMRI design with one stimulus type and two common models used in the GLM when modeling the resulting BOLD signal. The first panel shows the time series of the stimulus onsets. The second panel shows the stimulus onsets convolved with the double-gamma HRF, which can be interpreted as the expected BOLD signal if the measurement is related to the task. The parameter β in Equation 1 represents the amplitude of the signal related to the task. The third panel shows the FIR model, with each regressor a shifted version of the stimulus onsets. The parameter β represents the amplitude of the HRF at specific time points following stimulus onset. Units on the x-axis are seconds. Units on the y-axis are removed, as these are meaningless and often rescaled to have unit height.

contrasts of interest, thus $c(X^\top WX)^{-1}c^\top$ becomes a square matrix. With r_c the number of contrasts, there are two common ways to quantify the optimality of the design as defined in Equation 5.

$$\begin{aligned} F &= r_c/\mathrm{trace}(C(X^\top WX)^{-1}C^\top) \text{ for A optimality} \\ F &= \det(C(X^\top WX)^{-1}C^\top)^{-1/r_c} \text{ for D optimality} \end{aligned} \tag{5}$$

We denote F_e as the estimation efficiency if X is a FIR, and F_d as the detection power if X is a convolved design matrix.

2.2. Psychological measures of design optimality

Apart from the statistical concept of design efficiency, it is important to account for psychological factors that might render the experimental design invalid. The most important factor is predictability. For example in experiments addressing cognitive control, such as a stop-signal task, it is of the utmost importance that the trial type on any given trial cannot be easily predicted from the trial type on the previous trial, to avoid psychological confounding of the experiment. We quantify the optimality of the design in terms of confounding in Equation 6.

$$F_c = \sum_{r=1}^{R} \sum_{i=1}^{Q} \sum_{j=1}^{Q} n_{ij}^r - (n-r) P_i P_j, \tag{6}$$

where n_{ij}^r is the number of trials of type i at time point t preceding a trial of type j at time point $t+r$. The variable P_i is the proportion that this trial should occur in the experiment. If $F_c = 0$, there are no unforeseen contingencies between trial types. The final optimality criterion controls the desired trial type frequencies: $F_f = \sum_{i=1}^{Q} |n_i - nP_i|$, with n_i the number of trials of type i.

2.3. Multi-objective criterion

To ensure comparability across different optimality criteria, we first rescale the different optimality criteria to a scale of 0 to 1 as in Kao *et al.* (2009). To find the maximum F_d and F_e possible, we first run an optimization with weights 1 for respectively F_d and F_e and weights 0 for the other optimality criteria. In the multi-objective criterion, the F_d and F_e scores are divided by their respective maximum to ensure scores between 0 and 1. For F_f and F_c, the score for the worst possible design (a design with only the least probable stimulus) is taken as the maximum score. Second, whereas larger F_e and F_d represent better designs, the opposite is true for F_c and F_f. Therefore the scores for F_c and F_f are subtracted from 1. As such, the resulting optimality criterion is obtained in Equation 7.

$$F_i^* = \begin{cases} \frac{F_i}{\max(F_i)}, & i = d, e \\ 1 - \frac{F_i}{\max(F_i)}, & i = c, f \end{cases} \tag{7}$$

As no design can ensure optimality in all four optimality criteria, the goal of any design optimization depends on the researcher's goal of the experiment. Given prespecified weights w_i with $i = c, d, e, f, \sum_i w_i = 1, w_i \geq 0$, we define the weighted optimality criterion in Equation 8.

$$F^* = w_c F_c + w_d F_d + w_e F_e + w_f F_f \tag{8}$$

2.4. Optimization algorithms

Genetic algorithm

A genetic algorithm is a method for solving optimization problems inspired by natural selection in biological evolution. Contrary to classical optimization algorithms, a genetic algorithm generates a population of points at each iteration. A graphic representation of the genetic algorithm with an fMRI example is shown in Figure 2. The steps of the genetic algorithm are.

1. Create G initial random designs.
2. **Crossover.** Pair the best $G/2$ designs with each other.
3. **Mutation.** Randomly switch $q\%$ of all trials by random trial types.

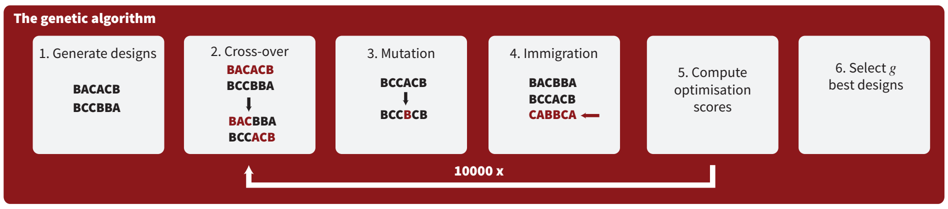


Figure 2: Graphical representation of the genetic algorithm. The examples in each step are pieces of experimental designs with 3 different trial types (A, B, C). In the example, the inter-trial interval is ignored.

4. **Immigration.** Add new random designs to the population.
5. **Natural selection.** Compute optimality scores and select G best designs.
6. Repeat steps 2–5 until a stopping rule is met.

A crucial part of the algorithm is drawing random designs from the population of designs (in step 1 and step 4). This could for example be achieved by using m-sequences to decide the order of the stimuli, the stimuli evenly spaced in time. We show in Section 3.4 how to sample random designs using **neurodesign**.

