fitr Documentation

Release 0.0.1

Abraham Nunes

Jul 24, 2017

Contents

1	Contents	1
2	Overview	29
3	Goals	31
4	Guiding Principles	33
5	What we're working on	35
6	Citing Fitr	37
7	Indices and tables	39
	Bibliography	41
	Python Module Index	43

CHAPTER 1

Contents

Installation

The current PyPI release of Fitr can be installed as follows:

```
pip install fitr
```

If you want the latest version on the GitHub master branch, install as follows:

```
pip install git+https://github.com/ComputationalPsychiatry/fitr.git
```

Conceptual Overview

Contents

Intro to Reinforcement Learning

Coming soon...

Modelling Behavioural Data

Pre-Built Models

We have included pre-built models for the following tasks:

Task	What it tests	Reference
N-Arm Bandit	Exploration/Exploitation	*[Daw2006]*
2-Step Task	Model-based vs. Model-free control	*[Daw2011]*

References

Fitting Models to Data

Model-Fitting Methods in Fitr

Fitr implements several model-fitting methods:

Method	Function	Reference
EM with Laplace Approximation	`fitr.EM()`	*[Huys2011]*
Empirical Priors	`fitr.EmpiricalPriors()`	*[Gershman2016]*
Markov Chain Monte-Carlo (Stan)	`fitr.MCMC()`	*[StanDevs]*

Here, "EM" refers to Expectation-Maximization.

References

Selecting the Best Model

Fitr facilitates model-selection via Bayesian Information Criterion (BIC), Aikake Information Criterion (AIC), and Bayesian Model Selection (BMS) *[Rigoux2014]*. BMS is a re-implementation of `spm_BMS()` from the Statistical Parametric Mapping toolbox for MATLAB.

References

Tutorials

We have several tutorials for Fitr written in Jupyter Notebooks:

1. Introductory tutorial (EM and Bayesian Model Selection)
2. Fitting a Model with MCMC
3. Use MCMC with your own Stan Code
4. Using Multiple Model-Fitting Routines for Same Model

Contributing to Fitr

Your contributions to Fitr are welcome and encouraged. Fitr is being developed on GitHub in order to facilitate improvements by the community. However, to ensure that Fitr develops as a robust piece of software, we have several guidelines for contributions. These have been chosen to mirror those of the SciPy/NumPy project.

Contributions to Fitr should have

1. **Unit tests**
    - It is important that Fitr functions well "out of the box," and this requires that code implemented in Fitr is appropriately tested.
    - Fitr uses Codecov.io to assess code coverage. In general, try to ensure that your new code is covered by unit tests.

- Unit tests are written in the `/tests` folder, where you can find examples of how unit tests are currently written.

2. **Documentation**
   - New code is not of great use unless the community knows what it is for and how to use it. As such, we ask that any new functions or modifications to existing functions carry the appropriate documentation.
   - If your contribution is substantial, it may be of use to write a tutorial, which are done with Jupyter Notebooks here.
   - Documentation of modules, classes, and functions can be done in Docstrings, then compiled with Sphinx and autodoc.
   - Documentation of Fitr code follows the SciPy/NumPy format
3. **Appropriate code style**
   - Fitr follows the PEP8 standard, and so we recommend that you run a linter to ensure that contributions adhere to this format.

Types of Contributions

At this early stage, we are open to any new contributions, big or small.

Many of the contribution requirements listed above were not adhered to at Fitr's inception, so even if you would like to help by correcting some of our past mistakes, this would be an important step toward Fitr's goals!

How to Contribute

1. Fork the GitHub repository
2. Create a new branch
3. Submit a pull request

Fitr's master branch is protected and requires Unit tests to pass, as well as 2 reviews before merging is allowed.

Requesting Features and Reporting Problems

Open an issue on the Fitr GitHub page, and we'll get on it as soon as possible!

Fitr API

Modules:

Inference

Modules:

Maximum-Likelihood Estimation

class `fitr.inference.mle.`**`MLE`**(*loglik_func*, *params*, *name='MLEModel'*)
    Maximum Likelihood parameter estimation

Attributes

name	(str) Name of the model being fit. We suggest using the free parameters.
loglik_func	(function) The log-likelihood function to be used for model fitting
params	(list) List of parameters from the rlparams module
nparams	(int) Number of free parameters in the model
param_rng	(list) List of strings denoting the parameter ranges (see rlparams module for further details)

Methods

fit(data, n_iterations=1000, opt_algorithm='BFGS')	Runs model-fitting algorithm
__printfitstart(self, n_iterations, algorithm, verbose)	(Private) function to print optimization info to console

fit(*data*, *n_iterations=1000*, *c_limit=0.0001*, *opt_algorithm='L-BFGS-B'*, *verbose=True*)

Runs the maximum a posterior model-fitting with empirical priors.

Parameters **data** : dict

Dictionary of data from all subjects.

n_iterations : int

Maximum number of iterations to allow.

c_limit : float

Threshold at which convergence is determined

opt_algorithm : {'L-BFGS-B'}

Algorithm to use for optimization. Only works at present with L-BFGS-B.

verbose : bool

Whether to print progress of model fitting

Returns ModelFitResult

Representation of the model fitting results

Expectation-Maximization

class fitr.inference.em.**EM**(*loglik_func*, *params*, *name='EMModel'*)

Expectation-Maximization with the Laplace Approximation *[Huys2011]*, [HuysEMCode].

Attributes

name	(str) Name of the model being fit. We suggest using the free parameters.
log-lik_func	(function) The log-likelihood function to be used for model fitting
params	(list) List of parameters from the rlparams module
nparams	(int) Number of free parameters in the model
param_rng	(list) List of strings denoting the parameter ranges (see rlparams module for further details)
prior	(scipy.stats distribution) The prior distribution over parameter estimates. Here this is fixed to a multivariate normal.
mu	(ndarray(shape=nparams)) The prior mean over parameters
cov	(ndarray(shape=(nparams,nparams))) The covariance matrix for prior over parameter estimates

Methods

fit(data, n_iterations=1000, c_limit=1, opt_algorithm='BFGS', diag=False, verbose=True)	Run the model-fitting algorithm
logposterior(x, states, actions, rewards)	Computes the log-posterior probability
group_level_estimate(param_est, hess_inv)	Updates the hyperparameters of the group-level prior
__printfitstart(self, n_iterations, c_limit, algorithm, init_grid, grid_reinit, dofull, early_stopping, verbose)	(Private) function to print optimization info to console
__printupdate(self, opt_iter, subject_i, posterior_ll, verbose)	(Private) function to print update on fit iteration to console

