neurodesign Documentation

Release 0.0.13

Joke Durnez

Mar 01, 2018

Contents

We have built a GUI for this package, in the form of an online toolbox available at **‘www.neuropowertools.org<http://www.neuropowertools.org>‘_**. The GUI uses functions from the neurodesign package.

CHAPTER 1

Getting started

1.1 Installing NeuroDesign

neurodesign is available on pypi. To install, run:

```
pip install neurodesign
```

1.2 About Design Optimisation Using the Genetic Algorithm

This toolbox is for the optimization of experimental designs for fMRI. Minimizing the variance of the design matrix will help detect or estimate (depending on the outcome of interest) the effect researchers are looking for. The genetic algorithm for experimental designs was introduced by Wager and Nichols (2002) and further improved by Kao, Mandal, Lazar and Stufken (2009). We implemented these methods in a python package and a userfriendly web-application and introduced some improvements and allows more flexibility for the experimental setup.

1.3 Design efficiency

The core idea of this package is to run an optimization algorithm that (among others) optimizes the design efficiency of an fMRI design using A-optimality, with this formula:

CHAPTER 2

Neurodesign documentation

2.1 Neurodesign: design optimisation

class src.neurodesign.**design**(*order*, *ITI*, *experiment*, *onsets=None*)

This class represents an experimental design for an fMRI experiment.

Parameters

- **order** (*list of integers*) – The stimulus order.
- **ITI** (*list of floats*) – The ITI's between all stimuli.
- **experiment** (*experiment object*) – The experimental setup.
- **onsets** (*list of floats*) – The onsets of all stimuli.

FCalc(*weights*, *Aoptimality=True*, *confoundorder=3*)

Compute weighted average of efficiencies.

Parameters **weights** (*list of floats*) – Weights given to each of the efficiency metrics in this order: Estimation, Detection, Frequencies, Confounders.

FcCalc(*confoundorder=3*)

Compute confounding efficiency.

Parameters **confoundorder** (*integer*) – To what order should confounding be protected

FdCalc(*Aoptimality=True*)

Compute detection power.

Parameters **Aoptimality** (*boolean*) – Kind of optimality to optimize: A- or D-optimality

FeCalc(*Aoptimality=True*)

Compute estimation efficiency.

Parameters **Aoptimality** (*boolean*) – Kind of optimality to optimize, A- or D-optimality

FfCalc()

Compute efficiency of frequencies.

check_hardprob()
Function to check whether frequencies of stimuli are **exactly** the prespecified frequencies.

Returns probcheck Boolean indicating probabilities are respected

check_maxrep(*maxrep*)
Function to check whether design does not exceed maximum repeats within design.

Parameters **maxrep** (*integer*) – How many times should a stimulus maximally be repeated.

Returns repcheck Boolean indicating maximum repeats are respected

crossover(*other*, *seed=1234*)
Function to crossover design with other design and create offspring.

Parameters

- **other** (*design object*) – The design with which the design will be mixed
- **seed** (*integer or None*) – The seed with which the change point will be sampled.

Returns offspring List of two offspring designs.

designmatrix()
Expand from order of stimuli to a fMRI timeseries.

mutation(*q*, *seed=1234*)
Function to mutate q% of the stimuli with another stimulus.

Parameters

- **q** (*float*) – The percentage of stimuli that should be mutated
- **seed** (*integer or None*) – The seed with which the mutation points are sampled.

Returns mutated Mutated design

class src.neurodesign.**experiment**(*TR*, *P*, *C*, *rho*, *stim_duration*, *n_stimuli*, *ITImodel=None*, *ITImin=None*, *ITImax=None*, *ITImean=None*, *restnum=0*, *restdur=0*, *t_pre=0*, *t_post=0*, *n_trials=None*, *duration=None*, *resolution=0.1*, *FeMax=1*, *FdMax=1*, *FcMax=1*, *FfMax=1*, *maxrep=None*, *hardprob=False*, *confoundorder=3*)
This class represents an fMRI experiment.

Parameters

- **TR** (*float*) – The repetition time.
- **P** (*ndarray*) – The probabilities of each trialtype.
- **C** (*ndarray*) – The contrast matrix. Example: np.array([[1,-1,0],[0,1,-1]])
- **rho** (*float*) – AR(1) correlation coefficient
- **n_stimuli** (*integer*) – The number of stimuli (or conditions) in the experiment.
- **n_trials** (*integer*) – The number of trials in the experiment. Either specify n_trials **or** duration.
- **duration** (*float*) – The total duration (seconds) of the experiment. Either specify n_trials **or** duration.
- **resolution** (*float*) – the maximum resolution of design matrix
- **stim_duration** (*float*) – duration (seconds) of stimulus

- **t_pre** (*float*) – duration (seconds) of trial part before stimulus presentation (eg. fixation cross)
- **t_post** (*float*) – duration (seconds) of trial part after stimulus presentation
- **maxrep** (*integer or None*) – maximum number of repetitions
- **hardprob** (*boolean*) – can the probabilities differ from the nominal value?
- **confoundorder** (*integer*) – The order to which confounding is controlled.
- **restnum** (*integer*) – Number of trials between restblocks
- **restdur** (*float*) – duration (seconds) of the rest blocks
- **ITImodel** (*string*) – Which model to sample from. Possibilities: "fixed","uniform","exponential"
- **ITImin** (*float*) – The minimum ITI (required with "uniform" or "exponential")
- **ITImean** (*float*) – The mean ITI (required with "fixed" or "exponential")
- **ITImax** (*float*) – The max ITI (required with "uniform" or "exponential")