Simulation-based optimization

While the genetic algorithm has been shown to be a powerful algorithm for experimental design optimization (Kao *et al.* 2009), there are certain instances where more strict control of the design is desired. For example, in cognitive control, the occurrence of one condition versus another is sometimes varied between subjects. In those instances, it is crucial that there is strict control of the proportion of trial types that are shown. The genetic algorithm punishes deviance from the proportions measured by F_f, but strict control cannot be ensured. Therefore, we included a random design generator. The implementation is identical to the *genetic algorithm*, with the *crossover*- and *mutation*-steps removed and only the *immigration*-step being performed.

3. Python module neurodesign

3.1. Installation

neurodesign is available on PyPi and can be installed as:

```
$ pip install neurodesign
```

Next, we will give an introduction to the Python module. For all functionality, please refer to the manual.

3.2. Specifying the characteristics of the experiment

In a first step, the experiment should be described in the class called '`experiment`'. This contains general information, such as the number of stimuli and the duration of the experiment,

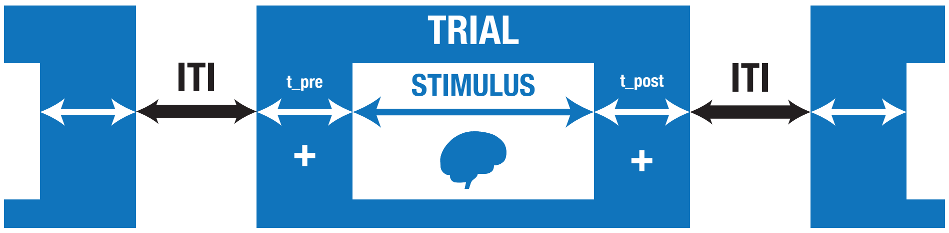


Figure 3: The basic layout of an experimental trial.

but also more specific information, such as the model with which the inter trial intervals (ITI) are sampled. This function will generate the assumed covariance matrix, the drift function and the whitening matrix. All parameters are described in Table 1, while a graphical representation of components of an experiment are described in Figure 3. We define a simple experimental setup with 20 trials and 3 conditions, which we will use to exemplify the next functions:

```
>>> import neurodesign
>>> EXP = neurodesign.experiment(TR = 1.2, n_trials = 20,
...   P = [0.3, 0.3, 0.4], C = [[1, -1, 0], [0, 1, -1]], n_stimuli = 3,
...   rho = 0.3, stim_duration = 1, ITImodel = "uniform", ITImin = 2,
...   ITImax = 4)
```

3.3. Generating a design matrix

Within the defined experimental setup, we can now define a design matrix, develop the design matrix and compute the optimality scores using the class ‘`design`’. We use equal weights for the different optimality criteria for the weighted average optimality attribute. The only input required is the stimulus order, the ITI’s and an object of class ‘`neurodesign.experiment`’:

```
>>> import neurodesign
>>> DES1 = neurodesign.design(
...   order = [0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1],
...   ITI = [2] * 20, experiment = EXP)
>>> DES1.designmatrix(); DES1.FCalc(weights = [0.25, 0.25, 0.25, 0.25])
```

Now using **matplotlib** (Hunter 2007), we can plot the convolved design matrix (see Figure 4):

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(DES1.Xconv)
```

We can now define a new design and compare both designs:

```
>>> DES2 = neurodesign.design(
...   order = [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
...   ITI = [2] * 20, experiment = EXP)
>>> DES2.designmatrix(); DES2.FCalc(weights = [0.25, 0.25, 0.25, 0.25])
>>> print("Ff of Design 1: "+str(DES1.Ff))
```

Argument	Description
`TR`	The repetition time of the scanner.
`n_stimuli`	The number of different stimulus types or conditions.
`P`	The probabilities of each stimulus type.
`C`	The contrast matrix.
`rho`	The assumed autocorrelation coefficient.
`n_trials`	The number of trials in the experiment. Either specify `n_trials` *or* `duration`.
`duration`	The total duration (seconds) of the experiment. Either specify `duration` *or* `n_trials`.
`resolution` (default = 0.1)	The resolution of the design matrix.
`t_pre` (default = 0)	Duration (seconds) of the trial before the stimulus presentation (e.g., fixation cross).
`stim_duration`	The duration (seconds) of the stimulus.
`t_post` (default = 0)	Duration (seconds) of the trial after the stimulus presentation.
`maxrep` (default = `None`)	The maximum number of times a stimulus is repeated consecutively.
`hardprob` (default = `False`)	`True` if the probabilities should be exactly the same as in `P`.
`restnum` (default = 0)	The number of trials between rest blocks.
`restdur` (default = 0)	The duration (seconds) of a rest block.
`ITImodel`	Which ITI model to sample from. Possibilities: `"fixed"`, `"uniform"` or `"exponential"`.
`ITImin`	The minimum ITI (used with `"uniform"` or `"exponential"` ITImodel).
`ITImean`	The mean ITI (used with `"fixed"` or `"exponential"` ITImodel).
`ITImax`	The max ITI (used with `"uniform"` or `"exponential"` ITImodel).
`confoundorder` (default = 3)	The order to which confounding is controlled.

Table 1: The arguments for an object of class '`neurodesign.experiment`'.

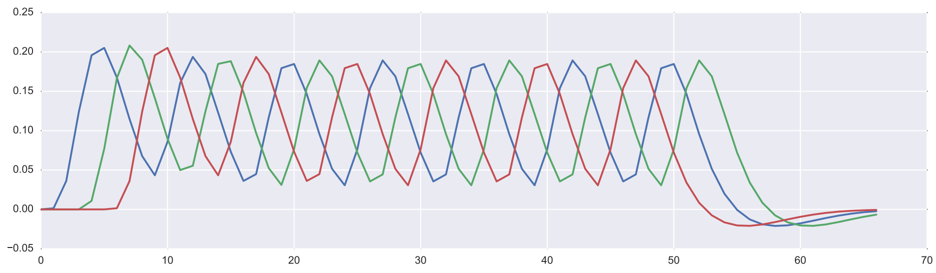


Figure 4: Plot of the convolved design matrix.