`fit`(*data*, *n_iterations=1000*, *c_limit=0.001*, *opt_algorithm='L-BFGS-B'*, *init_grid=False*, *grid_reinit=True*, *n_grid_points=5*, *n_reinit=1*, *dofull=True*, *early_stopping=True*, *verbose=True*)
Performs maximum a posteriori estimation of subject-level parameters

Parameters **data** : dict

Dictionary of data from all subjects.

n_iterations : int

Maximum number of iterations to allow.

c_limit : float

Threshold at which convergence is determined

opt_algorithm : {'BFGS', 'L-BFGS-B'}

Algorithm to use for optimization

init_grid : bool

Whether to initialize the optimizer using brute force grid search. If False, will sample from normal distribution with mean 0 and standard deviation 1.

grid_reinit : bool

If optimization does not converge, whether to reinitialize with values from grid search

n_grid_points : int

Number of points along each axis to evaluate during grid-search initialization (only meaningful if init_grid is True).

n_reinit : int

Number of times to reinitialize the optimizer if not converged

dofull : bool

Whether update of the full covariance matrix of the prior should be done. If False, the covariance matrix is limited to one in which the off-diagonal elements are set to zero.

early_stopping : bool

Whether to stop the EM procedure if the log-model-evidence begins decreasing (thereby reverting to the last iteration's results).

verbose : bool

Whether to print progress of model fitting

Returns ModelFitResult

Representation of the model fitting results

group_level_estimate (*param_est*, *hess_inv*, *dofull*, *verbose=True*)

Updates the group-level hyperparameters

Parameters **param_est** : ndarray(shape=(nsubjects, nparams))

Current parameter estimates for each subject

hess_inv : ndarray(shape=(nparams, nparams, nsubjects))

Inverse Hessian matrix estimate for each subject from the iteration with highest log-posterior probability

dofull : bool

Whether update of the full covariance matrix of the prior should be done. If False, the covariance matrix is limited to one in which the off-diagonal elements are set to zero.

verbose : bool

Controls degree to which results are printed

initialize_opt (*fn=None*, *grid=False*, *Ns=None*)

Returns initial values for the optimization

Parameters **fn** : function

Function over which grid search takes place

grid : bool

Whether to return initialization values from grid search

Ns : int

Number of points per axis over which to evaluate during grid search

Returns **x0** : ndarray

1 X N vector of initial values for each parameter

logposterior (*x*, *states*, *actions*, *rewards*)

Represents the log-posterior probability function

Parameters **x** : ndarray(nparams)

Array of parameters for single subject

states : ndarray(shape=[ntrials, nsteps])

Array of states encountered by subject

actions: ndarray(shape=[ntrials, nsteps])

Array of actions taken by subject

rewards : ndarray(shape=[ntrials, nsteps])

Array of rewards received by the subject.

Returns float

Log-posterior probability

Empirical Priors

class `fitr.inference.empiricalpriors.EmpiricalPriors`(*loglik_func*, *params*, *name='EmpiricalPriorsModel'*)

Inference procedure with empirical priors

Attributes

name	(str) Name of the model being fit. We suggest using the free parameters.
loglik_func	(function) The log-likelihood function to be used for model fitting
params	(list) List of parameters from the rlparams module
nparams	(int) Number of free parameters in the model
param_rng	(list) List of strings denoting the parameter ranges (see rlparams module for further details)

Methods

fit(data, n_iterations=1000, opt_algorithm='BFGS')	Runs model-fitting algorithm
logposterior(x, states, actions, rewards)	Computes the log-poseterior probability
__printfitstart(self, n_iterations, algorithm, verbose)	(Private) function to print optimization info to console
__printupdate(self, opt_iter, subject_i, posterior_ll, verbose)	(Private) function to print iteration info to console

`fit`(*data*, *n_iterations=1000*, *c_limit=0.001*, *opt_algorithm='L-BFGS-B'*, *verbose=True*)

Runs the maximum a posterior model-fitting with empirical priors.

Parameters data : dict

Dictionary of data from all subjects.

n_iterations : int

Maximum number of iterations to allow.

c_limit : float

Threshold at which convergence is determined

opt_algorithm : {'L-BFGS-B'}

Algorithm to use for optimization. Only works at present with L-BFGS-B.

verbose : bool

Whether to print progress of model fitting

Returns ModelFitResult

Representation of the model fitting results

logposterior (*x*, *states*, *actions*, *rewards*)
Represents the log-posterior probability function

Parameters **x** : ndarray(nparams)

Array of parameters for single subject

states : ndarray

Array of states encountered by subject. Number of rows should reflect number of trials. If the task is a multi-step per trial task, then the number of columns should reflect the number of steps, unless a custom likelihood function is used which does not require this.

actions: ndarray

Array of actions taken by subject. Number of rows should reflect number of trials. If the task is a multi-step per trial task, then the number of columns should reflect the number of steps, unless a custom likelihood function is used which does not require this.

rewards : ndarray

Array of rewards received by the subject. Number of rows should reflect number of trials. If there are multiple steps at which rewards are received, they should be stored in different columns, unless a custom likelihood funciton is used which does not require this.

Returns float

Log-posterior probability

Markov-Chain Monte-Carlo

class fitr.inference.mcmc.**MCMC** (*generative_model=None*, *name='FitrMCMCModel'*)
Uses Markov-Chain Monte-Carlo (via PyStan) to estimate models

Attributes

name	(str) Name of the model being fit
generative_model	(GenerativeModel object)

Methods

fit(self, data, chains=4, n_iterations=2000, warmup=None, thin=1, seed=None, init='random', sample_file=None, algorithm='NUTS', control=None, n_jobs=-1, compile_verbose=False, sampling_verbose=False)	Runs the MCMC Inference procedure with Stan
__initresults(self)	(Private) method to initialize MCMCFitResult object

fit(*data*, *chains=4*, *n_iterations=2000*, *warmup=None*, *thin=1*, *seed=None*, *init='random'*, *sample_file=None*, *algorithm='NUTS'*, *control=None*, *n_jobs=-1*, *compile_verbose=False*, *sampling_verbose=False*)