CreateLmComp()
This function generates components for the linear model: hrf, whitening matrix, autocorrelation matrix and CX

CreateTsComp()
This function computes the number of scans and timpoints (in seconds and resolution units)

canonical()
This function generates the canonical hrf

Parameters **resolution** (*float*) – resolution to sample the canonical hrf

countstim()
Function to compute some arguments depending on other arguments.

static drift(*s*, *deg=3*)
Function to compute a drift component

max_eff()
Function to compute maximum efficiency for Confounding and Frequency efficiency.

static spm_Gpdf(*s*, *h*, *l*)
Function to generate gamma pdf

class src.neurodesign.**optimisation**(*experiment, weights, preruncycles, cycles, seed=None, I=4, G=20, R=[0.4, 0.4, 0.2], q=0.01, Aoptimality=True, folder=None, outdes=3, convergence=1000, optimisation='GA'*)
This class represents the population of experimental designs for fMRI.

Parameters

- **experiment** (*experiment*) – The experimental setup of the fMRI experiment.
- **G** (*integer*) – The size of the generation
- **R** (*list*) – with which rate are the orders generated from ['blocked','random','mseq']
- **q** (*float*) – percentage of mutations
- **weights** (*list*) – weights attached to Fe, Fd, Ff, Fc
- **I** (*integer*) – number of immigrants

- **preruncycles** (*integer*) – number of prerun cycles (to find maximum Fe and Fd)
- **cycles** (*integer*) – number of cycles
- **seed** (*integer*) – seed
- **Aoptimality** (*boolean*) – optimises A-optimality if true, else D-optimality
- **convergence** (*integer*) – after how many stable iterations is there convergence
- **folder** (*string*) – folder to save output
- **outdes** (*integer*) – number of designs to be saved
- **optimisation** (*string*) – The type of optimisation - 'GA' or 'random'

add_new_designs(*weights=None*, *R=None*)
This function generates the population.

Parameters

- **experiment** (*experiment*) – The experimental setup of the fMRI experiment.
- **weights** (*list of floats, summing to 1*) – weights for efficiency calculation.
- **seed** (*integer or None*) – The seed for ramdom processes.

change_seed()
Function to change the seed.

check_develop(*design*, *weights=None*)
Function to check and develop a design to the population. Function will check design against strict options and develop the design if valid.

Parameters

- **design** (*design object*) – Design to be added to population.
- **weights** (*list of floats, summing to 1*) – weights for efficiency calculation.

clear()
Function to clear results between optimalisations (maximum Fe, Fd or opt)

optimise(*optimisation='GA'*)
Function to run design optimization

to_next_generation(*weights=None*, *seed=1234*, *optimisation=None*)
This function goes from one generation to the next.

Parameters

- **weights** (*list of floats, summing to 1*) – weights for efficiency calculation.
- **seed** (*integer or None*) – The seed for random processes.
- **optimisation** (*string*) – The type of optimisation - 'GA' or 'simulation'

2.2 Generate: generating stimulus order and ITI's

src.generate.**iti**(*ntrials*, *model*, *min=None*, *mean=None*, *max=None*, *lam=None*, *resolution=0.1*, *seed=1234*)
Function will generate an order of stimuli.

Parameters

- **ntrials** (*integer*) – The total number of trials
- **model** (*string*) – Which model to sample from. Possibilities: "fixed","uniform","exponential"
- **min** (*float*) – The minimum ITI (required with "uniform" or "exponential")
- **mean** (*float*) – The mean ITI (required with "fixed" or "exponential")
- **max** (*float*) – The max ITI (required with "uniform" or "exponential")
- **resolution** (*float*) – The resolution of the design: for rounding the ITI's
- **seed** (*integer or None*) – The seed with which the change point will be sampled.

Returns iti A list with the created ITI's

src.generate.**order**(*nstim*, *ntrials*, *probabilities*, *ordertype*, *seed=1234*)
Function will generate an order of stimuli.

Parameters

- **nstim** (*integer*) – The number of different stimuli (or conditions)
- **ntrials** (*integer*) – The total number of trials
- **probabilities** (*list*) – The probabilities of each stimulus
- **ordertype** (*string*) – Which model to sample from. Possibilities: "blocked", "random" or "msequence"
- **seed** (*integer or None*) – The seed with which the change point will be sampled.

Returns order A list with the created order of stimuli

2.3 Msequence: generating msequences

class src.msequence.**Msequence**
A class for an order of experimental trials.

GenMseq(*mLen*, *stimtypeno*, *seed*)
Function to generate a random msequence given the length of the desired sequence and the number of different values.

Parameters

- **stimtypeno** (*integer*) – Number of different stimulus types
- **mLen** – The length of the requested msequence (**will be shorter than full msequence**)
- **seed** (*integer*) – Seed with which msequence is sampled.

Mseq(*baseVal*, *powerVal*, *shift=None*, *whichSeq=None*, *userTaps=None*)
Function to generate a specific msequence given the base and power values.

Parameters

- **powerVal** (*integer*) – The power of the msequence
- **baseVal** (*integer*) – The base value of the msequence (equivalent to number of stimuli)
- **shift** (*integer*) – Shift of the msequence

- **whichSeq** (*integer*) – Index of the sequence desired in the taps file.
- **userTaps** (*list*) – if user wants to specify own polynomial taps

tapsfnc()
Function to generate taps leading to msequences.

2.4 Report: summarise results from optimisation

src.report.**make_report**(*POP*, *outfile='NeuroDesign.pdf'*)
Function to create a report of a finished design optimisation.