```
>>> print("Ff of Design 2: "+str(DES2.Ff))
>>> print("Fd of Design 1: "+str(DES1.Fd))
>>> print("Fd of Design 2: "+str(DES2.Fd))

Ff of Design 1: 0.8571428571428572
Ff of Design 2: 0.4285714285714286
Fd of Design 1: 0.0879554751884
Fd of Design 2: 0.266229261071
```

As the second design ignores the presence of the third condition, the frequency optimality (F_f) is much worse. However, the blocked character of the design largely improves the detection power. The principles of the genetic algorithm, such as crossover, can be applied to the designs:

```
>>> DES3,DES4 = DES1.crossover(DES2, seed = 2000)
>>> DES3.order

[0, 1, 2, 0, 1, 2, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]

>>> DES4.order

[0, 0, 0, 0, 0, 1, 1, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1]
```

3.4. Generating a random design

The package contains functions to generate random designs. We can generate a random order of stimuli using the function `neurodesign.generate.order`. Below, we generate a random order of stimuli, which is done by sampling from a multinomial distribution. Below, the resulting probabilities for each trialtype are shown.

```
>>> order = neurodesign.generate.order(nstim = 4, ntrials = 100,
...   probabilities = [0.25, 0.25, 0.25, 0.25], ordertype = "random",
...   seed = 1234)
print(order[:10])
from collections import Counter
Counter(order)

[3, 0, 0, 2, 0, 0, 2, 2, 2, 0]
Counter({0: 36, 1: 22, 2: 22, 3: 20})
```

Similarly, we can generate ITI's from 3 different distributions: fixed (all ITI's equal), uniform or from a truncated exponential distribution. Below we show how to use and evaluate the output of function `neurodesign.generate.iti`.

```
>>> iti,lam = neurodesign.generate.iti(ntrials = 40, model = "exponential",
...   min = 2, mean = 3, max = 8, resolution = 0.1, seed = 2134)
>>> print(iti[:10])
>>> print("mean ITI: %s \nmin ITI: %s \n max ITI: %s"
...   %(round(sum(iti)/len(iti), 2),
...   round(min(iti), 2), round(max(iti), 2)))
```

Argument	Description
`experiment`	The experimental setup of the fMRI experiment (of class '`neurodesign.experiment`').
`G` (default = 20)	The size of each generation.
`R` (default = `[0.4, 0.4, 0.2]`)	The rate with which the orders are generated from (a) blocked designs, (b) random designs and (c) m-sequences.
`q` (default = 0.01)	The percentage of mutations in each generation.
`weights`	The weights attached to $[F_e, F_d, F_f, F_c]$.
`I` (default = 4)	The number of immigrants in each generation.
`preruncycles`	The number of pre-run cycles to find the maximum value of F_e and F_d.
`cycles`	The number of cycles in the optimization.
`seed`	The random seed for the optimization.
`Aoptimality` (default = `True`)	Optimizes A-optimality if `True`, else D-optimality.
`convergence` (default = 1000)	After how many stable iterations is there convergence.
`folder`	The local folder to save the output.
`outdes` (default = 3)	The number of designs to be saved.
`optimization` (default = `"GA"`)	The optimizer of choice (`"GA"` or `"simulation"`).

Table 2: The arguments for an object of class '`neurodesign.optimisation`'.

```
[0.  2.  2.1 2.  2.  2.  5.4 2.  2.4 5.1]
mean ITI: 2.93
min ITI: 0.0
max ITI: 6.9
```

3.5. Optimizing the design

To optimize the design, we use the '`neurodesign.optimization`' class. All parameters are described in Table 2.