Runs the MCMC Inference procedure with Stan

Parameters data : dict

Subject level data

chains : int > 0

Number of chains in sampler

n_iter : int

How many iterations each chain should run (includes warmup)

warmup : int > 0, iter//2 by default

Number of warmup iterations.

thin : int > 0

Period for saving samples

seed : int or np.random.RandomState, optional

Positive integer to initialize random number generation

sample_file : str

File name specifying where samples for all parameters and other saved quantities will be written. If None, no samples will be written

algorithm : {'NUTS', 'HMC', 'Fixed_param'}, optional

Which of Stan's algorithms to implement

control : dict, optional

Dictionary of parameters to control sampler's behaviour (see PyStan documentation for details)

n_jobs : int, optional

Sample in parallel. If -1, all CPU cores are used. If 1, no parallel computing is used

compile_verbose : bool

Whether to print output from model compilation

sampling_verbose : bool

Whether to print intermediate output from model sampling

Returns ModelFitResult

Instance containing model fitting results

References

[R112]

Fitmodel: High Level Model-Fitting Wrapper

class `fitr.inference.fitmodel.FitModel` (*name='Anon Model'*, *loglik_func=None*, *params=None*, *generative_model=None*)

An object representing a model to be fit to behavioural data. This should be viewed as a high level wrapper for multiple potential model fitting algorithms which themselves can be run by using their respective classes.

Attributes

name	(str) Name of the model. We suggest identifying model based on free parameters.
loglik_func	(function) The log-likelihood function to be used to fit the data
params	(list) List of reinforcement learning parameter objects from the rlparams module.
generative_model	(GenerativeModel object) Object representing a generative model

Methods

fit(data, method='EM', c_limit=0.01)	Runs the specified model fitting algorithm with the given data.

`fit` (*data*, *method='EM'*, *c_limit=0.01*, *verbose=True*)

Runs model fitting

Parameters **data** : dict

Behavioural data.

method : {'EM', 'MLE', 'EmpiricalPriors', 'MCMC'}

The inference algorithm to use. Note that the data formats for 'MCMC' compared to the other methods is distinct, and should correspond appropriately to the method being employed

c_limit : float

Limit at which convergence of log-posterior probability is determined (only for methods 'EM' and 'EmpiricalPriors')

verbose : bool

Controls amount of printed output during model fitting

Returns **fitrfit** : object

Representation of the model fitting results

ModelFitResult

class `fitr.inference.modelfitresult.MCMCFitResult` (*method*, *nsubjects*, *nparams*, *name*)

Results of model fitting with MCMC

Attributes

name	(str) Model identifier. We suggest using free-parameters as identifiers
method	(str) Method employed in optimization.
nsubjects	(int) Number of subjects fitted.
nparams	(int) Number of free parameters in the fitted model.
params	(ndarray(shape=(nsubjects, nparams))) Array of parameter estimates
paramnames	(list) List of parameter names
stanfit :	Stan fit object
summary	(pandas.DataFrame) Summary of the MCMC fit results

Methods

get_paramestimates(self, FUN=np.mean)	Extracts parameter estimates
trace_plot(self, figsize=None, save_figure=False, filename='fitr-mcstan-traceplot.pdf')	Trace plot for fit results

get_paramestimates(*FUN=<Mock name='mock.median' id='140225606873888'>*)

Extracts parameter estimates

Parameters **FUN** : {numpy.mean, numpy.median}

make_summary()

Creates summary of Stan fitting results

trace_plot(*figsize=None*, *save_figure=False*, *filename='fitr-mcstan-traceplot.pdf'*)

Easy wrapper for Stan Traceplot

Parameters **figsize** : (optional) list [width in inches, height in inches]

Controls figure size

save_figure : bool

Whether to save the figure to disk

filename : str

The file name to be output

class fitr.inference.modelfitresult.**ModelFitResult**(*method*, *nsubjects*, *nparams*, *name=None*)

Class representing the results of a fitrmodel fitting.

Attributes

name	(str) Model identifier. We suggest using free-parameters as identifiers
method	(str) Method employed in optimization.
nsubjects	(int) Number of subjects fitted.
nparams	(int) Number of free parameters in the fitted model.
params	(ndarray(shape=(nsubjects, nparams))) Array of parameter estimates
paramnames	(list) List of parameter names

Methods

set_paramnames(params)	Sets names of RL parameters to the fitrfit object
plot_ae(actual, save_figure=False, filename='actual-estimate.pdf')	Plots estimated parameters against actual simulated parameters
summary_table(write_csv=False, filename='summary-table.csv', delimiter=',')	Writes a CSV file with summary statistics from the present model

ae_metrics(*actual*, *matches=None*)

Computes metrics summarizing the ability of the model to fit data generated from a known model

Parameters **matches** : list

List consisting of [rlparams object, column index in *actual*, column index in estimates]. Ensures comparisons are being made between the same parameters, particularly when the models have different numbers of free parameters.

Returns DataFrame

Including summary statistics of the parameter matching

plot_ae(*actual*, *save_figure=False*, *filename='actual-estimate.pdf'*)

Plots actual parameters (if provided) against estimates

Parameters **actual** : ndarray(shape=(nsubjects, nparams))

Array of actual parameters from a simulation

save_figure : bool

Whether to save the figure to disk

filename : str

The file name to be output

set_paramnames(*params*)

Sets the names of the RL parameters to the fitrfit object

Parameters **params** : list

List of parameters from the rlparams module

class fitr.inference.modelfitresult.**OptimizationFitResult**(*method*, *nsubjects*, *nparams*, *name*)

Results of model fitting with optimization methods

Attributes

name	(str) Model identifier. We suggest using free-parameters as identifiers
method	(str) Method employed in optimization.
nsub-jects	(int) Number of subjects fitted.
nparams	(int) Number of free parameters in the fitted model.
params	(ndarray(shape=(nsubjects, nparams))) Array of parameter estimates
param-names	(list) List of parameter names
errs	(ndarray(shape=(nsubjects, nparams))) Array of parameter estimate errors
nlog-post	(ndarray(shape=(nsubjects))) Subject level negative log-posterior probability
nloglik	(float) Subject level negative log-likelihood
LME	(float) Log-model evidence
BIC	(ndarray(shape=(nsubjects))) Subject-level Bayesian Information Criterion
AIC	(ndarray(shape=(nsubjects))) Subject-level Aikake Information Criterion
sum-mary	(DataFrame) Summary of means and standard deviations for each free parameter, as well as negative log-likelihood, log-model-evidence, BIC, and AIC for the model

Methods

plot_fit_ts(save_figure=False, filename='fit-stats.pdf') :	Plots the evolution of log-likelihood, log-model-evidence, AIC, and BIC over optimization iterations
param_hist(save_figure=False, filename='param-hist.pdf') :	Plots hitograms of parameters in the model
summary_table(write_csv=False, filename='summary-table.csv', delimiter=',')	Writes a CSV file with summary statistics from the present model

param_hist(*save_figure=False*, *filename='param-hist.pdf'*)

Plots histograms of the parameter estimates

Parameters **save_figure** : bool

Whether to save the figure to disk

filename : str

The file name to be output

plot_fit_ts(*save_figure=False*, *filename='fit-stats.pdf'*)

Plots the log-model-evidence, BIC, and AIC over optimization iterations

Parameters **save_figure** : bool

Whether to save the figure to disk

filename : str

The file name to be output

summary_table()

Generates a table summarizing the model-fitting results

Model Selection

Modules:

Aikake Information Criterion Model-Selection

class `fitr.model_selection.aic.`**`AIC`**(*model_fits*)

Model comparison with Aikake Information Criterion

Attributes

modelfits	(list) List of fitrfit objects from completed model fitting

Methods

run(self)	Runs model comparison by Aikake Information Criterion

`run`()

Runs model comparison by Aikake Information Criterion

Bayesian Information Criterion Model-Selection

class `fitr.model_selection.aic.`**`AIC`**(*model_fits*)

Model comparison with Aikake Information Criterion

Attributes

modelfits	(list) List of fitrfit objects from completed model fitting

Methods

run(self)	Runs model comparison by Aikake Information Criterion

`run`()

Runs model comparison by Aikake Information Criterion

Bayesian Model Selection

Functions for model selection/comparison.

References

Module Documentation

class `fitr.model_selection.bms.`**`BMS`**(*model_fits, c_limit=1e-99*)

Bayesian model selection [RigouxBMS].

Attributes

modelfits	(list) List of fitrfit objects from completed model fitting
nmodels	(int) Number of models to be compared
nsubjects: int	Number of subjects in the sample. (Must be equal across all fits).
c_limit	(float) Threshold at which to stop model comparison

Methods

run(self)	Runs model comparison by Bayesian Model Selection
dirichlet_exceedance(self, alpha)	Computes exceedance probabilities for a Dirichlet distribution
BOR(self, L, posterior, priors, C=None)	Computes Bayes Omnibus Risk (BOR)
FE(self, L, posterior, priors)	Derives free energy for current approximate posterior distribution
FE_null(self, L, options):	Derives the free energy of the 'null' hypothesis

BOR (*L*, *posterior*, *priors*, *C=None*)

Computes Bayes Omnibus Risk (BOR)

Parameters **L**

posterior

priors

C

Returns bor

Bayesian omnibus risk

Notes

As in [GershmanMfit].

FE (*L*, *posterior*, *priors*)

Derives free energy for current approximate posterior distribution [RigouxVBA].

Parameters **L**

Log model-evidence

posterior : dict

priors : dict

Returns F

Free energy of the current posterior

FE_null (*L*, *options*)

Derives the free energy of the 'null' hypothesis

Parameters **L**

Log model evidence

options : dict

Returns F0m

Evidence for the null (i.e. equal probabilities) over models

F0f

Evidence for the null (i.e. equal probabilities) over families

dirichlet_exceedance (*alpha*)

Computes exceedance probabilities for a Dirichlet distribution

Parameters alpha : float [0-1]

Returns xp

Exceedance probabilities

Notes

Implemented as in [GershmanMfit].

run ()

Runs Bayesian model selection algorithm

Returns ModelComparisonResult :

Object representing model comparison results

Cross Validation Methods

Functions for cross validation

class fitr.model_selection.cross_validation.**LOACV** (*cv_func*)

Look-one-ahead cross validation

Attributes

cv_func	(loacv function) A look-one-ahead cross-validation function from a Fitr model
results	(LookOneAheadCVResult) Stores results of the cross validation

Methods

run(params, data)	Runs the Look-One-Ahead cross validation

run (*params*, *data*)

Runs the Look-One-Ahead cross validation

Parameters params : ndarray(shape=(nsubjects, nparams))

Array of parameters

data : dict

Behavioural data in Fitr OptimizationData format

class fitr.model_selection.cross_validation.**LookOneAheadCVResult** (*params*)

Stores and manipulates results of a Look-One-Ahead cross-validation run

Attributes

nsubjects	(dict) Dictionary of
accuracy	(dict) Dictionary of accuracy values (overall and by subject)
raw	(dict) Dictionary

Methods

`accuracy_hist`([save_figure, filename, figsize])	Plots moving average of accuracy
`accuracy_maplot`([save_figure, filename, figsize])	Plots moving average of accuracy
`accuracy_param_scatter`([paramnames, ylim, ...])	Plots accuracy against parameter values.

accuracy_hist(*save_figure=False*, *filename='accuracy-hist.pdf'*, *figsize=None*)

Plots moving average of accuracy

Parameters **save_figure** : bool

Whether to save the plot

filename : str

Name of the file to which figure will be saved

figsize : (optional) tuple (width, height)

The size of the figure

accuracy_maplot(*save_figure=False*, *filename='accuracy-maplot.pdf'*, *figsize=None*)

Plots moving average of accuracy

Parameters **save_figure** : bool

Whether to save the plot

filename : str

Name of the file to which figure will be saved

figsize : (optional) tuple (width, height)

The size of the figure

accuracy_param_scatter(*paramnames=None*, *ylim=None*, *alpha=0.5*, *save_figure=False*, *filename='accuracy-param-scatter.pdf'*, *figsize=None*)

Plots accuracy against parameter values. Helpful to visually inspect the effects of various parameters on cross-validation accuracy

Parameters **paramnames** : (optional) list

List of parameter names in strings

ylim : (optional) tuple (min, max)

Y-axis limits

alpha : 0 < float < 1

Transparency of the plot points

save_figure : bool

Whether to save the plot

filename : str

Name of the file to which figure will be saved

figsize : (optional) tuple (width, height)

The size of the figure

Returns matplotlib.pyplot.figure

Model-Selection Result

class `fitr.model_selection.modelselectionresult.ModelSelectionResult` (*method*)