```
>>> POP = neurodesign.optimisation(experiment = EXP,
...   weights = [0, 0.5, 0.25, 0.25], preruncycles = 10000,
...   cycles = 10000, folder = "./", seed = 100, optimisation = "GA")
>>> POP.optimise()
```

4. neurodesign: The GUI

To make the methods more publicly available, we have created a graphical user interface running in a web application. The back-end of the application is written in Python and uses the Python module **neurodesign** described above, the front-end is generated using **django** (**django** Core Team 2013), and the application is deployed through a multi-container docker environment on Amazon Web Services.

There are 5 crucial windows of the GUI: main input, contrasts and probabilities, review, console, and settings. The main input window has fields for most parameters from Table 1.

Only the parameters P and C are asked in the second window ('Contrasts and probabilities'). The review window shows all parameters and also prints out the default settings for the genetic algorithm. These parameters, presented in Table 2, can be adjusted in the settings window. The console allows for the optimizations to be started, stopped and followed. When a design optimization is started, the user receives an email with a link to the console where the optimization can be followed (Figure 5). Once the optimization is finished, a zip file can be downloaded containing a chosen number of designs. Each design contains the onsets for each stimulus, a report with design diagnostics (such as collinearity among regressors, see Figure 6), and a script. The script can be used for future reference, or for regenerating the designs locally.

5. Comparison with other software

There are a few alternative design optimization programs available:

`mseq`

This function, available in the script `http://gru.stanford.edu/svn/matlab/buracas.m` was distributed with Buračas and Boynton (2002) and can be used to generate m-sequences using MATLAB (The MathWorks Inc. 2019).

riksneurotools

The tool **riksneurotools** is provided by Henson (2006) to compute the GLM efficiency of experimental designs for fMRI in MATLAB and is available online at `https://github.com/MRC-CBU/riksneurotools`.

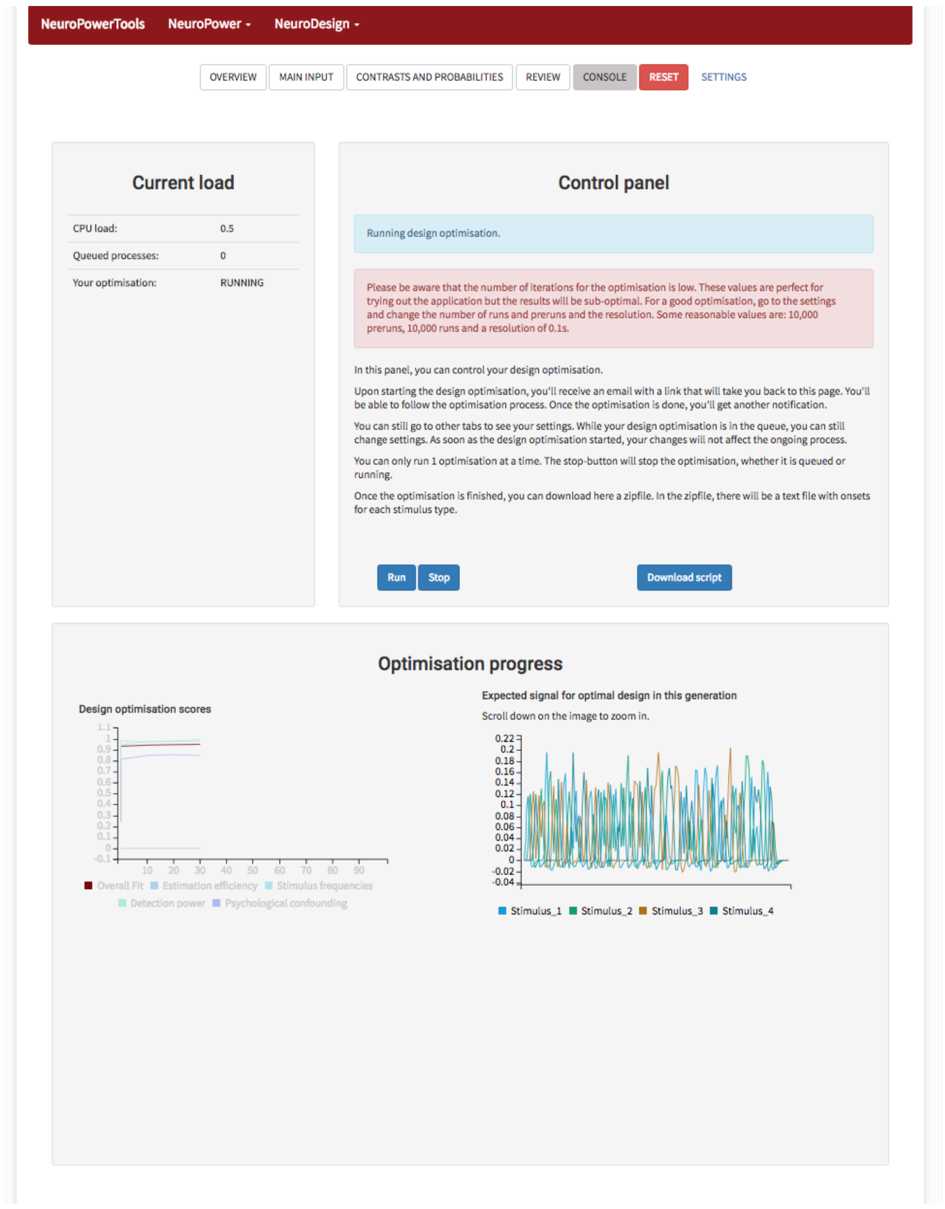
mttfmri *toolbox*

The multiple trial type fMRI MATLAB toolbox (**mttfmri**) can generate m-sequences, random designs and blocked designs, as well as compute their efficiency. The toolbox is distributed alongside Liu (2004a) and Liu (2004b). Using the toolbox, a theoretical trade-off between efficiency and power can be calculated. The toolbox assumes a fixed inter-trial interval. The toolbox can be obtained from `http://fmriserver.ucsd.edu/tliu/mttfmri_toolbox.html`.

Optseq2

Optseq2 is part of the **Freesurfer** suite (Dale 1999) and is a C toolbox for automatically scheduling events for rapid-presentation event-related experiments. It is available as a command line program, and it uses simulation-based optimization. With the toolbox, the order of the stimuli is optimized to be first-order counter-balanced, and random amounts of null stimulus are inserted in the design to jitter ITI's. The total duration and the minimum and maximum null time can be specified. Three cost functions can be optimized:

- `eff`: the efficiency: $1/\text{trace}(C(X^\top X)^{-1}C^\top)$.
- `vrfavg`: the average variance reduction factor (VRF): $\text{Avg}(1/C(X^\top X)^{-1}C^\top)$.
- `vrfstd`: a weighted combination of the average and the standard deviation of the VRF's.



Current load

CPU load:	0.5
Queued processes:	0
Your optimisation:	RUNNING

Control panel

Running design optimisation.

Please be aware that the number of iterations for the optimisation is low. These values are perfect for trying out the application but the results will be sub-optimal. For a good optimisation, go to the settings and change the number of runs and preruns and the resolution. Some reasonable values are: 10,000 preruns, 10,000 runs and a resolution of 0.1s.