Object containing results of model selection

Attributes

modelnames	(list) List of strings labeling models
xp	(ndarray) Exceedance probabilities for each model
pxp	(ndarray) Protected exceedance probabilities for each model
BIC	(ndarray) Bayesian information criterion measures for each model
AIC	(ndarray) Aikake information criterion measures for each model

Methods

plot(self, statistic, save_figure=False, filename='modelselection-plot.pdf', figsize=(10, 10))	Plots the results of model selection (bars)

`plot` (*statistic*, *save_figure=False*, *filename='modelselection-plot.pdf'*, *figsize=(10, 10)*)

Plots the results of model selection (bars)

Parameters **statistic** : {'pxp', 'xp', 'BIC', 'AIC'}

Which statistic is desired for the bar plot

save_figure : bool

Whether to save the figure

filename : str

The desired filename for the plot (must end in appropriate extension)

figsize : tuple, default (10, 10)

Task Models and Data Objects

Modules:

Synthetic Data

class `fitr.models.synthetic_data.SyntheticData`

Object representing synthetic data

Attributes

data	(dict) Dictionary containing data formatted for fitr's model fitting tools (except MCMC via Stan)
data_mcmc	(dict) Dictionary containing task data formatted for use with MCMC via Stan
params	(ndarray(shape=(nsubjects X nparams))) Subject parameters
group-names	(list) Strings representing names of groups whose data are represented

Methods

append_group(self, data=SyntheticData)	
get_nparams(self)	Returns the number of parameters in the data
get_nsubjects(self)	Returns the number of subjects in the data
cumreward_param_plot(self, alpha=0.9)	Plots the cumulative reward against model parameters. Useful to determine the relationship between reward acquisition and model parameters for a given task.
plot_cumreward(self)	Plots the cumulative reward over time for each subject

append_group(*data*, *which='all'*)

Appends data from other groups to the SyntheticData object

Parameters **data** : SyntheticData object

all : {'all', 'opt', 'mcmc'}

Whether to append all data, optimization data only, or MCMC data

cumreward_param_plot(*alpha=0.9*, *save_figure=False*, *filename='cumreward-param-plot-sim.pdf'*)

Plots parameter values against cumulative reward

Parameters **save_figure** : bool

Whether to save the figure to disk

filename : str

The name of the file to which to save the figure

get_nparams()

Finds the number of parameters in the model

Returns int

get_nsubjects()

Finds the number of subjects in the data

Returns int

plot_cumreward(*save_figure=False*, *filename='cumreward-plot-sim.pdf'*)

Plots cumulative reward over time for each subject

Parameters **save_figure** : bool

Whether to save the figure to disk

filename : str

The name of the file to which to save the figure `fitr.models.synthetic_data.`**`combine_groups`**(*x*, *y*)

Combines synthetic data objects for multiple groups

Parameters **x** : SyntheticData

Data for first simulated group

y : SyntheticData

Data for second simulated group

Returns SyntheticData

Combined groups

Metrics

Modules:

Distance Metrics

Distance measures

Module Documentation

`fitr.metrics.distance.`**`likelihood_distance`**(*loglik_func*, *data*, *params*, *diff_metric='sq'*, *dist_metric='cosine'*, *verbose=False*)

Estimates the likelihood of the data from the i'th subject using the parameter estimates of the j'th subject, for all i and j, then computes the distance between subjects' likelihood difference vectors

Parameters **loglik_func** : function

The log-likelihood function to be used

data : dict

Data formatted for input into the log-likelihood function

params : ndarray(shape=(nsubjects, nparams))

Array of parameter estimates

diff_metric : {'sq', 'diff', 'abs'}

Which type of difference measure to compute, 'diff' is simple subtractive difference, whereas 'sq' and 'abs' are the squared and absolute differences, respectively

dist_metric : str (default='cosine')

The pairwise distance metric to use. Any option that can be passed into `sklearn.metrics.pairwise_distances` can work.

verbose : bool

Whether to print out progress

Returns ndarray(shape=(nsubjects, nsubjects))

`fitr.metrics.distance.`**`parameter_distance`**(*params*, *dist_metric='canberra'*, *scale='minmax'*, *return_scaled=False*)

Computes distances between subjects' respective parameter estimates

Parameters **params** : ndarray(shape=(nsubjects, nsubjects))

Array of parameter estimates

dist_metric : str (default='canberra')

Distance metric to be used. Can take any value acceptable by `sklearn.metrics.pairwise_distances`.

scale : {'minmax', 'standard', 'none'}

How to scale the parameters for distance computation

return_scaled : bool

Whether to return scaled parameters

Model-Evaluation Metrics

Metrics used during model evaluation and model comparison

Module Documentation

`fitr.metrics.model_evaluation.`**`AIC`**(*nparams*, *loglik*)

Calculates Aikake information criterion

Parameters **nparams** : int

Number of parameters in the model

loglik : float or ndarray(dtype=float)

Log-likelihood

Returns float or ndarray(dtype=float)

`fitr.metrics.model_evaluation.`**`BIC`**(*loglik*, *nparams*, *nsteps*)

Calculates Bayesian information criterion

Parameters **loglik** : float or ndarray(dtype=float)

Log-likelihood

nparams : int

Number of parameters in the model

nsteps : int

Number of time steps in the task

Returns float or ndarray(dtype=float)

`fitr.metrics.model_evaluation.`**`LME`**(*logpost*, *nparams*, *hessian*)

Laplace approximated log-model-evidence (LME)

Parameters **logpost** : float or ndarray(dtype=float)

Log-posterior probability

nparams : int

Number of parameters in the model

hessian : ndarray(size=(nparams, nparams))

Hessian computed from parameter optimization

Returns float or ndarray(dtype=float)

Plotting Functions

Modules:

Distance Metric Plots

Plotting functions for distance metrics

Module Documentation

`fitr.plotting.distance.`**`distance_hist`**(*X*, *group_labels*, *xlab='Distance'*, *ylab=''*, *normed=1*, *alpha=0.5*, *save_figure=False*, *figsize=None*, *figname='distance-hist.pdf'*)

Creates a histogram of within- and between-group distances.