In this panel, you can control your design optimisation.

Upon starting the design optimisation, you'll receive an email with a link that will take you back to this page. You'll be able to follow the optimisation process. Once the optimisation is done, you'll get another notification.

You can still go to other tabs to see your settings. While your design optimisation is in the queue, you can still change settings. As soon as the design optimisation started, your changes will not affect the ongoing process.

You can only run 1 optimisation at a time. The stop-button will stop the optimisation, whether it is queued or running.

Once the optimisation is finished, you can download here a zipfile. In the zipfile, there will be a text file with onsets for each stimulus type.

Optimisation progress

Design optimisation scores

Expected signal for optimal design in this generation

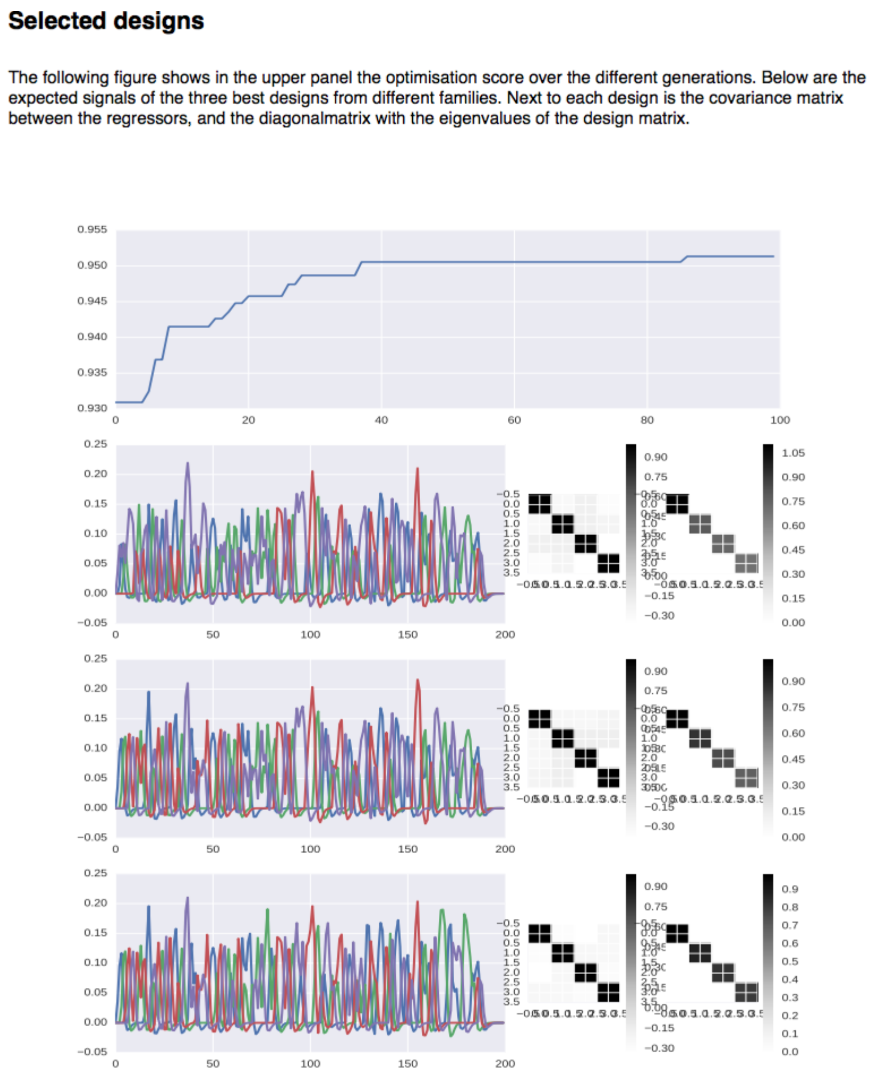
Scroll down on the image to zoom in.

Figure 5: Screenshot of the console where the optimization can be followed. Every 10 generations, the design is updated with the latest score and the best design.

Optseq2 by default optimizes the design to use the FIR model, but can be customized to optimize the design to use the convolved model.

Genetic algorithm

Wager and Nichols (2003) proposed the use of the genetic algorithm for design optimization, and published a MATLAB toolbox alongside, available online at `http://psych.colorado.edu/~tor/Software/genetic_algorithms.html`. The application implements a genetic algorithm to optimize experimental designs similar to this implementation. The differences are discussed below.



Selected designs

The following figure shows in the upper panel the optimisation score over the different generations. Below are the expected signals of the three best designs from different families. Next to each design is the covariance matrix between the regressors, and the diagonalmatrix with the eigenvalues of the design matrix.

Figure 6: Screenshot of the report describing the optimization and the best 3 designs from the optimization.

ER-fMRI

ER-fMRI is a software described in Kao (2009) which implements the genetic algorithm described in Kao *et al.* (2009). The implementation is similar to the implementation offered by Wager and Nichols (2003) with two key differences: (1) To compute the weighted average F of the different optimization scores, the scores (F_e, F_d, F_c, F_f) need to be on a unit scale. While Wager and Nichols (2003) rescale the scores for efficiency (F_e) and detection power (F_d) in each iteration of the genetic algorithm, Kao *et al.* (2009) propose to perform a pre-run in which only the F_e or F_d optimized, in order to provide an optimal score. In other words, Wager and Nichols (2003) rescale the scores within populations, while Kao *et al.* (2009) rescale the scores over populations. (2) the algorithm starts not only with random designs, but also includes aforementioned the m-sequences and blocked designs.

Main differences with ***neurodesign***

neurodesign includes all options of all the implementations above: the library can (1) produce m-sequences, (2) calculate efficiency scores, (3) generate random and blocked designs, (4) optimize the multi-objective criterion presented in Wager and Nichols (2003) and Kao *et al.* (2009), as well as (5) optimize designs using the genetic algorithm and a simulation-based optimizer. Whereas the approach of **neurodesign** is very closely related to **ER-fMRI**, we implement more elaborate control of the ITI's: in **ER-fMRI** and **Genetic algorithm**, the ITI's are modeled simply by introducing null events at random places during the experiment instead of experimental stimulation. This offers control of the minimum and maximum ITI, but it does not control the distribution of ITI's, as is implemented here. Furthermore, contrary to the other packages, we do not make a distinction between event-related responses and epoch responses. Event-related responses are often seen as a stimulus that results in a direct and short response, while an epoch response represents brain activation over a longer time (for example 2 seconds). Assuming an event-related response in fact equals assuming a response with duration equal to the resolution of the design. Since **neurodesign** supports any stimulus duration, it implicitly covers event-related responses and allows epoch responses. **neurodesign** also comes with a GUI, which increases the ease of use and accessibility of the application. At the same time, reproducibility of results is ensured by allowing scripting (or downloading scripts) that can be run in a containerized computing environment.

6. Design optimization and statistical power

Optimization

To demonstrate the effect of optimizing the experimental design, we performed a simulation study. We compare 3 possible designs: a design optimized using the genetic algorithm (GA), a design optimized using simulations (SIM), and a random design (not optimized – RND). We generated 100 random designs, and report the interval between the 5th and 95th percentile as a non-parametric prediction interval. All designs are generated with **neurodesign**.

We are planning an fMRI study with 3 different stimuli with equal probabilities. The TR is 2 seconds, and there are 450 trials of 1 second each. The ITI's are sampled from a truncated exponential distribution with (min, mean, max)-values of $(0.3, 1, 4)$ seconds. As such, the total duration of the experiment is 15 minutes, and there are 450 observations. We optimize for 4 different contrasts: $[1, 0, 0], [0, 1, 0], [0, 0, 1], [1, 0, -1]$. We ran 1000 cycles (pre-run and optimization). We choose the following values for the multi-objective criterion: $w_c = 0.25, w_f = 0.25, w_d = 0.5$, since we are mainly interested in statistical power, while keeping the predictability and frequency of the design under control. The evolution of the optimization scores can be seen in Figure 7. 90% of the optimization scores (F^{RND}) of the random design are in $[0.61, 0.70]$, while the resulting scores for the optimized designs are $F^{\text{GA}} = 0.87$ and $F^{\text{SIM}} = 0.80$. The three resulting designs can be seen in Figure 8.

Simulation

Based on the three obtained designs, we simulate fMRI data. We simulate data according to Equation 1, with X the design matrix obtained from the previous step, $\beta = (0.5, 0, -0.5)^\top$ and $\sigma = 1$. For simplicity, we ignore the temporal autocorrelation and analyze the data

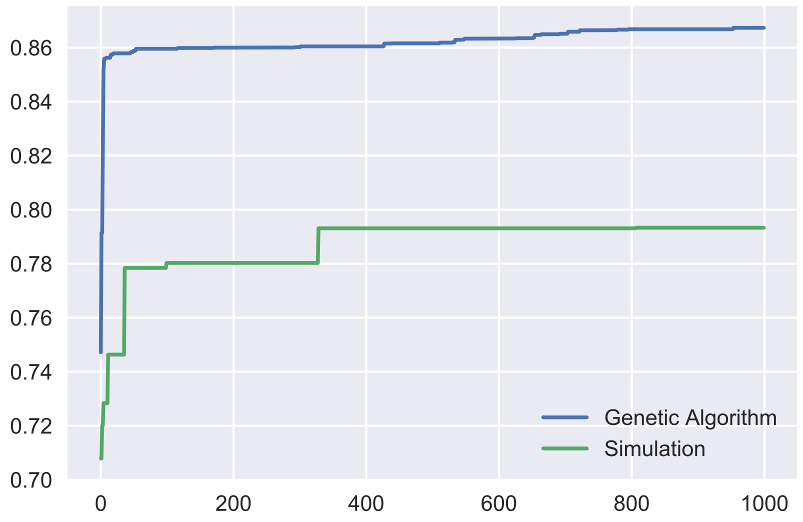


Figure 7: Multi-objective criterion evolution over 1000 iterations for an experimental design with 3 stimuli of 15 minutes.

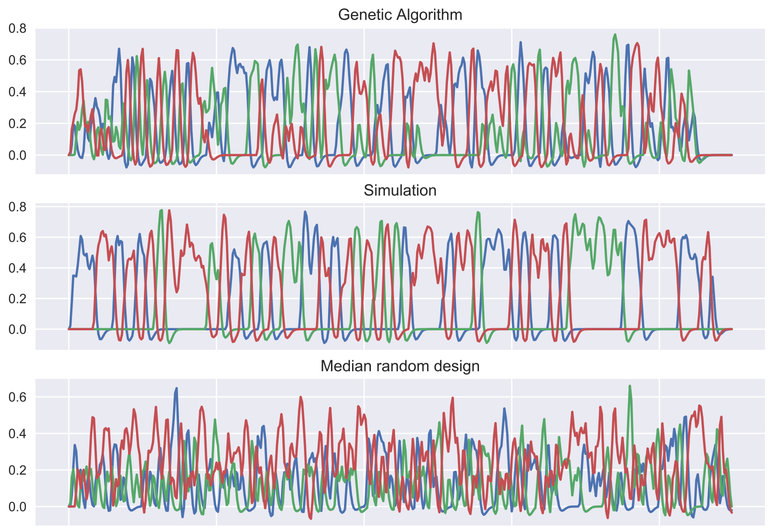


Figure 8: The resulting three designs after optimizing using the genetic algorithm, simulation based optimization, or without optimizing (median F).

using a simple linear regression model. Specifically, we look at the contrasts $(1, 0, 0)$ and $(1, 0, -1)$, respectively corresponding to absolute effect sizes of 0.5 and 1.0. The distribution of the resulting T-statistics based on $1,000$ simulations are shown in Figure 9. We assume a single test, thus performing statistical inference with $\alpha = 0.95$. From the simulations, the observed statistical power is calculated as the percentage a T-value exceeds the threshold, $\#T > t_\alpha/10^4$. Table 3 shows the resulting observed statistical power. We effectively show how using **neurodesign** significantly increases the statistical power, compared to a random

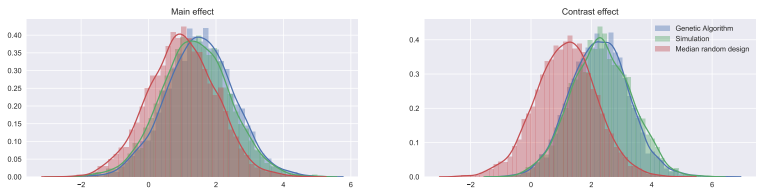


Figure 9: The distributions of T-statistics when simulating a BOLD signal using the three designs obtained using (1) genetic algorithm optimization, (2) simulation based optimization, (3) random draws (median F).

	Contrast $(1, 0, 0)$	Contrast $(1, 0, -1)$
Genetic Algorithm	0.45	0.73
Simulation-based	0.40	0.75
Random	0.26 [0.22, 0.31]	0.54 [0.21, 0.76]