Parameters **group_labels** : ndarray(size=n_labels)

Vector of group labels for each participant represented

xlab : str

X-axis label

ylab : str

Y-axis label

normed : 0 or 1 (default=1)

Whether the histogram should be normalized

alpha : float on interval (0, 1)

Transparency of scatterplot points

save_figure : bool

Whether to save the figure

figsize : (optional) list

Controls figure size

figname : str

The name under which the plot should be saved

`fitr.plotting.distance.`**`distance_scatter`**(*X*, *Y*, *group_labels=None*, *xlab=''*, *ylab=''*, *alpha=0.5*, *save_figure=False*, *figsize=None*, *figname='distance-scatter.pdf'*)

Creates a scatterplot between two distance metrics, demonstrating group separation, if any.

Parameters **group_labels** : (optional)

xlab : str

X-axis label

ylab : str

Y-axis label

alpha : float on interval (0, 1)

Transparency of scatterplot points

save_figure : bool

Whether to save the figure

figsize : (optional) list

Controls figure size

figname : str

The name under which the plot should be saved

ParamPlots

Various plotting functions for parameter estimates

Module Documentation

`fitr.plotting.paramplots.`**`param_scatter`**(*X, Y, paramnames=None, xlabel='Parameter Value', ylabel='y value', ylim=None, alpha=0.5, figsize=None, save_figure=False, filename='param-scatter.pdf'*)

Plots a value against parameter estimates for each parameter

Parameters **X** : ndarray(shape=(nsubjects, nparams))

Parameter array

Y : ndarray(shape=nsubjects)

Value to be plotted against parameters

paramnames : (optional) list

Parameter names (will be the title for each plot)

xlabel : str

Label for x axis

ylabel : str

Label for y axis

ylim : (optional) tuple (min, max)

Y-axis limits

alpha : 0 < float < 1

Transparency of scatter points

figsize : (optional) tuple (width, height)

save_figure : bool

Whether to save the plot

filename : str

Path to which to plot the figure

Returns matplotlib.pyplot.figure

RLParams

Module containing commonly used reinforcement learning parameter objects.

Module Documentation

class `fitr.rlparams.ChoiceRandomness`(*name='Choice Randomness'*, *rng='pos'*, *mean=4*, *sd=1*)
An choice randomness parameter object

Attributes

name	(str) Name of the parameter. To be used for plots and so forth.
rng	({'unit', 'pos', 'neg', 'unc'}) The domain over which the parameter lies (unit=[0,1], pos=[0,+Inf], neg=[-Inf,0], unc=[-Inf, +Inf])
dist	(scipy.stats.gamma distribution)

Methods

sample(size=1)	Samples from the parameter's distribution

class `fitr.rlparams.EligibilityTrace`(*name='Eligibility Trace'*, *rng='unit'*, *mean=0.5*, *sd=0.27*)
An eligibility trace parameter object.

Attributes

name	(str) Name of the parameter. To be used for plots and so forth.
rng	({'unit', 'pos', 'neg', 'unc'}) The domain over which the parameter lies (unit=[0,1], pos=[0,+Inf], neg=[-Inf,0], unc=[-Inf, +Inf])
dist	(scipy.stats.beta distribution)

Methods

sample(size=1)	Samples from the parameter's distribution

class `fitr.rlparams.LearningRate`(*name='Learning Rate'*, *rng='unit'*, *mean=0.5*, *sd=0.27*)
A learning rate object.

Attributes

name	(str) Name of the parameter. To be used for plots and so forth.
rng	({'unit', 'pos', 'neg', 'unc'}) The domain over which the parameter lies (unit=[0,1], pos=[0,+Inf], neg=[-Inf,0], unc=[-Inf, +Inf])
dist	(scipy.stats.beta distribution)

Methods

sample(size=1)	Samples from the parameter's distribution

class `fitr.rlparams.`**`MBMF_Balance`**(*name='Model-Based Control Weight'*, *rng='unit'*, *mean=0.5*, *sd=0.27*)

An object representing the parameter that balances model-based and model-free control.

Attributes

name	(str) Name of the parameter. To be used for plots and so forth.
rng	({'unit', 'pos', 'neg', 'unc'}) The domain over which the parameter lies (unit=[0,1], pos=[0,+Inf], neg=[-Inf,0], unc=[-Inf, +Inf])
dist	(scipy.stats.beta distribution)

Methods

sample(size=1)	Samples from the parameter's distribution

class `fitr.rlparams.`**`Param`**(*name=None*, *rng=None*)

A base parameter object that can be used to generate new parameters.

Attributes

name	(str) Name of the parameter. To be used for plots and so forth.
rng	({'unit', 'pos', 'neg', 'unc'}) The domain over which the parameter lies (unit=[0,1], pos=[0,+Inf], neg=[-Inf,0], unc=[-Inf, +Inf])

Methods

sample(size=1)	Samples from the parameter's distribution

`convert_meansd`(*mean*, *sd*, *dist*)

Converts mean and standard deviation to other distribution parameters.

Parameters **mean** : float

Mean value for distribution (must lie within region of support)

sd : float

Standard deviation for distribution

dist : {'beta', 'gamma'}

Target distribution

Notes

Currently, only the gamma and beta distributions are supported for this function.

The Beta distribution has two shape parameters $\{\alpha, \beta\} > 0$. Using the mean μ and the standard deviation σ, the α parameter can be calculated as

$$\alpha = (\frac{1-\mu}{\sigma^2} - \frac{1}{\mu})\mu^2$$

and the β parameter as

$$\beta = \alpha(\frac{1}{\mu} - 1)$$

Note that for the Beta distribution to be defined this way, the following constraint must hold for the mean, $0 < \mu < 1$, and the following for the variance, $0 < \sigma^2 \leq \mu - \mu^2$.

For the Gamma distribution, we have a shape parameter $\kappa > 0$ and a scale parameter θ. These can be calculated using the mean μ and standard deviation σ as

$$\theta = \frac{\sigma^2}{\mu}$$

and

$$\kappa = \frac{\mu^2}{\sigma^2}$$

plot_pdf(*xlim=None*, *figsize=None*, *save_figure=False*, *filename='parameter-pdf.pdf'*)

Plots the probability density function of this parameter

Parameters **xlim** : (optional) list of lower and upper bounds of x axis

figsize : (optional) list defining plot dimensions

save_figure : bool

Whether to save the figure at function call

filename : str

The name of the file at which to save the figure

sample(*size=1*)

Samples from the parameter's distribution

Parameters **size** : int

Number of samples to draw

Returns ndarray

class fitr.rlparams.**Perseveration**(*name='Perseveration'*, *rng='unc'*, *mean=0.0*, *sd=0.1*)

An perseveration parameter object

Attributes

name	(str) Name of the parameter. To be used for plots and so forth.
rng	({'unit', 'pos', 'neg', 'unc'}) The domain over which the parameter lies (unit=[0,1], pos=[0,+Inf], neg=[-Inf,0], unc=[-Inf, +Inf])
dist	(scipy.stats.norm distribution)

Methods

sample(size=1)	Samples from the parameter's distribution

class `fitr.rlparams.RewardSensitivity`(*name='Reward Sensitivity'*, *rng='unit'*, *mean=0.5*, *sd=0.27*)

A reward sensitivity object.

Attributes

name	(str) Name of the parameter. To be used for plots and so forth.
rng	({'unit', 'pos', 'neg', 'unc'}) The domain over which the parameter lies (unit=[0,1], pos=[0,+Inf], neg=[-Inf,0], unc=[-Inf, +Inf])
dist	(scipy.stats.beta distribution)

Methods

sample(size=1)	Samples from the parameter's distribution

Utils

Module containing functions that are used across Fitr modules

References

Module Documentation

`fitr.utils.action`(*x*)

Selects an action based on state-action values

Parameters **x** : ndarray

Array of action values (scaled by inverse softmax temperature).

Returns int

The index corresponding to the selected action

Notes

This function computes the softmax probability for each action in the input array, and subsequently samples from a multinomial distribution parameterized by the results of the softmax computation. Finally, it returns the index where the value is equal to 1 (i.e. which action was selected).

`fitr.utils.logsumexp`(*x*)

Numerically stable logsumexp.

Parameters **x** : ndarray(shape=(nactions))

Returns float

Notes

The numerically stable log-sum-exp is computed as follows:

$$\max X + \log \sum_X e^{X - \max X}$$

`fitr.utils.softmax(x)`
Computes numerically stable softmax

Parameters **x** : ndarray(shape=(nactions))

Returns ndarray(shape=(nactions))

Softmax probabilities for each action

`fitr.utils.trans_UC(values_U, rng)`
Transforms parameters from unconstrained to constrained space

Parameters **values_U** : ndarray

Parameter values

rng : {'unit', 'pos', 'half', 'all_unc'}

The constrained range of the parameter

Returns ndarray(shape=(nparams))

Notes

This code was taken from that published along with [Akam2015].

CHAPTER 2

Overview

In decision-making tasks, it is often of interest to understand the psychological mechanisms by which a subject makes choices. One way to approach this problem is by specifying a reinforcement learning (RL) model that describes those mechanisms, and then fitting it (by tuning its free parameters) to subjects' actual behavioural data. There are many ways to perform this analysis, which for the most part might be inaccessible to researchers without extensive mathematical training. We are building Fitr as an attempt to make this process simpler to implement.

For researchers interested in the computational aspects of decision-making research, Fitr also aims to offer pre-packaged routines to streamline the study & development of new tasks, models, and model-fitting/selection procedures.

CHAPTER 3

Goals

1. **Offer a free and open platform upon which researchers can easily**
    - Build and validate behavioural tasks
    - Build and validate behavioural models
    - Develop new model-fitting procedures
    - Fit models to behavioural data from subjects *in vivo*
2. Implement the state of the art methods for behavioural modelling studies in computational psychiatry
3. Integrate well with tools for collecting behavioural data
4. Integrate well with tools for collecting neurophysiological data

CHAPTER 4

Guiding Principles

1. **Accessibility**
    - Fitr should be open-source, free, and not dependent on commercial tools
2. **Parsimony**
    - Build tools that can turn data into results with minimal coding
3. **Modularity**
    - **Build modules, classes, and functions in a way that facilitates the computational modelling workflow as it applies**
        - Developing and validating tasks
        - Developing and validating models
        - Fitting/selecting models using data from human subjects
4. **Flexibility**
    - There are many ways to fit a model. Users should be able to easily test multiple models and multiple fitting methods without much additional code.
    - Allow users to easily integrate their own code, where desired
    - Facilitate development of "Pipelines" for Computational Psychiatry research
5. **Don't re-invent the wheel (unless you have to)**
    - If excellent open-source tools exist, don't rebuild them. Rather, make it possible for users to integrate them easily into their workflow
    - Always give credit wherever credit is due
6. **Build for communication and reproducibility**
    - Make it easy for researchers to generate the high-quality tables and plots necessary to communicate the results of their modelling studies.
    - Make it easy to reproduce results generated by Fitr pipelines

CHAPTER 5

What we're working on

- Adding new tasks and new models
- Writing more tutorials
- End-to-end model-fitting and model-selection
- Improving existing model-fitting algorithms
- Adding new model-fitting algorithms (Variational Bayes)
- Model-based neuroimaging

CHAPTER 6

Citing Fitr

Let us know if you publish a paper using Fitr and we will post it here. If you use Fitr in your work, please cite it so that we can (A) know how people have been using it, and (B) support further funding of our work.

- Abraham Nunes, Alexander Rudiuk, & Thomas Trappenberg. (2017). Fitr: A Toolbox for Computational Psychiatry Research. Zenodo. http://doi.org/10.5281/zenodo.439989

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Bibliography

[Daw2006] Daw, N.D. et al. (2006) Cortical substrates for exploratory decisions in humans. Nature 441, 876–879

[Daw2011] Daw, N.D. et al. (2011) Model-based influences on humans' choices and striatal prediction errors. Neuron 69, 1204–1215

[Gershman2016] Gershman, S.J. (2016) Empirical priors for reinforcement learning models. J. Math. Psychol. 71, 1–6

[Huys2011] Huys, Q. J. M., et al. (2011). Disentangling the roles of approach, activation and valence in instrumental and pavlovian responding. PLoS Computational Biology, 7(4).

[StanDevs] Stan Development Team. 2016. PyStan: the Python interface to Stan, Version 2.14.0.0. http://mc-stan.org

[Rigoux2014] Rigoux, L. et al. (2014) Bayesian model selection for group studies - Revisited. Neuroimage 84, 971–985

[R112] PyStan API documentation (https://pystan.readthedocs.io)

[RigouxVBA3] Rigoux L., Daunizeau J. VBA Toolbox

[RigouxBMS3] Rigoux, L. et al. (2014) Neuroimage 84, 971–985

[GershmanMfit3] Samuel Gershman's mfit package (on GitHub)

[Akam20156] Akam, T. et al. (2015) Simple Plans or Sophisticated Habits? State, Transition and Learning Interactions in the Two-Step Task. PLoS Comput. Biol. 11, 1–25

Python Module Index

f

fitr.inference.em, 4
fitr.inference.empiricalpriors, 7
fitr.inference.fitmodel, 10
fitr.inference.mcmc, 8
fitr.inference.mle, 3
fitr.inference.modelfitresult, 10
fitr.metrics.distance, 20
fitr.metrics.model_evaluation, 21
fitr.model_selection.aic, 14
fitr.model_selection.bms, 14
fitr.model_selection.cross_validation, 16
fitr.model_selection.modelselectionresult, 18
fitr.models.synthetic_data, 18
fitr.plotting.distance, 22
fitr.plotting.paramplots, 23
fitr.rlparams, 24
fitr.utils, 27

Index

A

accuracy_hist() (fitr.model_selection.cross_validation.LookOneAheadCVResult method), 17
accuracy_maplot() (fitr.model_selection.cross_validation.LookOneAheadCVResult method), 17
accuracy_param_scatter() (fitr.model_selection.cross_validation.LookOneAheadCVResult method), 17
action() (in module fitr.utils), 27
ae_metrics() (fitr.inference.modelfitresult.ModelFitResult method), 12
AIC (class in fitr.model_selection.aic), 14
AIC() (in module fitr.metrics.model_evaluation), 21
append_group() (fitr.models.synthetic_data.SyntheticData method), 19

B

BIC() (in module fitr.metrics.model_evaluation), 21
BMS (class in fitr.model_selection.bms), 14
BOR() (fitr.model_selection.bms.BMS method), 15

C

ChoiceRandomness (class in fitr.rlparams), 24
combine_groups() (in module fitr.models.synthetic_data), 19
convert_meansd() (fitr.rlparams.Param method), 25
cumreward_param_plot() (fitr.models.synthetic_data.SyntheticData method), 19

D

dirichlet_exceedance() (fitr.model_selection.bms.BMS method), 16
distance_hist() (in module fitr.plotting.distance), 22
distance_scatter() (in module fitr.plotting.distance), 22

E

EligibilityTrace (class in fitr.rlparams), 24
EM (class in fitr.inference.em), 4
EmpiricalPriors (class in fitr.inference.empiricalpriors), 7

F

FE() (fitr.model_selection.bms.BMS method), 15
FE_null() (fitr.model_selection.bms.BMS method), 15
fit() (fitr.inference.em.EM method), 5
fit() (fitr.inference.empiricalpriors.EmpiricalPriors method), 7
fit() (fitr.inference.fitmodel.FitModel method), 10
fit() (fitr.inference.mcmc.MCMC method), 8
fit() (fitr.inference.mle.MLE method), 4
FitModel (class in fitr.inference.fitmodel), 10
fitr.inference.em (module), 4
fitr.inference.empiricalpriors (module), 7
fitr.inference.fitmodel (module), 10
fitr.inference.mcmc (module), 8
fitr.inference.mle (module), 3
fitr.inference.modelfitresult (module), 10
fitr.metrics.distance (module), 20
fitr.metrics.model_evaluation (module), 21
fitr.model_selection.aic (module), 14
fitr.model_selection.bms (module), 14
fitr.model_selection.cross_validation (module), 16
fitr.model_selection.modelselectionresult (module), 18
fitr.models.synthetic_data (module), 18
fitr.plotting.distance (module), 22
fitr.plotting.paramplots (module), 23
fitr.rlparams (module), 24
fitr.utils (module), 27

G

get_nparams() (fitr.models.synthetic_data.SyntheticData method), 19
get_nsubjects() (fitr.models.synthetic_data.SyntheticData method), 19
get_paramestimates() (fitr.inference.modelfitresult.MCMCFitResult method), 11
group_level_estimate() (fitr.inference.em.EM method), 6

I

initialize_opt() (fitr.inference.em.EM method), 6

L

LearningRate (class in fitr.rlparams), 24
likelihood_distance() (in module fitr.metrics.distance), 20
LME() (in module fitr.metrics.model_evaluation), 21
LOACV (class in fitr.model_selection.cross_validation), 16
logposterior() (fitr.inference.em.EM method), 6
logposterior() (fitr.inference.empiricalpriors.EmpiricalPriors method), 8
logsumexp() (in module fitr.utils), 27
LookOneAheadCVResult (class in fitr.model_selection.cross_validation), 16

M

make_summary() (fitr.inference.modelfitresult.MCMCFitResult method), 11
MBMF_Balance (class in fitr.rlparams), 25
MCMC (class in fitr.inference.mcmc), 8
MCMCFitResult (class in fitr.inference.modelfitresult), 10
MLE (class in fitr.inference.mle), 3
ModelFitResult (class in fitr.inference.modelfitresult), 11
ModelSelectionResult (class in fitr.model_selection.modelselectionresult), 18

O

OptimizationFitResult (class in fitr.inference.modelfitresult), 12

P

Param (class in fitr.rlparams), 25
param_hist() (fitr.inference.modelfitresult.OptimizationFitResult method), 13
param_scatter() (in module fitr.plotting.paramplots), 23
parameter_distance() (in module fitr.metrics.distance), 20
Perseveration (class in fitr.rlparams), 26
plot() (fitr.model_selection.modelselectionresult.ModelSelectionResult method), 18
plot_ae() (fitr.inference.modelfitresult.ModelFitResult method), 12
plot_cumreward() (fitr.models.synthetic_data.SyntheticData method), 19
plot_fit_ts() (fitr.inference.modelfitresult.OptimizationFitResult method), 13
plot_pdf() (fitr.rlparams.Param method), 26

R

RewardSensitivity (class in fitr.rlparams), 27
run() (fitr.model_selection.aic.AIC method), 14
run() (fitr.model_selection.bms.BMS method), 16
run() (fitr.model_selection.cross_validation.LOACV method), 16

S

sample() (fitr.rlparams.Param method), 26
set_paramnames() (fitr.inference.modelfitresult.ModelFitResult method), 12
softmax() (in module fitr.utils), 28
summary_table() (fitr.inference.modelfitresult.OptimizationFitResult method), 13
SyntheticData (class in fitr.models.synthetic_data), 18

T

trace_plot() (fitr.inference.modelfitresult.MCMCFitResult method), 11
trans_UC() (in module fitr.utils), 28