Table 3: Observed statistical power from 10000 fMRI simulations with 3 different designs: (1) optimized using the genetic algorithm, (2) optimized using simulations and (3) not optimized (median [P5, P95]).

design. Note that even though certain randomly drawn designs result in higher power, these necessarily score lower in the other metrics, such as predictability (F_c) and stimulus frequencies (F_f), since none of the random designs result in a higher score for the multi-objective criterion.

7. Discussion

7.1. Default settings

Both the initialization of the experiment, represented by the class experiment in the Python module and the main input window in the GUI, and the genetic algorithm, represented by the class population in the Python module and the settings window in the GUI, have some default settings. The default settings of the Python module, shown in Tables 1 and 2, are pre-set to ensure a good initial optimization, while the GUI's default settings are pre-set for short optimization duration. While the default settings for the GUI can lead to a sub-optimal design, the user is warned (with a big red text block at the top of the page) that the settings should be changed if a good optimization is required. We have chosen these sub-optimal defaults for the GUI to provide a fast run through for first time users, as well as to avoid memory and CPU overload on the server end. For the experiment, we assume a priori that there are no rest blocks and that the trial only consists of stimulation (no fixation cross etc.). There is by default no limit on the maximum number of times a stimulus can be repeated, and the stimulus frequency is not controlled with a hard limit. The default resolution in the Python module is 0.1 seconds, while in the GUI it is 0.25 seconds. For the genetic algorithm, the default settings are as follows. The optimization calculates the A-optimality. In each generation, the percentage of mutations is 1%, the number of immigrants is 4 designs and the size of each generation is 20 designs, as is suggested by Kao (2009). When generating new designs, there are 40% blocked designs, 40% random designs and 20% m-sequences. Convergence is reached when the score is stable for 1000 generations. There are no default settings on the number of cycles in the Python module, while the GUI runs by default 10 cycles to find the maximum F_d and F_e and 100 cycles for the optimization (again with a clear message that this can be increased for better results).

7.2. Reproducibility

In line with the recent effort to make neuroimaging research fully reproducible, this application makes it possible to track the exact source of each design. Low level reproducibility is provided by making a script available for download with which the optimization can be regenerated. Running this script in Python, given that the required libraries are installed, will repeat the analysis. However, this script will repeat the analysis but does not guarantee the same results as the specific configuration of the computer on which the analysis is run can influence the results.

Higher level reproducibility, that guarantees replicability not only of the analysis but also of the results, is possible with the use of Docker containers, which is a small piece of software that emulates a given computational configuration (operating system, libraries, Python packages, ...). Based on the model presented by BIDS-apps (Gorgolewski, Alfaro-Almagro, Auer, Bellec, Capota, Mallar Chakravarty, Churchill, Cameron Craddock, Devenyi, Eklund, Esteban, Flandin, Ghosh, Swaroop Guntupalli, Jenkinson, Keshavan, Kiar, Raamana, Raffelt, Steele, Quirion, Smith, Strother, Varoquaux, Yarkoni, Wang, and Poldrack 2017), our analyses run in Docker containers that are open-source and available for download at https://hub.docker.com/r/neuropower/neuropower/. Running the following command in a terminal will replicate the analysis that has been performed through the GUI.

```
$ docker run -v /location_where_the_script_is/:/local \
>   -it neuropower/neuropower Python /local/name_of_the_script.py
```

This use of Docker containers is not only well suited for reproducibility of the GUI, but also allows the replication of results from a Python script (given that a random seed is set).

8. Conclusion

We present a toolbox for optimizing fMRI designs. The toolbox is an extension of currently available toolboxes, allowing for more complex design and better control and optimization of timing of stimuli. The toolbox is available through different modalities: a user-friendly GUI accessible at http://www.neuropowertools.org and a Python package. The code is available on https://www.github.com/neuropower.

Acknowledgments

This work was supported by the Laura and John Arnold Foundation. J.D. has received funding from the European Unions Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 706561.

References

Buračas GT, Boynton GM (2002). "Efficient Design of Event-Related fMRI Experiments Using M-Sequences." *Neuroimage*, **16**(3), 801–813. doi:10.1006/nimg.2002.1116.

Button KS, Ioannidis JPA, Mokrysz C, Nosek BA, Flint J, Robinson ESJ, Munafò MR (2013). "Power Failure: Why Small Sample Size Undermines the Reliability of Neuroscience." *Nature Reviews Neuroscience*, **14**(5), 365–376. doi:10.1038/nrn3475.

Dale AM (1999). "Optimal Experimental Design for Event-Related fMRI." *Human Brain Mapping*, **8**(2–3), 109–114. doi:10.1002/(sici)1097-0193(1999)8:2/3<109::aid-hbm7>3.3.co;2-n.

Durnez J, Degryse J, Moerkerke B, Seurinck R, Sochat V, Poldrack R, Nichols T (2016). "Power and Sample Size Calculations for fMRI Studies Based on the Prevalence of Active Peaks." doi:10.1101/049429. BioRxiv.

Durnez J, Moerkerke B, Nichols TE (2014). "Post-Hoc Power Estimation for Topological Inference in fMRI." *Neuroimage*, **84**, 45–64. doi:10.1016/j.neuroimage.2013.07.072.

Eklund A, Nichols TE, Knutsson H (2016). "Cluster Failure: Why fMRI Inferences for Spatial Extent Have Inflated False-Positive Rates." *Proceedings of the National Academy of Sciences of the United States of America*, **113**(28), 7900–7905. doi:10.1073/pnas.1602413113.

Gorgolewski KJ, Alfaro-Almagro F, Auer T, Bellec P, Capota M, Mallar Chakravarty M, Churchill NW, Cameron Craddock R, Devenyi GA, Eklund A, Esteban O, Flandin G, Ghosh SS, Swaroop Guntupalli J, Jenkinson M, Keshavan A, Kiar G, Raamana PR, Raffelt D, Steele CJ, Quirion PO, Smith RE, Strother SC, Varoquaux G, Yarkoni T, Wang Y, Poldrack RA (2017). "BIDS Apps: Improving Ease of Use, Accessibility and Reproducibility of Neuroimaging Data Analysis Methods." *PLOS Computational Biology*. doi:10.1371/journal.pcbi.1005209.

Hayasaka S, Peiffer AM, Hugenschmidt CE, Laurienti PJ (2007). "Power And Sample Size Calculation for Neuroimaging Studies By Non-Central Random Field Theory." *Neuroimage*, **37**(3), 721–730. doi:10.1016/j.neuroimage.2007.06.009.

Henson RN (2006). "Efficient Experimental Design for fMRI." In K Friston, J Ashburner, S Kiebel, T Nichols, W Penny (eds.), *Statistical Parametric Mapping: The Analysis of Functional Brain Images*, pp. 193–210. Elsevier, London.

Hunter JD (2007). "**matplotlib**: A 2D Graphics Environment." *Computing in Science & Engineering*, **9**(3), 90–95. doi:10.1109/MCSE.2007.55.

Ioannidis JPA (2005). "Why Most Published Research Findings Are False." *PLoS Medicine*, **2**(8), e124. doi:10.1371/journal.pmed.0020124.

Kao MH (2009). "Multi-Objective Optimal Experimental Designs for ER-fMRI Using MATLAB." *Journal of Statistical Software*, **30**(11), 1–13. doi:10.18637/jss.v030.i11.

Kao MH, Mandal A, Lazar N, Stufken J (2009). "Multi-Objective Optimal Experimental Designs for Event-Related fMRI Studies." *Neuroimage*, **44**(3), 849–856. doi:10.1016/j.neuroimage.2008.09.025.

Lin YL, Phao FKH, Kao MH (2007). "Optimal Design of fMRI Experiments Using Circulant (Almost-)Orthogonal Arrays." *The Annals of Statistics*, **34**(6), 2483–2510. doi:10.1214/16-aos1531.

Liu T (2004a). "Efficiency, Power, and Entropy in Event-Related fMRI with Multiple Trial Types. Part I: Theory." *Neuroimage*, **21**(1), 387–400. doi:10.1016/j.neuroimage.2003.09.030.

Liu T (2004b). "Efficiency, Power, and Entropy in Event-Related fMRI with Multiple Trial Types. Part II: Design of Experiments." *Neuroimage*, **21**(1), 401–413. doi:10.1016/j.neuroimage.2003.09.031.

Mumford JA, Nichols TE (2008). "Power Calculation for Group fMRI Studies Accounting for Arbitrary Design and Temporal Autocorrelation." *Neuroimage*, **39**(1), 261–268. doi:10.1016/j.neuroimage.2007.07.061.

Open Science Collaboration (2015). "Estimating the Reproducibility of Psychological Science." *Science*, **349**(6251), aac4716. doi:10.1126/science.aac4716.

django Core Team (2013). ***django**: The Web Framework for Perfectionists with Deadlines.* Version 1.5, URL https://djangoproject.com.

The MathWorks Inc (2019). *MATLAB – The Language of Technical Computing, Version R2019a.* Natick, Massachusetts. URL http://www.mathworks.com/products/matlab/.

van Rossum G, *et al.* (2011). *Python Programming Language*. URL http://www.python.org.

Wager TD, Nichols TE (2003). "Optimization of Experimental Design in fMRI: A General Framework Using a Genetic Algorithm." *Neuroimage*, **18**(2), 293–309. doi:10.1016/s1053-8119(02)00046-0.

Affiliation:

Joke Durnez
Department of Psychology
Stanford University
560 Serra Mall
Stanford, CA 94305
E-mail: joke.durnez@gmail.com
URL: http://www.neuropowertools.org

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/

MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd