pyHPDM Documentation

Release 0.9.9

Dominic Hunt

Mar 22, 2020

Contents

python Human Probabilistic Decision-Modelling (pyHPDM) is a framework for modelling and fitting the responses of people to probabilistic decision making tasks.

CHAPTER 1

Prerequisites

This code has been tested using `Python 2.7`. Apart from the standard Python libraries it also depends on the SciPy librariesand a few others listed in `requirements.txt`. For those installing Python for the first time I would recommend the Anaconda Python distribution.

CHAPTER 2

Installation

For now this is just Python code that you download and use, not a package.

CHAPTER 3

Usage

The framework has until now either been run with a run script or live in a command-line (or jupyter notebook).

A task simulation can be simply created by running `simulation.simulation()`. Equally, for fitting participant data, the function is `dataFitting.data_fitting`. For now, no example data has been provided.

More complex example running scripts can be found in `./runScripts/`. Here, a number of scripts have been created as templates: `runScript_sim.py` for simulating the `probSelect` task and `runScript_fit.py` for fitting the data generated from `runScript_sim.py`. A visual display of the interactions in one of these scripts will soon be created.

A new method of passing in the fitting or simulation configuration is to use a YAML configuration file. This is done, for both simulations and data fitting, using the function `start.run_script` For example, to run the YAML configuration equivalent to the `runScript_sim.py` from a command line would be `:start.run_script('./runScripts/runScripts_sim.yaml')`.

CHAPTER 4

Testing

Testing is done using pytest.

CHAPTER 5

License

This project is licenced under the MIT license.

CHAPTER 6

Documentation

The documentation can be found on readthedocs or in `./doc/_build/html`, with the top level file being `index.html`

To update the documentation you will need to install Sphinx and a set of extensions. The list of extensions can be found in `./doc/conf.py`. To update the documentation follow the instruction in `./doc/readme.md`

Contents:

6.1 simulation module

6.1.1 simulation Module

Author Dominic Hunt

6.1.1.1 Functions

csv_model_simulation(modelData, simID, ...)	Saves the fitting data to a CSV file
log_simulation_parameters(task_parameters, ...)	Writes to the log the description and the label of the task and model
record_simulation(file_name_generator, ...)	Records the data from an task-model run.
run([task_name, task_changing_properties, ...])	A framework for letting models interact with tasks and record the data

csv_model_simulation

simulation.**csv_model_simulation**(*modelData*, *simID*, *file_name_generator*)

Saves the fitting data to a CSV file

Parameters

- **modelData** (*dict*) – The data from the model
- **simID** (*string*) – The identifier for the simulation

- **file_name_generator** (*function*) – Creates a new file with the name <handle> and the extension <extension>. It takes two string parameters: (`handle`, `extension`) and returns one `fileName` string

log_simulation_parameters

`simulation.`**log_simulation_parameters**(*task_parameters*, *model_parameters*, *simID*)

Writes to the log the description and the label of the task and model

Parameters

- **task_parameters** (*dict*) – The task parameters
- **model_parameters** (*dict*) – The model parameters
- **simID** (*string*) – The identifier for each simulation.

See also:

recordSimParams() Records these parameters for later use

record_simulation

`simulation.`**record_simulation**(*file_name_generator*, *task_data*, *model_data*, *simID*, *pickle=False*)

Records the data from an task-model run. Creates a pickled version

Parameters

- **file_name_generator** (*function*) – Creates a new file with the name <handle> and the extension <extension>. It takes two string parameters: (`handle`, `extension`) and returns one `fileName` string
- **task_data** (*dict*) – The data from the task
- **model_data** (*dict*) – The data from the model
- **simID** (*basestring*) – The label identifying the simulation
- **pickle** (*bool, optional*) – If true the data for each model, task and participant is recorded. Default is `False`

See also:

pickleLog() records the picked data

run

`simulation.`**run**(*task_name=u'Basic'*, *task_changing_properties=None*, *task_constant_properties=None*, *model_name=u'QLearn'*, *model_changing_properties=None*, *model_constant_properties=None*, *model_changing_properties_repetition=1*, *label=None*, *config_file=None*, *output_path=None*, *pickle=False*, *min_log_level=u'INFO'*, *numpy_error_level=u'log'*)

A framework for letting models interact with tasks and record the data

Parameters

- **task_name** (*string*) – The name of the file where a tasks.taskTemplate.Task class can be found. Default `Basic`

- **task_changing_properties** (*dictionary of floats or lists of floats*) – Parameters are the options that you are or are likely to change across task instances. When a parameter contains a list, an instance of the task will be created for every combination of this parameter with all the others. Default `None`
- **task_constant_properties** (*dictionary of float, string or binary valued elements*) – These contain all the the task options that describe the task being studied but do not vary across task instances. Default `None`
- **model_name** (*string*) – The name of the file where a model.modelTemplate.Model class can be found. Default `QLearn`
- **model_changing_properties** (*dictionary containing floats or lists of floats, optional*) – Parameters are the options that you are or are likely to change across model instances. When a parameter contains a list, an instance of the model will be created for every combination of this parameter with all the others. Default `None`
- **model_constant_properties** (*dictionary of float, string or binary valued elements, optional*) – These contain all the the model options that define the version of the model being studied. Default `None`
- **model_changing_properties_repetition** (*int, optional*) – The number of times each parameter combination is repeated.
- **config_file** (*string, optional*) – The file name and path of a `.yaml` configuration file. Overrides all other parameters if found. Default `None`
- **output_path** (*string, optional*) – The path that will be used for the run output. Default `None`
- **pickle** (*bool, optional*) – If true the data for each model, task and participant is recorded. Default is `False`
- **label** (*string, optional*) – The label for the simulation. Default `None`, which means nothing will be saved
- **min_log_level** (*basestring, optional*) – Defines the level of the log from (`DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`). Default `INFO`
- **numpy_error_level** (*{'log', 'raise'}*) – Defines the response to numpy errors. Default `log`. See numpy.seterr

See also:

tasks.taskTemplate(), *model.modelTemplate()*

Author Dominic Hunt

simulation.**csv_model_simulation**(*modelData*, *simID*, *file_name_generator*)

Saves the fitting data to a CSV file

Parameters

- **modelData** (*dict*) – The data from the model
- **simID** (*string*) – The identifier for the simulation
- **file_name_generator** (*function*) – Creates a new file with the name <handle> and the extension <extension>. It takes two string parameters: (`handle`, `extension`) and returns one `fileName` string

simulation.**log_simulation_parameters**(*task_parameters*, *model_parameters*, *simID*)

Writes to the log the description and the label of the task and model

Parameters

- **task_parameters** (*dict*) – The task parameters
- **model_parameters** (*dict*) – The model parameters

- **simID** (*string*) – The identifier for each simulation.

See also:

recordSimParams() Records these parameters for later use

simulation.**record_simulation**(*file_name_generator*, *task_data*, *model_data*, *simID*, *pickle=False*)

Records the data from an task-model run. Creates a pickled version

Parameters

- **file_name_generator** (*function*) – Creates a new file with the name <handle> and the extension <extension>. It takes two string parameters: (`handle`, `extension`) and returns one `fileName` string
- **task_data** (*dict*) – The data from the task
- **model_data** (*dict*) – The data from the model
- **simID** (*basestring*) – The label identifying the simulation
- **pickle** (*bool, optional*) – If true the data for each model, task and participant is recorded. Default is `False`

See also:

pickleLog() records the picked data

simulation.**run**(*task_name=u'Basic'*, *task_changing_properties=None*, *task_constant_properties=None*, *model_name=u'QLearn'*, *model_changing_properties=None*, *model_constant_properties=None*, *model_changing_properties_repetition=1*, *label=None*, *config_file=None*, *output_path=None*, *pickle=False*, *min_log_level=u'INFO'*, *numpy_error_level=u'log'*)

A framework for letting models interact with tasks and record the data

Parameters

- **task_name** (*string*) – The name of the file where a tasks.taskTemplate.Task class can be found. Default `Basic`
- **task_changing_properties** (*dictionary of floats or lists of floats*) – Parameters are the options that you are or are likely to change across task instances. When a parameter contains a list, an instance of the task will be created for every combination of this parameter with all the others. Default `None`
- **task_constant_properties** (*dictionary of float, string or binary valued elements*) – These contain all the the task options that describe the task being studied but do not vary across task instances. Default `None`
- **model_name** (*string*) – The name of the file where a model.modelTemplate.Model class can be found. Default `QLearn`
- **model_changing_properties** (*dictionary containing floats or lists of floats, optional*) – Parameters are the options that you are or are likely to change across model instances. When a parameter contains a list, an instance of the model will be created for every combination of this parameter with all the others. Default `None`
- **model_constant_properties** (*dictionary of float, string or binary valued elements, optional*) – These contain all the the model options that define the version of the model being studied. Default `None`
- **model_changing_properties_repetition** (*int, optional*) – The number of times each parameter combination is repeated.

- **config_file** (*string, optional*) – The file name and path of a `.yaml` configuration file. Overrides all other parameters if found. Default `None`
- **output_path** (*string, optional*) – The path that will be used for the run output. Default `None`
- **pickle** (*bool, optional*) – If true the data for each model, task and participant is recorded. Default is `False`
- **label** (*string, optional*) – The label for the simulation. Default `None`, which means nothing will be saved
- **min_log_level** (*basestring, optional*) – Defines the level of the log from (`DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`). Default `INFO`
- **numpy_error_level** (*{'log', 'raise'}*) – Defines the response to numpy errors. Default `log`. See numpy.seterr

See also:

tasks.taskTemplate(), *model.modelTemplate()*

6.2 dataFitting module

6.2.1 dataFitting Module

Author Dominic Hunt

6.2.1.1 Functions

fit_record(participant_fits, file_name_generator)	Returns the participant fits summary as a csv file
log_fitting_parameters(fit_info)	Records and outputs to the log the parameters associated with the fitting algorithms
log_model_fitted_parameters(...)	Logs the model and task parameters that used as initial fitting conditions
log_model_fitting_parameters(model, ...)	Logs the model and task parameters that used as initial fitting conditions
record_fitting(fitting_data, label, ... [, ...])	Records formatted versions of the fitting data
record_participant_fit(participant, ... [, ...])	Record the data relevant to the participant fitting
run([data_folder, data_format, ...])	A framework for fitting models to data for tasks, along with recording the data associated with the fits.
xlsx_fitting_data(fitting_data, label, ...)	Saves the fitting data to an XLSX file

fit_record

dataFitting.**fit_record**(*participant_fits*, *file_name_generator*)

Returns the participant fits summary as a csv file

Parameters

- **participant_fits** (*dict*) – A summary of the recovered parameters
- **file_name_generator** (*function*) – Creates a new file with the name <handle> and the extension <extension>. It takes two string parameters: (`handle`, `extension`) and returns one `fileName` string

log_fitting_parameters

dataFitting.**log_fitting_parameters**(*fit_info*)

Records and outputs to the log the parameters associated with the fitting algorithms

Parameters **fit_info** (*dict*) – The details of the fitting

log_model_fitted_parameters

dataFitting.**log_model_fitted_parameters**(*model_fit_variables*, *model_parameters*, *fit_quality*, *participant_name*)

Logs the model and task parameters that used as initial fitting conditions

Parameters

- **model_fit_variables** (*dict*) – The model parameters that have been fitted over and varied.
- **model_parameters** (*dict*) – The model parameters for the fitted model
- **fit_quality** (*float*) – The value of goodness of fit
- **participant_name** (*int or string*) – The identifier for each participant

log_model_fitting_parameters

dataFitting.**log_model_fitting_parameters**(*model*, *model_fit_variables*, *model_other_args*)

Logs the model and task parameters that used as initial fitting conditions

Parameters

- **model** (*string*) – The name of the model
- **model_fit_variables** (*dict*) – The model parameters that will be fitted over and varied.
- **model_other_args** (*dict*) – The other parameters used in the model whose attributes have been modified by the user

record_fitting

dataFitting.**record_fitting**(*fitting_data*, *label*, *participant*, *participant_model_variables*, *participant_fits*, *file_name_generator*, *save_fitting_progress=False*)

Records formatted versions of the fitting data

Parameters

- **fitting_data** (*dict, optional*) – Dictionary of details of the different fits, including an ordered dictionary containing the parameter values tested, in the order they were tested, and a list of the fit qualities of these parameters.
- **label** (*basestring*) – The label used to identify the fit in the file names
- **participant** (*dict*) – The participant data
- **participant_model_variables** (*dict of string*) – A dictionary of model settings whose values should vary from participant to participant based on the values found in the imported participant data files. The key is the label given in the participant data file, as a string, and the value is the associated label in the model, also as a string.

- **participant_fits** (*defaultdict of lists*) – A dictionary to be filled with the summary of the participant fits
- **file_name_generator** (*function*) – Creates a new file with the name <handle> and the extension <extension>. It takes two string parameters: (`handle`, `extension`) and returns one `fileName` string
- **save_fitting_progress** (*bool, optional*) – Specifies if the results from each iteration of the fitting process should be returned. Default `False`

Returns **participant_fits** – A dictionary to be filled with the summary of the previous and current participant fits

Return type defaultdict of lists

record_participant_fit

dataFitting.**record_participant_fit**(*participant*, *part_name*, *model_data*, *model_name*, *fitting_data*, *partModelVars*, *participantFits*, *fileNameGen=None*, *pickleData=False*, *saveFittingProgress=False*, *expData=None*)

Record the data relevant to the participant fitting

Parameters

- **participant** (*dict*) – The participant data
- **part_name** (*int or string*) – The identifier for each participant
- **model_data** (*dict*) – The data from the model
- **model_name** (*basestring*) – The label given to the model
- **fitting_data** (*dict*) – Dictionary of details of the different fits, including an ordered dictionary containing the parameter values tested, in the order they were tested, and a list of the fit qualities of these parameters
- **partModelVars** (*dict of string*) – A dictionary of model settings whose values should vary from participant to participant based on the values found in the imported participant data files. The key is the label given in the participant data file, as a string, and the value is the associated label in the model, also as a string.
- **participantFits** (*defaultdict of lists*) – A dictionary to be filled with the summary of the participant fits
- **fileNameGen** (*function or None*) – Creates a new file with the name <handle> and the extension <extension>. It takes two string parameters: (`handle`, `extension`) and returns one `fileName` string. Default `None`
- **pickleData** (*bool, optional*) – If true the data for each model, task and participant is recorded. Default is `False`
- **saveFittingProgress** (*bool, optional*) – Specifies if the results from each iteration of the fitting process should be returned. Default `False`
- **expData** (*dict, optional*) – The data from the task. Default `None`

Returns **participantFits** – A dictionary to be filled with the summary of the previous and current participant fits

Return type defaultdict of lists

See also:

outputting.pickleLog() records the picked data

run

dataFitting.**run**(*data_folder=u'./'*, *data_format=u'csv'*, *data_file_filter=None*, *data_file_terminal_ID=True*, *data_read_options=None*, *data_split_by=None*, *data_group_by=None*, *data_extra_processing=None*, *model_name=u'QLearn'*, *model_changing_properties=None*, *model_constant_properties=None*, *participantID=u'Name'*, *participant_choices=u'Actions'*, *participant_rewards=u'Rewards'*, *model_fit_value=u'ActionProb'*, *fit_subset=None*, *task_stimuli=None*, *participant_action_options=None*, *fit_method=u'Evolutionary'*, *fit_method_args=None*, *fit_measure=u'-loge'*, *fit_measure_args=None*, *fit_extra_measures=None*, *participant_varying_model_parameters=None*, *label=None*, *save_fitting_progress=False*, *config_file=None*, *output_path=None*, *pickle=False*, *boundary_excess_cost_function=None*, *min_log_level=u'INFO'*, *numpy_error_level=u'log'*, *fit_float_error_response_value=1e-100*, *calculate_covariance=False*)

A framework for fitting models to data for tasks, along with recording the data associated with the fits.

Parameters

- **data_folder** (*string or list of strings, optional*) – The folder where the data can be found. Default is the current folder.
- **data_format** (*string, optional*) – The file type of the data, from `mat`, `csv`, `xlsx` and `pkl`. Default is `csv`
- **data_file_filter** (*callable, string, list of strings or None, optional*) – A function to process the file names or a list of possible prefixes as strings or a single string. Default `None`, no file names removed
- **data_file_terminal_ID** (*bool, optional*) – Is there an ID number at the end of the filename? If not then a more general search will be performed. Default `True`
- **data_read_options** (*dict, optional*) – The keyword arguments for the data importing method chosen
- **data_split_by**(*string or list of strings, optional*) – If multiple participant datasets are in one file sheet, this specifies the column or columns that can distinguish and identify the rows for each participant. Default `None`
- **data_group_by** (*list of strings, optional*) – A list of parts of filenames that are repeated across participants, identifying all the files that should be grouped together to form one participants data. The rest of the filename is assumed to identify the participant. Default is `None`
- **data_extra_processing**(*callable, optional*) – A function that modifies the dictionary of data read for each participant in such that it is appropriate for fitting. Default is `None`
- **model_name** (*string, optional*) – The name of the file where a model.modelTemplate.Model class can be found. Default `QLearn`
- **model_changing_properties** (*dictionary with values of tuple of two floats, optional*) – Parameters are the options that you allow to vary across model fits. Each model parameter is specified as a dict key. The value is a tuple containing the upper and lower search bounds, e.g. `alpha` has the bounds (0, 1). Default `None`
- **model_constant_properties** (*dictionary of float, string or binary valued elements, optional*) – These contain all the the model options that define the version of the model being studied. Default `None`
- **participantID** (*basestring, optional*) – The key (label) used to identify each participant. Default `Name`

- **participant_choices** (*string, optional*) – The participant data key of their action choices. Default `'Actions'`
- **participant_rewards** (*string, optional*) – The participant data key of the participant reward data. Default `'Rewards'`
- **model_fit_value** (*string, optional*) – The key to be compared in the model data. Default `'ActionProb'`
- **fit_subset** (`float('Nan')`, `None`, `"rewarded"`, `"unrewarded"`, `"all"` or list of int, optional) – Describes which, if any, subset of trials will be used to evaluate the performance of the model. This can either be described as a list of trial numbers or, by passing - `"all"` for fitting all trials - `float('Nan')` or `"unrewarded"` for all those trials whose feedback was `float('Nan')` - `"rewarded"` for those who had feedback that was not `float('Nan')` Default `None`, which means all trials will be used.
- **task_stimuli** (*list of strings or None, optional*) – The keys containing the observational parameters seen by the participant before taking a decision on an action. Default `None`
- **participant_action_options** (*string or list of strings or None or one element list with a list, optional*) – If a string or list of strings these are treated as dict keys where the valid actions for each trial can be found. If None then all trials will use all available actions. If the list contains one list then it will be treated as a list of valid actions for each trialstep. Default `'None'`
- **fit_method** (*string, optional*) – The fitting method to be used. The names accepted are those of the modules in the folder fitAlgs containing a FitAlg class. Default `'evolutionary'`
- **fit_method_args** (*dict, optional*) – A dictionary of arguments specific to the fitting method. Default `None`
- **fit_measure** (*string, optional*) – The name of the function used to calculate the quality of the fit. The value it returns provides the fitter with its fitting guide. Default `-loge`
- **fit_measure_args** (*dict, optional*) – The parameters used to initialise fitMeasure and extraFitMeasures. Default `None`
- **fit_extra_measures** (*list of strings, optional*) – List of fit measures not used to fit the model, but to provide more information. Any arguments needed for these measures should be placed in fitMeasureArgs. Default `None`
- **participant_varying_model_parameters** (*dict of string, optional*) – A dictionary of model settings whose values should vary from participant to participant based on the values found in the imported participant data files. The key is the label given in the participant data file, as a string, and the value is the associated label in the model, also as a string. Default `{}`
- **label** (*string, optional*) – The label for the data fitting. Default `None` will mean no data is saved to files.
- **save_fitting_progress** (*bool, optional*) – Specifies if the results from each iteration of the fitting process should be returned. Default `False`
- **config_file** (*string, optional*) – The file name and path of a `.yaml` configuration file. Overrides all other parameters if found. Default `None`
- **output_path** (*string, optional*) – The path that will be used for the run output. Default `None`
- **pickle** (*bool, optional*) – If true the data for each model, and participant is recorded. Default is `False`

- **boundary_excess_cost_function** (*basestring or callable returning a function, optional*) – The function is used to calculate the penalty for exceeding the boundaries. Default is `boundFunc.scalarBound()`
- **min_log_level** (*basestring, optional*) – Defines the level of the log from (`DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`). Default `INFO`
- **numpy_error_level** (*{'log', 'raise'}*) – Defines the response to numpy errors. Default `log`. See numpy.seterr
- **fit_float_error_response_value** (*float, optional*) – If a floating point error occurs when running a fit the fitter function will return a value for each element of fpRespVal. Default is ‘‘1/1e100‘
- **calculate_covariance** (*bool, optional*) – Is the covariance calculated. Default `False`

See also:

modelGenerator() The model factory

outputting() The outputting functions

fitAlgs.fitAlg.FitAlg() General class for a method of fitting data

fitAlgs.fitSims.fitSim() General class for a method of simulating the fitting of data

data.Data() Data import class

xlsx_fitting_data

`dataFitting.`**xlsx_fitting_data**(*fitting_data*, *label*, *participant*, *file_name_generator*)

Saves the fitting data to an XLSX file

Parameters

- **fitting_data** (*dict, optional*) – Dictionary of details of the different fits, including an ordered dictionary containing the parameter values tested, in the order they were tested, and a list of the fit qualities of these parameters.
- **label** (*basestring*) – The label used to identify the fit in the file names
- **participant** (*dict*) – The participant data
- **file_name_generator** (*function*) – Creates a new file with the name <handle> and the extension <extension>. It takes two string parameters: (`handle`, `extension`) and returns one `fileName` string

6.2.1.2 Classes

`FitAlg`([fit_sim, fit_measure, ...])	The abstract class for fitting data
`FitSim`([participant_choice_property, ...])	A class for fitting data by passing the participant data through the model.
LengthError	
`ModelGen`(model_name[, parameters, other_options])	Generates model class instances based on a model and a set of varying parameters
OrderError	

LengthError

exception `dataFitting.`**LengthError**

OrderError

exception dataFitting.**OrderError**

6.2.1.3 Class Inheritance Diagram

Author Dominic Hunt

exception dataFitting.**LengthError**
Bases: exceptions.Exception

exception dataFitting.**OrderError**
Bases: exceptions.Exception

dataFitting.**fit_record**(*participant_fits*, *file_name_generator*)
Returns the participant fits summary as a csv file

Parameters

- **participant_fits** (*dict*) – A summary of the recovered parameters
- **file_name_generator** (*function*) – Creates a new file with the name <handle> and the extension <extension>. It takes two string parameters: (handle, extension) and returns one fileName string

dataFitting.**log_fitting_parameters**(*fit_info*)
Records and outputs to the log the parameters associated with the fitting algorithms

Parameters **fit_info** (*dict*) – The details of the fitting

dataFitting.**log_model_fitted_parameters**(*model_fit_variables*, *model_parameters*, *fit_quality*, *participant_name*)
Logs the model and task parameters that used as initial fitting conditions

Parameters

- **model_fit_variables** (*dict*) – The model parameters that have been fitted over and varied.
- **model_parameters** (*dict*) – The model parameters for the fitted model
- **fit_quality** (*float*) – The value of goodness of fit
- **participant_name** (*int or string*) – The identifier for each participant

dataFitting.**log_model_fitting_parameters**(*model*, *model_fit_variables*, *model_other_args*)
Logs the model and task parameters that used as initial fitting conditions

Parameters

- **model** (*string*) – The name of the model
- **model_fit_variables** (*dict*) – The model parameters that will be fitted over and varied.
- **model_other_args** (*dict*) – The other parameters used in the model whose attributes have been modified by the user

dataFitting.**record_fitting**(*fitting_data*, *label*, *participant*, *participant_model_variables*, *participant_fits*, *file_name_generator*, *save_fitting_progress=False*)
Records formatted versions of the fitting data

Parameters

- **fitting_data** (*dict, optional*) – Dictionary of details of the different fits, including an ordered dictionary containing the parameter values tested, in the order they were tested, and a list of the fit qualities of these parameters.
- **label** (*basestring*) – The label used to identify the fit in the file names
- **participant** (*dict*) – The participant data
- **participant_model_variables** (*dict of string*) – A dictionary of model settings whose values should vary from participant to participant based on the values found in the imported participant data files. The key is the label given in the participant data file, as a string, and the value is the associated label in the model, also as a string.
- **participant_fits** (*defaultdict of lists*) – A dictionary to be filled with the summary of the participant fits
- **file_name_generator** (*function*) – Creates a new file with the name <handle> and the extension <extension>. It takes two string parameters: (`handle`, `extension`) and returns one `fileName` string
- **save_fitting_progress** (*bool, optional*) – Specifies if the results from each iteration of the fitting process should be returned. Default `False`

Returns **participant_fits** – A dictionary to be filled with the summary of the previous and current participant fits

Return type defaultdict of lists

dataFitting.**record_participant_fit**(*participant*, *part_name*, *model_data*, *model_name*, *fitting_data*, *partModelVars*, *participantFits*, *fileNameGen=None*, *pickleData=False*, *saveFittingProgress=False*, *expData=None*)

Record the data relevant to the participant fitting

Parameters

- **participant** (*dict*) – The participant data
- **part_name** (*int or string*) – The identifier for each participant
- **model_data** (*dict*) – The data from the model
- **model_name** (*basestring*) – The label given to the model
- **fitting_data** (*dict*) – Dictionary of details of the different fits, including an ordered dictionary containing the parameter values tested, in the order they were tested, and a list of the fit qualities of these parameters
- **partModelVars** (*dict of string*) – A dictionary of model settings whose values should vary from participant to participant based on the values found in the imported participant data files. The key is the label given in the participant data file, as a string, and the value is the associated label in the model, also as a string.
- **participantFits** (*defaultdict of lists*) – A dictionary to be filled with the summary of the participant fits
- **fileNameGen** (*function or None*) – Creates a new file with the name <handle> and the extension <extension>. It takes two string parameters: (`handle`, `extension`) and returns one `fileName` string. Default `None`
- **pickleData** (*bool, optional*) – If true the data for each model, task and participant is recorded. Default is `False`
- **saveFittingProgress** (*bool, optional*) – Specifies if the results from each iteration of the fitting process should be returned. Default `False`
- **expData** (*dict, optional*) – The data from the task. Default `None`

Returns **participantFits** – A dictionary to be filled with the summary of the previous and current participant fits

Return type defaultdict of lists

See also:

outputting.pickleLog() records the picked data

dataFitting.**run**(*data_folder=u'./'*, *data_format=u'csv'*, *data_file_filter=None*, *data_file_terminal_ID=True*, *data_read_options=None*, *data_split_by=None*, *data_group_by=None*, *data_extra_processing=None*, *model_name=u'QLearn'*, *model_changing_properties=None*, *model_constant_properties=None*, *participantID=u'Name'*, *participant_choices=u'Actions'*, *participant_rewards=u'Rewards'*, *model_fit_value=u'ActionProb'*, *fit_subset=None*, *task_stimuli=None*, *participant_action_options=None*, *fit_method=u'Evolutionary'*, *fit_method_args=None*, *fit_measure=u'-loge'*, *fit_measure_args=None*, *fit_extra_measures=None*, *participant_varying_model_parameters=None*, *label=None*, *save_fitting_progress=False*, *config_file=None*, *output_path=None*, *pickle=False*, *boundary_excess_cost_function=None*, *min_log_level=u'INFO'*, *numpy_error_level=u'log'*, *fit_float_error_response_value=1e-100*, *calculate_covariance=False*)

A framework for fitting models to data for tasks, along with recording the data associated with the fits.

Parameters

- **data_folder** (*string or list of strings, optional*) – The folder where the data can be found. Default is the current folder.
- **data_format** (*string, optional*) – The file type of the data, from `mat`, `csv`, `xlsx` and `pkl`. Default is `csv`
- **data_file_filter** (*callable, string, list of strings or None, optional*) – A function to process the file names or a list of possible prefixes as strings or a single string. Default `None`, no file names removed
- **data_file_terminal_ID** (*bool, optional*) – Is there an ID number at the end of the filename? If not then a more general search will be performed. Default `True`
- **data_read_options** (*dict, optional*) – The keyword arguments for the data importing method chosen
- **data_split_by** (*string or list of strings, optional*) – If multiple participant datasets are in one file sheet, this specifies the column or columns that can distinguish and identify the rows for each participant. Default `None`
- **data_group_by** (*list of strings, optional*) – A list of parts of filenames that are repeated across participants, identifying all the files that should be grouped together to form one participants data. The rest of the filename is assumed to identify the participant. Default is `None`
- **data_extra_processing** (*callable, optional*) – A function that modifies the dictionary of data read for each participant in such that it is appropriate for fitting. Default is `None`
- **model_name** (*string, optional*) – The name of the file where a model.modelTemplate.Model class can be found. Default `QLearn`
- **model_changing_properties** (*dictionary with values of tuple of two floats, optional*) – Parameters are the options that you allow to vary across model fits. Each model parameter is specified as a dict key. The value is a tuple containing the upper and lower search bounds, e.g. `alpha` has the bounds (0, 1). Default `None`

- **model_constant_properties** (*dictionary of float, string or binary valued elements, optional*) – These contain all the the model options that define the version of the model being studied. Default `None`
- **participantID** (*basestring, optional*) – The key (label) used to identify each participant. Default `Name`
- **participant_choices** (*string, optional*) – The participant data key of their action choices. Default `'Actions'`
- **participant_rewards** (*string, optional*) – The participant data key of the participant reward data. Default `'Rewards'`
- **model_fit_value** (*string, optional*) – The key to be compared in the model data. Default `'ActionProb'`
- **fit_subset** (`float('Nan')`, `None`, `"rewarded"`, `"unrewarded"`, `"all"` or list of int, optional) – Describes which, if any, subset of trials will be used to evaluate the performance of the model. This can either be described as a list of trial numbers or, by passing - `"all"` for fitting all trials - `float('Nan')` or `"unrewarded"` for all those trials whose feedback was `float('Nan')` - `"rewarded"` for those who had feedback that was not `float('Nan')` Default `None`, which means all trials will be used.
- **task_stimuli** (*list of strings or None, optional*) – The keys containing the observational parameters seen by the participant before taking a decision on an action. Default `None`
- **participant_action_options** (*string or list of strings or None or one element list with a list, optional*) – If a string or list of strings these are treated as dict keys where the valid actions for each trial can be found. If None then all trials will use all available actions. If the list contains one list then it will be treated as a list of valid actions for each trialstep. Default `'None'`
- **fit_method** (*string, optional*) – The fitting method to be used. The names accepted are those of the modules in the folder fitAlgs containing a FitAlg class. Default `'evolutionary'`
- **fit_method_args** (*dict, optional*) – A dictionary of arguments specific to the fitting method. Default `None`
- **fit_measure** (*string, optional*) – The name of the function used to calculate the quality of the fit. The value it returns provides the fitter with its fitting guide. Default `-loge`
- **fit_measure_args** (*dict, optional*) – The parameters used to initialise fitMeasure and extraFitMeasures. Default `None`
- **fit_extra_measures** (*list of strings, optional*) – List of fit measures not used to fit the model, but to provide more information. Any arguments needed for these measures should be placed in fitMeasureArgs. Default `None`
- **participant_varying_model_parameters** (*dict of string, optional*) – A dictionary of model settings whose values should vary from participant to participant based on the values found in the imported participant data files. The key is the label given in the participant data file, as a string, and the value is the associated label in the model, also as a string. Default `{}`
- **label** (*string, optional*) – The label for the data fitting. Default `None` will mean no data is saved to files.
- **save_fitting_progress** (*bool, optional*) – Specifies if the results from each iteration of the fitting process should be returned. Default `False`
- **config_file** (*string, optional*) – The file name and path of a `.yaml` configuration file. Overrides all other parameters if found. Default `None`

- **output_path** (*string, optional*) – The path that will be used for the run output. Default `None`
- **pickle** (*bool, optional*) – If true the data for each model, and participant is recorded. Default is `False`
- **boundary_excess_cost_function** (*basestring or callable returning a function, optional*) – The function is used to calculate the penalty for exceeding the boundaries. Default is `boundFunc.scalarBound()`
- **min_log_level** (*basestring, optional*) – Defines the level of the log from (`DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`). Default `INFO`
- **numpy_error_level** (*{'log', 'raise'}*) – Defines the response to numpy errors. Default `log`. See numpy.seterr
- **fit_float_error_response_value** (*float, optional*) – If a floating point error occurs when running a fit the fitter function will return a value for each element of fpRespVal. Default is ‘‘1/1e100‘
- **calculate_covariance** (*bool, optional*) – Is the covariance calculated. Default `False`

See also:

modelGenerator() The model factory

outputting() The outputting functions

fitAlgs.fitAlg.FitAlg() General class for a method of fitting data

fitAlgs.fitSims.fitSim() General class for a method of simulating the fitting of data

data.Data() Data import class

`dataFitting.`**xlsx_fitting_data**(*fitting_data*, *label*, *participant*, *file_name_generator*)

Saves the fitting data to an XLSX file

Parameters

- **fitting_data** (*dict, optional*) – Dictionary of details of the different fits, including an ordered dictionary containing the parameter values tested, in the order they were tested, and a list of the fit qualities of these parameters.
- **label** (*basestring*) – The label used to identify the fit in the file names
- **participant** (*dict*) – The participant data
- **file_name_generator** (*function*) – Creates a new file with the name <handle> and the extension <extension>. It takes two string parameters: (`handle`, `extension`) and returns one `fileName` string

6.3 data module

6.3.1 data Module

This module allows for the importing of participant data for use in fitting

Author Dominic Hunt

6.3.1.1 Functions

sort_by_last_number(dataFiles)	

sort_by_last_number

data.**sort_by_last_number**(*dataFiles*)

6.3.1.2 Classes

Data(participants[, participantID, choices, ...])	
DimentionError	
FileError	
FileFilterError	
FileTypeError	
FoldersError	
IDError	
LengthError	
ProcessingError	

Data

class data.**Data**(*participants*, *participantID=u'ID'*, *choices=u'actions'*, *feedbacks=u'feedbacks'*, *stimuli=None*, *action_options=None*, *process_data_function=None*)

Bases: list

Methods Summary

extend(iterable)	Combines two Data instances into one
from_csv([folder, file_name_filter, ...])	Import data from a folder full of .csv files, where each file contains the information of one participant
from_mat([folder, file_name_filter, ...])	Import data from a folder full of .mat files, where each file contains the information of one participant
from_pkl([folder, file_name_filter, ...])	Import data from a folder full of .pkl files, where each file contains the information of one participant.
from_xlsx([folder, file_name_filter, ...])	Import data from a folder full of .xlsx files, where each file contains the information of one participant
load_data([file_type, folders, ...])	Import data from a folder.

Methods Documentation

extend(*iterable*)

Combines two Data instances into one

Parameters **iterable** (*Data instance or list of participant dicts*) –

classmethod **from_csv**(*folder=u'./'*, *file_name_filter=None*, *terminal_ID=True*, *split_by=None*, *participantID=None*, *choices=u'actions'*, *feedbacks=u'feedbacks'*, *stimuli=None*, *action_options=None*, *group_by=None*, *extra_processing=None*, *csv_read_options=None*)

Import data from a folder full of .csv files, where each file contains the information of one participant

Parameters

- **folder** (*string, optional*) – The folder where the data can be found. Default is the current folder.
- **file_name_filter** (*callable, string, list of strings or None, optional*) – A function to process the file names or a list of possible prefixes as strings or a single string. Default `None`, no file names removed
- **terminal_ID** (*bool, optional*) – Is there an ID number at the end of the filename? If not then a more general search will be performed. Default `True`
- **split_by** (*string or list of strings, optional*) – If multiple participants datasets are in one file sheet, this specifies the column or columns that can distinguish and identify the rows for each participant. Default `None`
- **participantID** (*string, optional*) – The dict key where the participant ID can be found. Default `None`, which results in the file name being used.
- **choices** (*string, optional*) – The dict key where the participant choices can be found. Default `'actions'`
- **feedbacks** (*string, optional*) – The dict key where the feedbacks the participant received can be found. Default `'feedbacks'`
- **stimuli** (*string or list of strings, optional*) – The dict keys where the stimulus cues for each trial can be found. Default `'None'`
- **action_options** (*string or list of strings or None or one element list with a list, optional*) – If a string or list of strings these are treated as dict keys where the valid actions for each trial can be found. If None then all trials will use all available actions. If the list contains one list then it will be treated as a list of valid actions for each trialstep. Default `'None'`
- **group_by** (*list of strings, optional*) – A list of parts of filenames that are repeated across participants, identifying all the files that should be grouped together to form one participants data. The rest of the filename is assumed to identify the participant. Default is `None`
- **extra_processing** (*callable, optional*) – A function that modifies the dictionary of data read for each participant in such that it is appropriate for fitting. Default is `None`
- **csv_read_options** (*dict, optional*) – The keyword arguments for pandas.read_csv. Default `{}`

Returns Data

Return type Data class instance

See also:

`pandas.read_csv()`

classmethod from_mat(*folder=u'./'*, *file_name_filter=None*, *terminal_ID=True*, *participantID=None*, *choices=u'actions'*, *feedbacks=u'feedbacks'*, *stimuli=None*, *action_options=None*, *group_by=None*, *extra_processing=None*)

Import data from a folder full of .mat files, where each file contains the information of one participant

Parameters

- **folder** (*string, optional*) – The folder where the data can be found. Default is the current folder.
- **file_name_filter** (*callable, string, list of strings or None, optional*) – A function to process the file names or a list of possible prefixes as strings or a single string. Default `None`, no file names removed

- **terminal_ID** (*bool, optional*) – Is there an ID number at the end of the filename? If not then a more general search will be performed. Default `True`
- **participantID** (*string, optional*) – The dict key where the participant ID can be found. Default `None`, which results in the file name being used.
- **choices** (*string, optional*) – The dict key where the participant choices can be found. Default `'actions'`
- **feedbacks** (*string, optional*) – The dict key where the feedbacks the participant received can be found. Default `'feedbacks'`
- **stimuli** (*string or list of strings, optional*) – The dict keys where the stimulus cues for each trial can be found. Default `'None'`
- **action_options** (*string or list of strings or None or one element list with a list, optional*) – If a string or list of strings these are treated as dict keys where the valid actions for each trial can be found. If None then all trials will use all available actions. If the list contains one list then it will be treated as a list of valid actions for each trialstep. Default `'None'`
- **group_by** (*list of strings, optional*) – A list of parts of filenames that are repeated across participants, identifying all the files that should be grouped together to form one participants data. The rest of the filename is assumed to identify the participant. Default is `None`
- **extra_processing** (*callable, optional*) – A function that modifies the dictionary of data read for each participant in such that it is appropriate for fitting. Default is `None`

Returns Data

Return type Data class instance

See also:

`scipy.io.loadmat()`

classmethod from_pkl(*folder=u'./'*, *file_name_filter=None*, *terminal_ID=True*, *participantID=None*, *choices=u'actions'*, *feedbacks=u'feedbacks'*, *stimuli=None*, *action_options=None*, *group_by=None*, *extra_processing=None*)

Import data from a folder full of .pkl files, where each file contains the information of one participant. This will principally be used to import data stored by task simulations

Parameters

- **folder** (*string, optional*) – The folder where the data can be found. Default is the current folder.
- **file_name_filter** (*callable, string, list of strings or None, optional*) – A function to process the file names or a list of possible prefixes as strings or a single string. Default `None`, no file names removed
- **terminal_ID** (*bool, optional*) – Is there an ID number at the end of the filename? If not then a more general search will be performed. Default `True`
- **participantID** (*string, optional*) – The dict key where the participant ID can be found. Default `None`, which results in the file name being used.
- **choices** (*string, optional*) – The dict key where the participant choices can be found. Default `'actions'`
- **feedbacks** (*string, optional*) – The dict key where the feedbacks the participant received can be found. Default `'feedbacks'`
- **stimuli** (*string or list of strings, optional*) – The dict keys where the stimulus cues for each trial can be found. Default `'None'`

- **action_options** (*string or list of strings or None or one element list with a list, optional*) – If a string or list of strings these are treated as dict keys where the valid actions for each trial can be found. If None then all trials will use all available actions. If the list contains one list then it will be treated as a list of valid actions for each trialstep. Default `'None'`
- **group_by** (*list of strings, optional*) – A list of parts of filenames that are repeated across participants, identifying all the files that should be grouped together to form one participants data. The rest of the filename is assumed to identify the participant. Default is `None`
- **extra_processing** (*callable, optional*) – A function that modifies the dictionary of data read for each participant in such that it is appropriate for fitting. Default is `None`

Returns Data

Return type Data class instance

classmethod from_xlsx(*folder=u'./'*, *file_name_filter=None*, *terminal_ID=True*, *split_by=None*, *participantID=None*, *choices=u'actions'*, *feedbacks=u'feedbacks'*, *stimuli=None*, *action_options=None*, *group_by=None*, *extra_processing=None*, *xlsx_read_options=None*)

Import data from a folder full of .xlsx files, where each file contains the information of one participant

Parameters

- **folder** (*string, optional*) – The folder where the data can be found. Default is the current folder.
- **file_name_filter** (*callable, string, list of strings or None, optional*) – A function to process the file names or a list of possible prefixes as strings or a single string. Default `None`, no file names removed
- **terminal_ID** (*bool, optional*) – Is there an ID number at the end of the filename? If not then a more general search will be performed. Default `True`
- **split_by** (*string or list of strings, optional*) – If multiple participants datasets are in one file sheet, this specifies the column or columns that can distinguish and identify the rows for each participant. Default `None`
- **participantID** (*string, optional*) – The dict key where the participant ID can be found. Default `None`, which results in the file name being used.
- **choices** (*string, optional*) – The dict key where the participant choices can be found. Default `'actions'`
- **feedbacks** (*string, optional*) – The dict key where the feedbacks the participant received can be found. Default `'feedbacks'`
- **stimuli** (*string or list of strings, optional*) – The dict keys where the stimulus cues for each trial can be found. Default `'None'`
- **action_options** (*string or list of strings or None or one element list with a list, optional*) – If a string or list of strings these are treated as dict keys where the valid actions for each trial can be found. If None then all trials will use all available actions. If the list contains one list then it will be treated as a list of valid actions for each trialstep. Default `'None'`
- **group_by** (*list of strings, optional*) – A list of parts of filenames that are repeated across participants, identifying all the files that should be grouped together to form one participants data. The rest of the filename is assumed to identify the participant. Default is `None`

- **extra_processing** (*callable, optional*) – A function that modifies the dictionary of data read for each participant in such that it is appropriate for fitting. Default is `None`
- **xlsx_read_options** (*dict, optional*) – The keyword arguments for pandas.read_excel

Returns Data

Return type Data class instance

See also:

`pandas.read_excel()`

classmethod load_data(*file_type=u'csv'*, *folders=u'./'*, *file_name_filter=None*, *terminal_ID=True*, *split_by=None*, *participantID=None*, *choices=u'actions'*, *feedbacks=u'feedbacks'*, *stimuli=None*, *action_options=None*, *group_by=None*, *extra_processing=None*, *data_read_options=None*)

Import data from a folder. This is a wrapper function for the other import methods

Parameters

- **file_type** (*string, optional*) – The file type of the data, from `mat`, `csv`, `xlsx` and `pkl`. Default is `csv`
- **folders** (*string or list of strings, optional*) – The folder or folders where the data can be found. Default is the current folder.
- **file_name_filter** (*callable, string, list of strings or None, optional*) – A function to process the file names or a list of possible prefixes as strings or a single string. Default `None`, no file names removed
- **terminal_ID** (*bool, optional*) – Is there an ID number at the end of the filename? If not then a more general search will be performed. Default `True`
- **split_by** (*string or list of strings, optional*) – If multiple participant datasets are in one file sheet, this specifies the column or columns that can distinguish and identify the rows for each participant. Default `None`
- **participantID** (*string, optional*) – The dict key where the participant ID can be found. Default `None`, which results in the file name being used.
- **choices** (*string, optional*) – The dict key where the participant choices can be found. Default `'actions'`
- **feedbacks** (*string, optional*) – The dict key where the feedbacks the participant received can be found. Default `'feedbacks'`
- **stimuli** (*string or list of strings, optional*) – The dict keys where the stimulus cues for each trial can be found. Default `'None'`
- **action_options** (*string or list of strings or None or one element list with a list, optional*) – If a string or list of strings these are treated as dict keys where the valid actions for each trial can be found. If None then all trials will use all available actions. If the list contains one list then it will be treated as a list of valid actions for each trialstep. Default `'None'`
- **group_by** (*list of strings, optional*) – A list of parts of filenames that are repeated across participants, identifying all the files that should be grouped together to form one participants data. The rest of the filename is assumed to identify the participant. Default is `None`
- **extra_processing** (*callable, optional*) – A function that modifies the dictionary of data read for each participant in such that it is appropriate for fitting. Default is `None`

- **data_read_options** (*dict, optional*) – The keyword arguments for the data importing method chosen

Returns Data

Return type Data class instance

DimentionError

exception data.**DimentionError**

FileError

exception data.**FileError**

FileFilterError

exception data.**FileFilterError**

FileTypeError

exception data.**FileTypeError**

FoldersError

exception data.**FoldersError**

IDError

exception data.**IDError**

LengthError

exception data.**LengthError**

ProcessingError

exception data.**ProcessingError**

6.3.1.3 Class Inheritance Diagram

This module allows for the importing of participant data for use in fitting

Author Dominic Hunt

class data.**Data**(*participants*, *participantID=u'ID'*, *choices=u'actions'*, *feedbacks=u'feedbacks'*, *stimuli=None*, *action_options=None*, *process_data_function=None*)

Bases: list

extend(*iterable*)

Combines two Data instances into one

Parameters **iterable** (*Data instance or list of participant dicts*) –

classmethod from_csv(*folder=u'./'*, *file_name_filter=None*, *terminal_ID=True*, *split_by=None*, *participantID=None*, *choices=u'actions'*, *feedbacks=u'feedbacks'*, *stimuli=None*, *action_options=None*, *group_by=None*, *extra_processing=None*, *csv_read_options=None*)

Import data from a folder full of .csv files, where each file contains the information of one participant

Parameters

- **folder** (*string, optional*) – The folder where the data can be found. Default is the current folder.
- **file_name_filter** (*callable, string, list of strings or None, optional*) – A function to process the file names or a list of possible prefixes as strings or a single string. Default `None`, no file names removed
- **terminal_ID** (*bool, optional*) – Is there an ID number at the end of the filename? If not then a more general search will be performed. Default `True`
- **split_by** (*string or list of strings, optional*) – If multiple participants datasets are in one file sheet, this specifies the column or columns that can distinguish and identify the rows for each participant. Default `None`
- **participantID** (*string, optional*) – The dict key where the participant ID can be found. Default `None`, which results in the file name being used.
- **choices** (*string, optional*) – The dict key where the participant choices can be found. Default `'actions'`
- **feedbacks** (*string, optional*) – The dict key where the feedbacks the participant received can be found. Default `'feedbacks'`
- **stimuli** (*string or list of strings, optional*) – The dict keys where the stimulus cues for each trial can be found. Default `'None'`
- **action_options** (*string or list of strings or None or one element list with a list, optional*) – If a string or list of strings these are treated as dict keys where the valid actions for each trial can be found. If None then all trials will use all available actions. If the list contains one list then it will be treated as a list of valid actions for each trialstep. Default `'None'`
- **group_by** (*list of strings, optional*) – A list of parts of filenames that are repeated across participants, identifying all the files that should be grouped together to form one participants data. The rest of the filename is assumed to identify the participant. Default is `None`
- **extra_processing** (*callable, optional*) – A function that modifies the dictionary of data read for each participant in such that it is appropriate for fitting. Default is `None`
- **csv_read_options** (*dict, optional*) – The keyword arguments for pandas.read_csv. Default `{}`

Returns **Data**

Return type Data class instance

See also:

`pandas.read_csv()`

classmethod from_mat(*folder=u'./'*, *file_name_filter=None*, *terminal_ID=True*, *participantID=None*, *choices=u'actions'*, *feedbacks=u'feedbacks'*, *stimuli=None*, *action_options=None*, *group_by=None*, *extra_processing=None*)

Import data from a folder full of .mat files, where each file contains the information of one participant

Parameters

- **folder** (*string, optional*) – The folder where the data can be found. Default is the current folder.
- **file_name_filter** (*callable, string, list of strings or None, optional*) – A function to process the file names or a list of possible prefixes as strings or a single string. Default `None`, no file names removed
- **terminal_ID** (*bool, optional*) – Is there an ID number at the end of the filename? If not then a more general search will be performed. Default `True`
- **participantID** (*string, optional*) – The dict key where the participant ID can be found. Default `None`, which results in the file name being used.
- **choices** (*string, optional*) – The dict key where the participant choices can be found. Default `'actions'`
- **feedbacks** (*string, optional*) – The dict key where the feedbacks the participant received can be found. Default `'feedbacks'`
- **stimuli** (*string or list of strings, optional*) – The dict keys where the stimulus cues for each trial can be found. Default `'None'`
- **action_options** (*string or list of strings or None or one element list with a list, optional*) – If a string or list of strings these are treated as dict keys where the valid actions for each trial can be found. If None then all trials will use all available actions. If the list contains one list then it will be treated as a list of valid actions for each trialstep. Default `'None'`
- **group_by** (*list of strings, optional*) – A list of parts of filenames that are repeated across participants, identifying all the files that should be grouped together to form one participants data. The rest of the filename is assumed to identify the participant. Default is `None`
- **extra_processing** (*callable, optional*) – A function that modifies the dictionary of data read for each participant in such that it is appropriate for fitting. Default is `None`

Returns Data

Return type Data class instance

See also:

`scipy.io.loadmat()`

classmethod from_pkl(*folder=u'./'*, *file_name_filter=None*, *terminal_ID=True*, *participantID=None*, *choices=u'actions'*, *feedbacks=u'feedbacks'*, *stimuli=None*, *action_options=None*, *group_by=None*, *extra_processing=None*)

Import data from a folder full of .pkl files, where each file contains the information of one participant. This will principally be used to import data stored by task simulations

Parameters

- **folder** (*string, optional*) – The folder where the data can be found. Default is the current folder.
- **file_name_filter** (*callable, string, list of strings or None, optional*) – A function to process the file names or a list of possible prefixes as strings or a single string. Default `None`, no file names removed
- **terminal_ID** (*bool, optional*) – Is there an ID number at the end of the filename? If not then a more general search will be performed. Default `True`
- **participantID** (*string, optional*) – The dict key where the participant ID can be found. Default `None`, which results in the file name being used.

- **choices** (*string, optional*) – The dict key where the participant choices can be found. Default `'actions'`
- **feedbacks** (*string, optional*) – The dict key where the feedbacks the participant received can be found. Default `'feedbacks'`
- **stimuli** (*string or list of strings, optional*) – The dict keys where the stimulus cues for each trial can be found. Default `'None'`
- **action_options** (*string or list of strings or None or one element list with a list, optional*) – If a string or list of strings these are treated as dict keys where the valid actions for each trial can be found. If None then all trials will use all available actions. If the list contains one list then it will be treated as a list of valid actions for each trialstep. Default `'None'`
- **group_by** (*list of strings, optional*) – A list of parts of filenames that are repeated across participants, identifying all the files that should be grouped together to form one participants data. The rest of the filename is assumed to identify the participant. Default is `None`
- **extra_processing** (*callable, optional*) – A function that modifies the dictionary of data read for each participant in such that it is appropriate for fitting. Default is `None`

Returns Data

Return type Data class instance

classmethod from_xlsx(*folder=u'./'*, *file_name_filter=None*, *terminal_ID=True*, *split_by=None*, *participantID=None*, *choices=u'actions'*, *feedbacks=u'feedbacks'*, *stimuli=None*, *action_options=None*, *group_by=None*, *extra_processing=None*, *xlsx_read_options=None*)

Import data from a folder full of .xlsx files, where each file contains the information of one participant

Parameters

- **folder** (*string, optional*) – The folder where the data can be found. Default is the current folder.
- **file_name_filter** (*callable, string, list of strings or None, optional*) – A function to process the file names or a list of possible prefixes as strings or a single string. Default `None`, no file names removed
- **terminal_ID** (*bool, optional*) – Is there an ID number at the end of the filename? If not then a more general search will be performed. Default `True`
- **split_by** (*string or list of strings, optional*) – If multiple participants datasets are in one file sheet, this specifies the column or columns that can distinguish and identify the rows for each participant. Default `None`
- **participantID** (*string, optional*) – The dict key where the participant ID can be found. Default `None`, which results in the file name being used.
- **choices** (*string, optional*) – The dict key where the participant choices can be found. Default `'actions'`
- **feedbacks** (*string, optional*) – The dict key where the feedbacks the participant received can be found. Default `'feedbacks'`
- **stimuli** (*string or list of strings, optional*) – The dict keys where the stimulus cues for each trial can be found. Default `'None'`
- **action_options** (*string or list of strings or None or one element list with a list, optional*) – If a string or list of strings these are treated as dict keys where the valid actions for each trial can be found. If

None then all trials will use all available actions. If the list contains one list then it will be treated as a list of valid actions for each trialstep. Default `'None'`

- **group_by** (*list of strings, optional*) – A list of parts of filenames that are repeated across participants, identifying all the files that should be grouped together to form one participants data. The rest of the filename is assumed to identify the participant. Default is `None`
- **extra_processing** (*callable, optional*) – A function that modifies the dictionary of data read for each participant in such that it is appropriate for fitting. Default is `None`
- **xlsx_read_options** (*dict, optional*) – The keyword arguments for pandas.read_excel

Returns Data

Return type Data class instance

See also:

`pandas.read_excel()`

classmethod load_data(*file_type=u'csv'*, *folders=u'./'*, *file_name_filter=None*, *terminal_ID=True*, *split_by=None*, *participantID=None*, *choices=u'actions'*, *feedbacks=u'feedbacks'*, *stimuli=None*, *action_options=None*, *group_by=None*, *extra_processing=None*, *data_read_options=None*)

Import data from a folder. This is a wrapper function for the other import methods

Parameters

- **file_type** (*string, optional*) – The file type of the data, from `mat`, `csv`, `xlsx` and `pkl`. Default is `csv`
- **folders** (*string or list of strings, optional*) – The folder or folders where the data can be found. Default is the current folder.
- **file_name_filter** (*callable, string, list of strings or None, optional*) – A function to process the file names or a list of possible prefixes as strings or a single string. Default `None`, no file names removed
- **terminal_ID** (*bool, optional*) – Is there an ID number at the end of the filename? If not then a more general search will be performed. Default `True`
- **split_by** (*string or list of strings, optional*) – If multiple participant datasets are in one file sheet, this specifies the column or columns that can distinguish and identify the rows for each participant. Default `None`
- **participantID** (*string, optional*) – The dict key where the participant ID can be found. Default `None`, which results in the file name being used.
- **choices** (*string, optional*) – The dict key where the participant choices can be found. Default `'actions'`
- **feedbacks** (*string, optional*) – The dict key where the feedbacks the participant received can be found. Default `'feedbacks'`
- **stimuli** (*string or list of strings, optional*) – The dict keys where the stimulus cues for each trial can be found. Default `'None'`
- **action_options** (*string or list of strings or None or one element list with a list, optional*) – If a string or list of strings these are treated as dict keys where the valid actions for each trial can be found. If None then all trials will use all available actions. If the list contains one list then it will be treated as a list of valid actions for each trialstep. Default `'None'`

- **group_by** (*list of strings, optional*) – A list of parts of filenames that are repeated across participants, identifying all the files that should be grouped together to form one participants data. The rest of the filename is assumed to identify the participant. Default is `None`
- **extra_processing** (*callable, optional*) – A function that modifies the dictionary of data read for each participant in such that it is appropriate for fitting. Default is `None`
- **data_read_options** (*dict, optional*) – The keyword arguments for the data importing method chosen

Returns Data

Return type Data class instance

exception data.**DimentionError**
Bases: exceptions.Exception

exception data.**FileError**
Bases: exceptions.Exception

exception data.**FileFilterError**
Bases: exceptions.Exception

exception data.**FileTypeError**
Bases: exceptions.Exception

exception data.**FoldersError**
Bases: exceptions.Exception

exception data.**IDError**
Bases: exceptions.Exception

exception data.**LengthError**
Bases: exceptions.Exception

exception data.**ProcessingError**
Bases: exceptions.Exception

data.**sort_by_last_number**(*dataFiles*)

6.4 taskGenerator module

6.4.1 taskGenerator Module

Author Dominic Hunt

6.4.1.1 Classes

Task()	The abstract tasks class from which all others inherit
TaskGeneration(task_name[, parameters, ...])	Generates task class instances based on a task and a set of varying parameters

TaskGeneration

class taskGenerator.**TaskGeneration**(*task_name*, *parameters=None*, *other_options=None*)
Bases: object

Generates task class instances based on a task and a set of varying parameters

Parameters

- **task_name** (*string*) – The name of the file where a tasks.taskTemplate.Task class can be found
- **parameters** (*dictionary of floats or lists of floats*) – Parameters are the options that you are or are likely to change across task instances. When a parameter contains a list, an instance of the task will be created for every combination of this parameter with all the others. Default `None`
- **other_options** (*dictionary of float, string or binary valued elements*) – These contain all the the task options that describe the task being studied but do not vary across task instances. Default `None`

Methods Summary

iter_task_ID()	Yields the tasks IDs.
new_task(task_number)	Produces the next tasks instance
next()	Produces the next task instance for the iterator

Methods Documentation

iter_task_ID()
Yields the tasks IDs. To be used with self.new_task(expID) to receive the next tasks instance

Returns **expID** – The ID number that refers to the next tasks parameter combination.

Return type int

new_task(*task_number*)
Produces the next tasks instance

Parameters **task_number** (*int*) – The number of the tasks instance to be initialised

Returns **instance**

Return type tasks.taskTemplate.Task instance

next()
Produces the next task instance for the iterator

Returns **instance**

Return type tasks.taskTemplate.Task instance

6.4.1.2 Class Inheritance Diagram

Author Dominic Hunt

class taskGenerator.**TaskGeneration**(*task_name*, *parameters=None*, *other_options=None*)
Bases: object

Generates task class instances based on a task and a set of varying parameters

Parameters

- **task_name** (*string*) – The name of the file where a tasks.taskTemplate.Task class can be found
- **parameters** (*dictionary of floats or lists of floats*) – Parameters are the options that you are or are likely to change across task instances. When a parameter contains a list, an instance of the task will be created for every combination of this parameter with all the others. Default `None`

- **other_options** (*dictionary of float, string or binary valued elements*) – These contain all the the task options that describe the task being studied but do not vary across task instances. Default `None`

iter_task_ID()
Yields the tasks IDs. To be used with self.new_task(expID) to receive the next tasks instance

Returns **expID** – The ID number that refers to the next tasks parameter combination.

Return type int

new_task(*task_number*)
Produces the next tasks instance

Parameters **task_number** (*int*) – The number of the tasks instance to be initialised

Returns **instance**

Return type tasks.taskTemplate.Task instance

next()
Produces the next task instance for the iterator

Returns **instance**

Return type tasks.taskTemplate.Task instance

6.5 tasks package

6.5.1 tasks Package

6.5.2 Submodules

6.5.2.1 tasks.balltask module

pyhpdm version of the balltask task TODO: describe tasks

class tasks.balltask.**Balltask**(*nbr_of_bags=6, bag_colors=[u'red', u'green', u'blue'], balls_per_bag=3*)
Bases: *tasks.taskTemplate.Task*

feedback()
Responds to the action from the participant balltask has no rewards so we return None

next()
Produces the next stimulus for the iterator

Returns

- **stimulus** (*None*)
- **nextValidActions** (*(0, 1, 2) representing red, green, blue in default case*) – but can be many colors. it's assumed this always goes in same order left to right as bag_colors parameter

Raises `StopIteration`

proceed()
Updates the task after feedback

receiveAction(*action*)
Receives the next action from the participant

Parameters **action** (*int or string*) – The action taken by the model

returnTaskState()
Returns all the relevant data for this task run

Returns **history** – A dictionary containing the class parameters as well as the other useful data

Return type dictionary

storeState()
Stores the state of all the important variables so that they can be output later

class tasks.balltask.**RewardBalltaskDirect**(***kwargs*)
Bases: *model.modelTemplate.Rewards*

Processes the reward for models expecting just the reward

processFeedback(*feedback*, *lastAction*, *stimuli*)

Returns

Return type modelFeedback

class tasks.balltask.**StimulusBalltaskSimple**(***kwargs*)
Bases: *model.modelTemplate.Stimulus*

Processes the stimulus cues for models expecting just the event

processStimulus(*observation*)
Processes the decks stimuli for models expecting just the event

Returns

- **stimuliPresent** (*int or list of int*) – The elements present of the stimulus
- **stimuliActivity** (*float or list of float*) – The activity of each of the elements

6.5.2.2 tasks.basic module

Author Dominic Hunt

Note A simple example of a task class with all the necessary components

class tasks.basic.**Basic**(*trials=100*)
Bases: *tasks.taskTemplate.Task*

An example of a task with all the necessary components, but nothing changing

Parameters **trials** (*int*) – The number of trials in the task

Name
The name of the class used when recording what has been used.

Type string

feedback()
Responds to the action from the participant

next()
the task class is an iterator [link to iterator documentation] this function produces the next stimulus for the task iterator

Returns

- **stimulus** (*None*)
- **nextValidActions** (Tuple of ints or None) – The list of valid actions that the model can respond with. Set to (0,1), as they never vary.

Raises StopIteration

proceed()
Updates the task after feedback

receiveAction(*action*)
Receives the next action from the participant

Parameters **action** (*int or string*) – The action taken by the model

returnTaskState()
Returns all the relevant data for this task run

Returns **results** – A dictionary containing the class parameters as well as the other useful data

Return type dictionary

storeState()
Stores the state of all the important variables so that they can be output later

class tasks.basic.**RewardBasicDirect**(***kwargs*)
Bases: *model.modelTemplate.Rewards*

Processes the reward for models expecting just the reward

processFeedback(*feedback*, *lastAction*, *stimuli*)

Returns

Return type modelFeedback

class tasks.basic.**StimulusBasicSimple**(***kwargs*)
Bases: *model.modelTemplate.Stimulus*

Processes the stimulus cues for models expecting just the event

processStimulus(*observation*)
Processes the decks stimuli for models expecting just the event

Returns

- **stimuliPresent** (*int or list of int*) – The elements present of the stimulus
- **stimuliActivity** (*float or list of float*) – The activity of each of the elements

6.5.2.3 tasks.beads module

Author Dominic Hunt

Reference Jumping to conclusions: a network model predicts schizophrenic patients' performance on a probabilistic reasoning task. *Moore, S. C., & Sellen, J. L. (2006)*. Cognitive, Affective & Behavioral Neuroscience, 6(4), 261–9. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/17458441

class tasks.beads.**Beads**(*N=None, beadSequence=[1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0]*)
Bases: *tasks.taskTemplate.Task*

Based on the Moore & Sellen Beads task

Many methods are inherited from the tasks.taskTemplate.Task class. Refer to its documentation for missing methods.

Name
The name of the class used when recording what has been used.

Type string

Parameters

- **N** (*int, optional*) – Number of beads that could potentially be shown

- **beadSequence** (*list or array of {0,1}, optional*) – The sequence of beads to be shown. Bead sequences can also be embedded in the code and then referred to by name. The only current one is *MooreSellen*, the default sequence.

next()
Produces the next bead for the iterator

Returns

- **bead** (*{0,1}*)
- **nextValidActions** (Tuple of ints or None) – The list of valid actions that the model can respond with. Set to (0,1), as they never vary.

Raises StopIteration

receiveAction(*action*)
Receives the next action from the participant

Parameters **action** (*int or string*) – The action taken by the model

returnTaskState()
Returns all the relevant data for this task run

Returns **results** – A dictionary containing the class parameters as well as the other useful data

Return type dictionary

storeState()
Stores the state of all the important variables so that they can be output later

class tasks.beads.**RewardBeadDirect**(***kwargs*)
Bases: *model.modelTemplate.Rewards*

Processes the beads reward for models expecting just the reward

processFeedback(*feedback*, *lastAction*, *stimuli*)

Returns

Return type modelFeedback

class tasks.beads.**StimulusBeadDirect**(***kwargs*)
Bases: *model.modelTemplate.Stimulus*

Processes the beads stimuli for models expecting just the event

processStimulus(*observation*)
Processes the decks stimuli for models expecting just the event

Returns

- **stimuliPresent** (*int or list of int*)
- **stimuliActivity** (*float or list of float*)

class tasks.beads.**StimulusBeadDualDirect**(***kwargs*)
Bases: *model.modelTemplate.Stimulus*

Processes the beads stimuli for models expecting a tuple of [event,1-event]

processStimulus(*observation*)
Processes the decks stimuli for models expecting just the event

Returns

- **stimuliPresent** (*int or list of int*) – The elements present of the stimulus
- **stimuliActivity** (*float or list of float*) – The activity of each of the elements

class tasks.beads.**StimulusBeadDualInfo**(***kwargs*)
Bases: *model.modelTemplate.Stimulus*

Processes the beads stimuli for models expecting the reward information from two possible actions

Parameters **oneProb** (float in `[0,1]`) – The probability of a 1 from the first jar. This is also the probability of a 0 from the second jar. `event_info` is calculated as `oneProb*event + (1-oneProb)*(1-event)`

oneProb = [0, 1]

processStimulus(*observation*)
Processes the decks stimuli for models expecting just the event

Returns

- **stimuliPresent** (*int or list of int*) – The elements present of the stimulus
- **stimuliActivity** (*float or list of float*) – The activity of each of the elements

tasks.beads.**generateSequence**(*numBeads*, *oneProb*, *switchProb*)
Designed to generate a sequence of beads with a probability of switching jar at any time.

Parameters

- **numBeads** (*int*) – The number of beads in the sequence
- **oneProb** (float in `[0,1]`) – The probability of a 1 from the first jar. This is also the probability of a 0 from the second jar.
- **switchProb** (float in `[0,1]`) – The probability that the drawn beads change the jar they are being drawn from

Returns **sequence** – The generated sequence of beads

Return type list of `{0,1}`

6.5.2.4 tasks.decks module

Author Dominic Hunt

Reference Regulatory fit effects in a choice task *Worthy, D. a, Maddox, W. T., & Markman, A. B. (2007)*. Psychonomic Bulletin & Review, 14(6), 1125–32. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/18229485

class tasks.decks.**Decks**(*draws=None, decks=array([[2, 2, 1, 1, 2, 1, 1, 3, 2, 6, 2, 8, 1, 6, 2, 1, 1, 5, 8, 5, 10, 10, 8, 3, 10, 7, 10, 8, 3, 4, 9, 10, 3, 6, 3, 5, 10, 10, 10, 7, 3, 8, 5, 8, 6, 9, 4, 4, 4, 10, 6, 4, 10, 3, 10, 5, 10, 3, 10, 10, 5, 4, 6, 10, 7, 7, 10, 10, 10, 3, 1, 4, 1, 3, 1, 7, 1, 3, 1, 8], [7, 10, 5, 10, 6, 6, 10, 10, 10, 8, 4, 8, 10, 4, 9, 10, 8, 6, 10, 10, 10, 4, 7, 10, 5, 10, 4, 10, 10, 9, 2, 9, 8, 10, 7, 7, 1, 10, 2, 6, 4, 7, 2, 1, 1, 1, 7, 10, 1, 4, 2, 1, 1, 1, 4, 1, 4, 1, 1, 1, 1, 3, 1, 4, 1, 1, 1, 5, 1, 1, 1, 7, 2, 1, 2, 1, 4, 1, 4, 1]]), discard=False*)
Bases: *tasks.taskTemplate.Task*

Based on the Worthy&Maddox 2007 paper "Regulatory fit effects in a choice task.

Many methods are inherited from the tasks.taskTemplate.Task class. Refer to its documentation for missing methods.

Name
The name of the class used when recording what has been used.

Type string

Parameters

- **draws** (*int, optional*) – Number of cards drawn by the participant
- **decks** (*array of floats, optional*) – The decks of cards

- **discard** (*bool*) – Defines if you discard the card not chosen or if you keep it.

feedback()
Responds to the action from the participant

next()
Produces the next stimulus for the iterator

Returns

- **stimulus** (*None*)
- **nextValidActions** (Tuple of ints or None) – The list of valid actions that the model can respond with. Set to (0,1), as they never vary.

Raises StopIteration

proceed()
Updates the task after feedback

receiveAction(*action*)
Receives the next action from the participant

Parameters **action** (*int or string*) – The action taken by the model

returnTaskState()
Returns all the relevant data for this task run

Returns **results** – A dictionary containing the class parameters as well as the other useful data

Return type dictionary

storeState()
Stores the state of all the important variables so that they can be output later

class tasks.decks.**RewardDecksAllInfo**(***kwargs*)
Bases: *model.modelTemplate.Rewards*

Processes the decks reward for models expecting the reward information from all possible actions

Parameters

- **maxRewardVal** (*int*) – The highest value a reward can have
- **minRewardVal** (*int*) – The lowest value a reward can have
- **number_actions** (*int*) – The number of actions the participant can perform. Assumes the lowest valued action is 0

Returns **deckRew** – The function expects to be passed a tuple containing the reward and the last action. The reward that is a float and action is {0,1}. The function returns a array of length (maxRewardVal-minRewardVal)*number_actions.

Return type function

Name
The identifier of the function

Type string

Examples

```
>>> rew = RewardDecksAllInfo(maxRewardVal=10, minRewardVal=1, number_actions=2)
>>> rew.processFeedback(6, 0, 1)
array([1., 1., 1., 1., 1., 2., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,␣
↪1., 1.])
```

(continues on next page)

(continued from previous page)

```
>>> rew.processFeedback(6, 1, 1)
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 2., 1., 1.,␣
↪1., 1.])
```

maxRewardVal = 10

minRewardVal = 1

number_actions = 2

processFeedback(*reward*, *action*, *stimuli*)

Returns

Return type modelFeedback

class tasks.decks.**RewardDecksDualInfo**(***kwargs*)

Bases: *model.modelTemplate.Rewards*

Processes the decks reward for models expecting the reward information from two possible actions.

epsilon = 1

maxRewardVal = 10

processFeedback(*reward*, *action*, *stimuli*)

Returns

Return type modelFeedback

class tasks.decks.**RewardDecksDualInfoLogistic**(***kwargs*)

Bases: *model.modelTemplate.Rewards*

Processes the decks rewards for models expecting the reward information from two possible actions.

epsilon = 0.3

maxRewardVal = 10

minRewardVal = 1

processFeedback(*reward*, *action*, *stimuli*)

Returns

Return type modelFeedback

class tasks.decks.**RewardDecksLinear**(***kwargs*)

Bases: *model.modelTemplate.Rewards*

Processes the decks reward for models expecting just the reward

processFeedback(*feedback*, *lastAction*, *stimuli*)

Returns

Return type modelFeedback

class tasks.decks.**RewardDecksNormalised**(***kwargs*)

Bases: *model.modelTemplate.Rewards*

Processes the decks reward for models expecting just the reward, but in range [0,1]

Parameters **maxReward**(*int, optional*) – The highest value a reward can have. Default 10

See also:

model.OpAL

maxReward = 10

processFeedback(*feedback*, *lastAction*, *stimuli*)

Returns

Return type modelFeedback

class tasks.decks.**RewardDecksPhi**(***kwargs*)
Bases: *model.modelTemplate.Rewards*

Processes the decks reward for models expecting just the reward, but in range [0, 1]

Parameters **phi** (*float*) – The scaling value of the reward

phi = 1

processFeedback(*feedback*, *lastAction*, *stimuli*)

Returns

Return type modelFeedback

class tasks.decks.**StimulusDecksLinear**(***kwargs*)
Bases: *model.modelTemplate.Stimulus*

processStimulus(*observation*)
Processes the decks stimuli for models expecting just the event

Returns

- **stimuliPresent** (*int or list of int*) – The elements present of the stimulus
- **stimuliActivity** (*float or list of float*) – The activity of each of the elements

6.5.2.5 tasks.taskTemplate module

tasks.taskTemplate Module

Author Dominic

Classes

Task()	The abstract tasks class from which all others inherit

Task

class tasks.taskTemplate.**Task**
Bases: object

The abstract tasks class from which all others inherit

Many general methods for tasks are found only here

Name
The name of the class used when recording what has been used.

Type string

Methods Summary

feedback()	Responds to the action from the participant
get_name()	Returns the name of the class

Continued on next page

Table 10 – continued from previous page

next()	Produces the next stimulus for the iterator
params()	Returns the parameters of the task as a dictionary
proceed()	Updates the task before the next trialstep
receiveAction(action)	Receives the next action from the participant
returnTaskState()	Returns all the relevant data for this task run
standardResultOutput()	
storeState()	Stores the state of all the important variables so that they can be output later

Methods Documentation

feedback()
Responds to the action from the participant

Returns **feedback**

Return type None, int or float

classmethod get_name()
Returns the name of the class

next()
Produces the next stimulus for the iterator

Returns

- **stimulus** (*None*)
- **nextValidActions** (*Tuple of ints*) – The list of valid actions that the model can respond with. Set to `None`, as they never vary.

Raises `StopIteration`

params()
Returns the parameters of the task as a dictionary

Returns **parameters** – The parameters of the task

Return type dict

proceed()
Updates the task before the next trialstep

receiveAction(*action*)
Receives the next action from the participant

Parameters **action** (*int or string*) – The action taken by the model

returnTaskState()
Returns all the relevant data for this task run

Returns **results** – A dictionary containing the class parameters as well as the other useful data

Return type dictionary

standardResultOutput()

storeState()
Stores the state of all the important variables so that they can be output later

Class Inheritance Diagram

Author Dominic

class tasks.taskTemplate.**Task**
Bases: object

The abstract tasks class from which all others inherit

Many general methods for tasks are found only here

Name
The name of the class used when recording what has been used.

Type string

feedback()
Responds to the action from the participant

Returns **feedback**

Return type None, int or float

classmethod get_name()
Returns the name of the class

next()
Produces the next stimulus for the iterator

Returns

- **stimulus** (*None*)
- **nextValidActions** (*Tuple of ints*) – The list of valid actions that the model can respond with. Set to None, as they never vary.

Raises StopIteration

params()
Returns the parameters of the task as a dictionary

Returns **parameters** – The parameters of the task

Return type dict

proceed()
Updates the task before the next trialstep

receiveAction(*action*)
Receives the next action from the participant

Parameters **action** (*int or string*) – The action taken by the model

returnTaskState()
Returns all the relevant data for this task run

Returns **results** – A dictionary containing the class parameters as well as the other useful data

Return type dictionary

standardResultOutput()

storeState()
Stores the state of all the important variables so that they can be output later

6.5.2.6 tasks.pavlov module

Author Dominic Hunt

Reference Value and prediction error in medial frontal cortex: integrating the single-unit and systems levels of analysis. *Silvetti, M., Seurinck, R., & Verguts, T. (2011).* Frontiers in Human Neuroscience, 5(August), 75. doi:10.3389/fnhum.2011.00075

class tasks.pavlov.**Pavlov**(*rewMag=4*, *rewProb=array([0.87*, *0.33])*, *stimMag=1*, *stimDur=20*, *rewDur=4*, *simDur=30*, *stimRepeats=7*)

Bases: `tasks.taskTemplate.Task`

Based on the Silvetti et al 2011 paper *"Value and prediction error in medial frontal cortex: integrating the single-unit and systems levels of analysis."*

Many methods are inherited from the tasks.taskTemplate.Task class. Refer to its documentation for missing methods.

Name

The name of the class used when recording what has been used.

Type string

Parameters

- **rewMag** (*float, optional*) – The size of the stimulus. Default 4
- **rewProb** (*array of floats, optional*) – The probabilities of each stimulus producing a reward. Default [0.85,0.33]
- **stimMag** (*float, optional*) – The size of the stimulus. Default 1
- **stimDur** (*int, optional*) – The duration, in tens of ms, that the stimulus is produced for. This should be longer than rewDur since rewDur is set to end when stimDur ends. Default 200
- **rewDur** (*int, optional*) – The duration, in tens of ms, that the reward is produced for. Default 40
- **simDur** (*int, optional*) – The duration, in tens of ms, that each stimulus event is run for. Default 300
- **stimRepeats** (*int, optional*) – The number of times a stimulus is introduced. Default 72

feedback()

Responds to the action from the participant

next()

Produces the next stimulus for the iterator

Returns

- **nextStim** (*tuple of c, rewSig and stimDur, described below*)
- **c** (*list of floats*) – Contains the inputs for each of the stimuli
- **rewSig** (*list of lists of floats*) – Each list contains the rewards at each time
- **stimDur** (*int*)
- **nextValidActions** (*Tuple of ints*) – The list of valid actions that the model can respond with. Set to `None`, as there are no actions.

Raises `StopIteration`

proceed()

Updates the task after feedback

receiveAction(*action*)

Receives the next action from the participant

Parameters **action** (*int or string*) – The action taken by the model

returnTaskState()

Returns all the relevant data for this task run

Returns **results** – A dictionary containing the class parameters as well as the other useful data

Return type dictionary

storeState()
Stores the state of all the important variables so that they can be output later

tasks.pavlov.**pavlovStimTemporal**()
Passes the pavlov stimuli to models that cope with stimuli and rewards that have a duration.

Returns **pavlovStim** – The function expects to be passed an event with three components: `(stim,rew,stimDur)``and an action (unused) and yield a series of events ``t,c,r`. stim` is the value of the stimulus. It is expected to be a list-like object. `rew` is a list containing the reward for each trialstep. The reward is expected to be a float. `stimDur` is the duration of the stimulus, an `int`. This should be less than the length of `rew`. `c` the stimulus. `r` the reward. `t` is the time

Return type function

tasks.pavlov.**Name**
The identifier of the function

Type string

6.5.2.7 tasks.probSelect module

Author Dominic Hunt

Reference Genetic triple dissociation reveals multiple roles for dopamine in reinforcement learning. Frank, M. J., Moustafa, A. a, Haughey, H. M., Curran, T., & Hutchison, K. E. (2007). Proceedings of the National Academy of Sciences of the United States of America, 104(41), 16311–16316. doi:10.1073/pnas.0706111104

class tasks.probSelect.**ProbSelect**(*reward_probability=0.7*, *learning_action_pairs=None*, *action_reward_probabilities=None*, *learning_length=240*, *test_length=60*, *number_actions=None*, *reward_size=1*)
Bases: *tasks.taskTemplate.Task*

Probabilistic selection task based on Genetic triple dissociation reveals multiple roles for dopamine in reinforcement l
Frank, M. J., Moustafa, A. a, Haughey, H. M., Curran, T., & Hutchison, K. E. (2007). Proceedings of the National Academy of Sciences of the United States of America, 104(41), 16311–16316. doi:10.1073/pnas.0706111104

Many methods are inherited from the tasks.taskTemplate.Task class. Refer to its documentation for missing methods.

Name
The name of the class used when recording what has been used.

Type string

Parameters

- **reward_probability** (*float in range [0,1], optional*) – The probability that a reward is given for choosing action A. Default is 0.7
- **action_reward_probabilities** (*dictionary, optional*) – A dictionary of the potential actions that can be taken and the probability of a reward. Default {0:rewardProb, 1:1-rewardProb, 2:0.5, 3:0.5}
- **learning_action_pairs** (*list of tuples, optional*) – The pairs of actions shown together in the learning phase.

- **learning_length** (*int, optional*) – The number of trials in the learning phase. Default is 240
- **test_length** (*int, optional*) – The number of trials in the test phase. Default is 60
- **reward_size** (*float, optional*) – The size of reward given if successful. Default 1
- **number_actions** (*int, optional*) – The number of actions that can be chosen at any given time, chosen at random from actRewardProb. Default 4

Notes

The task is broken up into two sections: a learning phase and a transfer phase. Participants choose between pairs of four actions: A, B, M1 and M2. Each provides a reward with a different probability: A:P>0.5, B:1-P<0.5, M1=M2=0.5. The transfer phase has all the action pairs but no feedback. This class only covers the learning phase, but models are expected to be implemented as if there is a transfer phase.

feedback()
Responds to the action from the participant

next()
Produces the next stimulus for the iterator

Returns

- **stimulus** (*None*)
- **next_valid_actions** (*Tuple of length 2 of ints*) – The list of valid actions that the model can respond with.

Raises StopIteration

proceed()
Updates the task after feedback

receiveAction(*action*)
Receives the next action from the participant

Parameters **action** (*int or string*) – The action taken by the model

returnTaskState()
Returns all the relevant data for this task run

Returns **results** – A dictionary containing the class parameters as well as the other useful data

Return type dictionary

storeState()
Stores the state of all the important variables so that they can be output later

class tasks.probSelect.**RewardProbSelectDirect**(***kwargs*)
Bases: *model.modelTemplate.Rewards*

Processes the probabilistic selection reward for models expecting just the reward

processFeedback(*reward*, *action*, *stimuli*)

Returns

Return type modelFeedback

class tasks.probSelect.**StimulusProbSelectDirect**(***kwargs*)
Bases: *model.modelTemplate.Stimulus*

Processes the selection stimuli for models expecting just the event

Examples

```
>>> stim = StimulusProbSelectDirect()
>>> stim.processStimulus(1)
(1, 1)
>>> stim.processStimulus(0)
(1, 1)
```

processStimulus(*observation*)

Processes the decks stimuli for models expecting just the event

Returns

- **stimuliPresent** (*int or list of int*)
- **stimuliActivity** (*float or list of float*)

6.5.2.8 tasks.probStim module

Author Dominic Hunt

class tasks.probStim.**Probstim**(*cues=None*, *actualities=None*, *trialsteps=100*, *numStimuli=4*, *correctProb=0.8*, *correctProbabilities=None*, *rewardlessT=None*)

Bases: *tasks.taskTemplate.Task*

Basic probabilistic

Many methods are inherited from the tasks.taskTemplate.Task class. Refer to its documentation for missing methods.

Name

The name of the class used when recording what has been used.

Type string

Parameters

- **actualities** (*int, optional*) – The actual reality the cues pointed to. The correct response the participant is trying to get correct
- **cues** (*array of floats, optional*) – The cues used to guess the actualities
- **trialsteps** (*int, optional*) – If no provided cues, it is the number of trialsteps for the generated set of cues. Default `100`
- **numStimuli** (*int, optional*) – If no provided cues, it is the number of distinct stimuli for the generated set of cues. Default `4`
- **correctProb** (*float in [0,1], optional*) – If no actualities provided, it is the probability of the correct answer being answer 1 rather than answer 0. The default is `0.8`
- **correctProbs** (*list or array of floats in [0,1], optional*) – If no actualities provided, it is the probability of the correct answer being answer 1 rather than answer 0 for each of the different stimuli. Default `[corrProb, 1-corrProb] * (numStimuli//2) + [corrProb] * (numStimuli%2)`
- **rewardlessT** (*int, optional*) – If no actualities provided, it is the number of actualities at the end of the tasks that will have a `None` reward. Default `2*numStimuli`

feedback()

Feedback to the action from the participant

next()
Produces the next stimulus for the iterator

Returns

- **stimulus** (*Tuple*) – The current cues
- **nextValidActions** (Tuple of ints or `None`) – The list of valid actions that the model can respond with. Set to (0,1), as they never vary.

Raises `StopIteration`

proceed()
Updates the task after feedback

receiveAction(*action*)
Receives the next action from the participant

Parameters **action** (*int or string*) – The action taken by the model

returnTaskState()
Returns all the relevant data for this task run

Returns **results** – A dictionary containing the class parameters as well as the other useful data

Return type dictionary

storeState()
Stores the state of all the important variables so that they can be output later

class tasks.probStim.**RewardProbStimDiff**(***kwargs*)
Bases: *model.modelTemplate.Rewards*

Processes the reward for models expecting reward corrections

processFeedback(*feedback*, *lastAction*, *stimuli*)

Returns

Return type modelFeedback

class tasks.probStim.**RewardProbStimDualCorrection**(***kwargs*)
Bases: *model.modelTemplate.Rewards*

Processes the reward for models expecting the reward correction from two possible actions.

epsilon = 1

processFeedback(*feedback*, *lastAction*, *stimuli*)

Returns

Return type modelFeedback

class tasks.probStim.**StimulusProbStimDirect**(***kwargs*)
Bases: *model.modelTemplate.Stimulus*

Processes the stimuli for models expecting just the event

processStimulus(*observation*)
Processes the decks stimuli for models expecting just the event

Returns

- **stimuliPresent** (*int or list of int*) – The elements present of the stimulus
- **stimuliActivity** (*float or list of float*) – The activity of each of the elements

6.5.2.9 tasks.weather module

Author Dominic Hunt

Reference Probabilistic classification learning in amnesia. Knowlton, B. J., Squire, L. R., & Gluck, M. a. (1994). Learning & Memory(Cold Spring Harbor, N.Y.), 1(2), 106–120. http://doi.org/10.1101/lm.1.2.106

class tasks.weather.**RewardWeatherDiff**(***kwargs*)
Bases: *model.modelTemplate.Rewards*

Processes the weather reward for models expecting reward corrections

processFeedback(*feedback*, *lastAction*, *stimuli*)

Returns

Return type modelFeedback

class tasks.weather.**RewardWeatherDualCorrection**(***kwargs*)
Bases: *model.modelTemplate.Rewards*

Processes the decks reward for models expecting the reward correction from two possible actions.

epsilon = 1

processFeedback(*feedback*, *lastAction*, *stimuli*)

Returns

Return type modelFeedback

class tasks.weather.**RewardsWeatherDirect**(***kwargs*)
Bases: *model.modelTemplate.Rewards*

Processes the weather reward for models expecting the reward feedback

processFeedback(*feedback*, *lastAction*, *stimuli*)

Returns

Return type modelFeedback

class tasks.weather.**StimulusWeatherDirect**(***kwargs*)
Bases: *model.modelTemplate.Stimulus*

Processes the weather stimuli for models expecting just the event

processStimulus(*observation*)
Processes the decks stimuli for models expecting just the event

Returns

- **stimuliPresent** (*int or list of int*) – The elements present of the stimulus
- **stimuliActivity** (*float or list of float*) – The activity of each of the elements

class tasks.weather.**Weather**(*cueProbs=[[0.2, 0.8, 0.2, 0.8], [0.8, 0.2, 0.8, 0.2]], learningLen=200, testLen=100, number_cues=None, cues=None, actualities=None*)
Bases: *tasks.taskTemplate.Task*

Based on the 1994 paper "Probabilistic classification learning in amnesia."

Many methods are inherited from the tasks.taskTemplate.Task class. Refer to its documentation for missing methods.

Name
The name of the class used when recording what has been used.

Type string

Parameters

- **cueProbs** (*array of int, optional*) – If generating data, the likelihood of each cue being associated with each actuality. Each row of the array describes one actuality, with each column representing one cue. Each column is assumed sum to 1
- **number_cues** (*int, optional*) – The number of cues
- **learningLen** (*int, optional*) – The number of trials in the learning phase. Default is 200
- **testLen** (*int, optional*) – The number of trials in the test phase. Default is 100
- **actualities** (*array of int, optional*) – The actual reality the cues pointed to; the correct response the participant is trying to get correct
- **cues** (*array of floats, optional*) – The stimulus cues used to guess the actualities

defaultCueProbs = [[0.2, 0.8, 0.2, 0.8], [0.8, 0.2, 0.8, 0.2]]

feedback()
Feedback to the action from the participant

next()
Produces the next stimulus for the iterator

Returns

- **stimulus** (*Tuple*) – The current cues
- **nextValidActions** (Tuple of ints or None) – The list of valid actions that the model can respond with. Set to (0,1), as they never vary.

Raises StopIteration

proceed()
Updates the task after feedback

receiveAction(*action*)
Receives the next action from the participant

Parameters **action** (*int or string*) – The action taken by the model

returnTaskState()
Returns all the relevant data for this task run

Returns **results** – A dictionary containing the class parameters as well as the other useful data

Return type dictionary

storeState()
Stores the state of all the important variables so that they can be output later

tasks.weather.**genActualities**(*cueProbs*, *cues*, *learningLen*, *testLen*)

Parameters

- **cueProbs** –
- **cues** –
- **learningLen** –
- **testLen** –

Returns

Return type actions

tasks.weather.**genCues**(*number_cues*, *taskLen*)

Parameters

- **cueProbs** –
- **taskLen** –

Returns

Return type cues

6.6 modelGenerator module

6.6.1 modelGenerator Module

Author Dominic Hunt

6.6.1.1 Classes

`Model`([number_actions, number_cues, ...])	The model class is a general template for a model.
`ModelGen`(model_name[, parameters, other_options])	Generates model class instances based on a model and a set of varying parameters
`Rewards`(**kwargs)	This acts as an interface between the feedback from a task and the feedback a model can process
`Stimulus`(**kwargs)	Stimulus processor class.

ModelGen

class modelGenerator.**ModelGen**(*model_name*, *parameters=None*, *other_options=None*)

Bases: `object`

Generates model class instances based on a model and a set of varying parameters

Parameters

- **model_name** (*string*) – The name of the file where a model.modelTemplate.Model class can be found
- **parameters** (*dictionary containing floats or lists of floats, optional*) – Parameters are the options that you are or are likely to change across model instances. When a parameter contains a list, an instance of the model will be created for every combination of this parameter with all the others. Default `None`
- **other_options** (*dictionary of float, string or binary valued elements, optional*) – These contain all the the model options that define the version of the model being studied. Default `None`

Methods Summary

`iter_details`()	Yields a list containing a model object and parameters to initialise them
`next`()	Produces the next item for the iterator

Methods Documentation

iter_details()

Yields a list containing a model object and parameters to initialise them

Returns

- **model** (*model.modelTemplate.Model*) – The model to be initialised
- **parameters** (*ordered dictionary of floats or bools*) – The model instance parameters
- **other_options** (*dictionary of floats, strings and binary values*)

next()
Produces the next item for the iterator

Returns models

Return type list of model.model.model instances

6.6.1.2 Class Inheritance Diagram

Author Dominic Hunt

class modelGenerator.**ModelGen**(*model_name*, *parameters=None*, *other_options=None*)
Bases: object

Generates model class instances based on a model and a set of varying parameters

Parameters

- **model_name** (*string*) – The name of the file where a model.modelTemplate.Model class can be found
- **parameters** (*dictionary containing floats or lists of floats, optional*) – Parameters are the options that you are or are likely to change across model instances. When a parameter contains a list, an instance of the model will be created for every combination of this parameter with all the others. Default None
- **other_options** (*dictionary of float, string or binary valued elements, optional*) – These contain all the the model options that define the version of the model being studied. Default None

iter_details()
Yields a list containing a model object and parameters to initialise them

Returns

- **model** (*model.modelTemplate.Model*) – The model to be initialised
- **parameters** (*ordered dictionary of floats or bools*) – The model instance parameters
- **other_options** (*dictionary of floats, strings and binary values*)

next()
Produces the next item for the iterator

Returns models

Return type list of model.model.model instances

6.7 model package

6.7.1 Subpackages

6.7.1.1 model.decision package

Submodules

model.decision.binary module

model.decision.binary Module

Author Dominic Hunt

A collection of decision making functions where there are only two possible actions

Functions

single([task_responses])	Decisions using a switching probability

single

model.decision.binary.**single**(*task_responses=(0, 1)*)
Decisions using a switching probability

Parameters **task_responses** (*tuple of length two, optional*) – Provides the two action responses expected by the task

Returns

- **decision_function** (*function*) – Calculates the decisions based on the probabilities and returns the decision and the probability of that decision
- **decision** (*int or None*) – The action to be taken by the model
- **probabilities** (*OrderedDict of valid responses*) – A dictionary of considered actions as keys and their associated probabilities as values

Examples

```
>>> np.random.seed(100)
>>> dec = single()
>>> dec(0.23)
(0, OrderedDict([(0, 0.77), (1, 0.23)]))
>>> dec(0.23, 0)
(0, OrderedDict([(0, 0.77), (1, 0.23)]))
```

Author Dominic Hunt

A collection of decision making functions where there are only two possible actions

model.decision.binary.**single**(*task_responses=(0, 1)*)
Decisions using a switching probability

Parameters **task_responses** (*tuple of length two, optional*) – Provides the two action responses expected by the task

Returns

- **decision_function** (*function*) – Calculates the decisions based on the probabilities and returns the decision and the probability of that decision
- **decision** (*int or None*) – The action to be taken by the model
- **probabilities** (*OrderedDict of valid responses*) – A dictionary of considered actions as keys and their associated probabilities as values

Examples

```
>>> np.random.seed(100)
>>> dec = single()
>>> dec(0.23)
(0, OrderedDict([(0, 0.77), (1, 0.23)]))
>>> dec(0.23, 0)
(0, OrderedDict([(0, 0.77), (1, 0.23)]))
```

model.decision.discrete module

model.decision.discrete Module

Author Dominic Hunt

A collection of decision making functions where there are no limits on the number of actions, but they are countable.

Functions

maxProb([task_responses])	Decisions for an arbitrary number of choices
probThresh([task_responses, eta])	Decisions for an arbitrary number of choices
weightProb([task_responses])	Decisions for an arbitrary number of choices

maxProb

model.decision.discrete.**maxProb**(*task_responses=(0, 1)*)

Decisions for an arbitrary number of choices

Choice made by choosing the most likely

Parameters **task_responses** (*tuple*) – Provides the action responses expected by the tasks for each probability estimate.

Returns

- **decision_function** (*function*) – Calculates the decisions based on the probabilities and returns the decision and the probability of that decision
- **decision** (*int or None*) – The action to be taken by the model
- **probDict** (*OrderedDict of valid responses*) – A dictionary of considered actions as keys and their associated probabilities as values

See also:

models.QLearn(),models.QLearn2(),models.OpAL()

Examples

```
>>> np.random.seed(100)
>>> d = maxProb([1,2,3])
>>> d([0.6, 0.3, 0.5])
(1, OrderedDict([(1, 0.6), (2, 0.3), (3, 0.5)]))
>>> d([0.2, 0.3, 0.5], trial_responses=[1, 2])
(2, OrderedDict([(1, 0.2), (2, 0.3), (3, 0.5)]))
>>> d([0.2, 0.3, 0.5], trial_responses=[])
```

(continues on next page)

(continued from previous page)

```
(None, OrderedDict([(1, 0.2), (2, 0.3), (3, 0.5)]))
>>> d = maxProb(["A", "B", "C"])
>>> d([0.6, 0.3, 0.5], trial_responses=["A", "B"])
('A', OrderedDict([('A', 0.6), ('B', 0.3), ('C', 0.5)]))
```

probThresh

model.decision.discrete.**probThresh**(*task_responses=(0, 1)*, *eta=0.8*)

Decisions for an arbitrary number of choices

Choice made by choosing when certain (when probability above a certain value), otherwise randomly

Parameters

- **task_responses** (*tuple*) – Provides the action responses expected by the tasks for each probability estimate.
- **eta** (*float, optional*) – The value above which a non-random decision is made. Default value is 0.8

Returns

- **decision_function** (*function*) – Calculates the decisions based on the probabilities and returns the decision and the probability of that decision
- **decision** (*int or None*) – The action to be taken by the model
- **probDict** (*OrderedDict of valid responses*) – A dictionary of considered actions as keys and their associated probabilities as values

Examples

```
>>> np.random.seed(100)
>>> d = probThresh(task_responses=[0, 1, 2, 3], eta=0.8)
>>> d([0.2, 0.8, 0.3, 0.5])
(1, OrderedDict([(0, 0.2), (1, 0.8), (2, 0.3), (3, 0.5)]))
>>> d([0.2, 0.8, 0.3, 0.5], trial_responses=[0, 2])
(0, OrderedDict([(0, 0.2), (1, 0.8), (2, 0.3), (3, 0.5)]))
>>> d([0.2, 0.8, 0.3, 0.5], trial_responses=[])
(None, OrderedDict([(0, 0.2), (1, 0.8), (2, 0.3), (3, 0.5)]))
>>> d = probThresh(["A","B","C"])
>>> d([0.2, 0.3, 0.8], trial_responses=["A", "B"])
('A', OrderedDict([('A', 0.2), ('B', 0.3), ('C', 0.8)]))
```

weightProb

model.decision.discrete.**weightProb**(*task_responses=(0, 1)*)

Decisions for an arbitrary number of choices

Choice made by choosing randomly based on which are valid and what their associated probabilities are

Parameters **task_responses** (*tuple*) – Provides the action responses expected by the task for each probability estimate.

Returns

- **decision_function** (*function*) – Calculates the decisions based on the probabilities and returns the decision and the probability of that decision
- **decision** (*int or None*) – The action to be taken by the model

- **probDict** (*OrderedDict of valid responses*) – A dictionary of considered actions as keys and their associated probabilities as values

See also:

`models.QLearn()`,`models.QLearn2()`,`models.OpAL()`

Examples

```
>>> np.random.seed(100)
>>> d = weightProb([0, 1, 2, 3])
>>> d([0.4, 0.8, 0.3, 0.5])
(1, OrderedDict([(0, 0.2), (1, 0.4), (2, 0.15), (3, 0.25)]))
>>> d([0.1, 0.3, 0.4, 0.2])
(1, OrderedDict([(0, 0.1), (1, 0.3), (2, 0.4), (3, 0.2)]))
>>> d([0.2, 0.5, 0.3, 0.5], trial_responses=[0, 2])
(2, OrderedDict([(0, 0.4), (1, 0), (2, 0.6), (3, 0)]))
>>> d = weightProb(["A", "B", "C"])
>>> d([0.2, 0.3, 0.5], trial_responses=["A", "B"])
(u'B', OrderedDict([(u'A', 0.4), (u'B', 0.6), (u'C', 0)]))
>>> d([0.2, 0.3, 0.5], trial_responses=[])
(None, OrderedDict([(u'A', 0.2), (u'B', 0.3), (u'C', 0.5)]))
```

Author Dominic Hunt

A collection of decision making functions where there are no limits on the number of actions, but they are countable.

`model.decision.discrete.`**`maxProb`**(*task_responses=(0, 1)*)

Decisions for an arbitrary number of choices

Choice made by choosing the most likely

Parameters **`task_responses`** (*`tuple`*) – Provides the action responses expected by the tasks for each probability estimate.

Returns

- **decision_function** (*function*) – Calculates the decisions based on the probabilities and returns the decision and the probability of that decision
- **decision** (*int or None*) – The action to be taken by the model
- **probDict** (*OrderedDict of valid responses*) – A dictionary of considered actions as keys and their associated probabilities as values

See also:

`models.QLearn()`,`models.QLearn2()`,`models.OpAL()`

Examples

```
>>> np.random.seed(100)
>>> d = maxProb([1,2,3])
>>> d([0.6, 0.3, 0.5])
(1, OrderedDict([(1, 0.6), (2, 0.3), (3, 0.5)]))
>>> d([0.2, 0.3, 0.5], trial_responses=[1, 2])
(2, OrderedDict([(1, 0.2), (2, 0.3), (3, 0.5)]))
>>> d([0.2, 0.3, 0.5], trial_responses=[])
(None, OrderedDict([(1, 0.2), (2, 0.3), (3, 0.5)]))
>>> d = maxProb(["A", "B", "C"])
>>> d([0.6, 0.3, 0.5], trial_responses=["A", "B"])
('A', OrderedDict([('A', 0.6), ('B', 0.3), ('C', 0.5)]))
```

model.decision.discrete.**probThresh**(*task_responses=(0, 1)*, *eta=0.8*)

Decisions for an arbitrary number of choices

Choice made by choosing when certain (when probability above a certain value), otherwise randomly

Parameters

- **task_responses** (*tuple*) – Provides the action responses expected by the tasks for each probability estimate.
- **eta** (*float, optional*) – The value above which a non-random decision is made. Default value is 0.8

Returns

- **decision_function** (*function*) – Calculates the decisions based on the probabilities and returns the decision and the probability of that decision
- **decision** (*int or None*) – The action to be taken by the model
- **probDict** (*OrderedDict of valid responses*) – A dictionary of considered actions as keys and their associated probabilities as values

Examples

```
>>> np.random.seed(100)
>>> d = probThresh(task_responses=[0, 1, 2, 3], eta=0.8)
>>> d([0.2, 0.8, 0.3, 0.5])
(1, OrderedDict([(0, 0.2), (1, 0.8), (2, 0.3), (3, 0.5)]))
>>> d([0.2, 0.8, 0.3, 0.5], trial_responses=[0, 2])
(0, OrderedDict([(0, 0.2), (1, 0.8), (2, 0.3), (3, 0.5)]))
>>> d([0.2, 0.8, 0.3, 0.5], trial_responses=[])
(None, OrderedDict([(0, 0.2), (1, 0.8), (2, 0.3), (3, 0.5)]))
>>> d = probThresh(["A","B","C"])
>>> d([0.2, 0.3, 0.8], trial_responses=["A", "B"])
('A', OrderedDict([('A', 0.2), ('B', 0.3), ('C', 0.8)]))
```

model.decision.discrete.**weightProb**(*task_responses=(0, 1)*)

Decisions for an arbitrary number of choices

Choice made by choosing randomly based on which are valid and what their associated probabilities are

Parameters **task_responses** (*tuple*) – Provides the action responses expected by the task for each probability estimate.

Returns

- **decision_function** (*function*) – Calculates the decisions based on the probabilities and returns the decision and the probability of that decision
- **decision** (*int or None*) – The action to be taken by the model
- **probDict** (*OrderedDict of valid responses*) – A dictionary of considered actions as keys and their associated probabilities as values

See also:

models.QLearn(),models.QLearn2(),models.OpAL()

Examples

```
>>> np.random.seed(100)
>>> d = weightProb([0, 1, 2, 3])
>>> d([0.4, 0.8, 0.3, 0.5])
```

(continues on next page)

(continued from previous page)

```
(1, OrderedDict([(0, 0.2), (1, 0.4), (2, 0.15), (3, 0.25)]))
>>> d([0.1, 0.3, 0.4, 0.2])
(1, OrderedDict([(0, 0.1), (1, 0.3), (2, 0.4), (3, 0.2)]))
>>> d([0.2, 0.5, 0.3, 0.5], trial_responses=[0, 2])
(2, OrderedDict([(0, 0.4), (1, 0), (2, 0.6), (3, 0)]))
>>> d = weightProb(["A", "B", "C"])
>>> d([0.2, 0.3, 0.5], trial_responses=["A", "B"])
(u'B', OrderedDict([(u'A', 0.4), (u'B', 0.6), (u'C', 0)]))
>>> d([0.2, 0.3, 0.5], trial_responses=[])
(None, OrderedDict([(u'A', 0.2), (u'B', 0.3), (u'C', 0.5)]))
```

6.7.2 Submodules

6.7.2.1 model.ACBasic module

Author Dominic Hunt

Reference Based on ideas we had.

class model.ACBasic.**ACBasic**(*alpha=0.3*, *beta=4*, *invBeta=None*, *alphaE=None*, *alphaA=None*, *expect=None*, *actorExpect=None*, ***kwargs*)

Bases: *model.modelTemplate.Model*

A basic, complete actor-critic model

Name

The name of the class used when recording what has been used.

Type string

Parameters

- **alpha** (*float, optional*) – Learning rate parameter
- **alphaE** (*float, optional*) – Learning rate parameter for the update of the expectations. Default `lpha`
- **alphaA** (*float, optional*) – Learning rate parameter for the update of the actor. Default `lpha`
- **beta** (*float, optional*) – Sensitivity parameter for probabilities
- **invBeta** (*float, optional*) – Inverse of sensitivity parameter. Defined as $\frac{1}{\beta+1}$. Default `0.2`
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`

- **expect** (*array of floats, optional*) – The initialisation of the expected reward. Default `ones((number_actions, number_cues)) * 5 / number_cues`
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns **probabilities** – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters **actionValues** (*1D ndArray of floats*) –

Returns **probArray** – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.2 model.ACE module

Author Dominic Hunt

Reference Based on ideas we had.

class model.ACE.**ACE**(*alpha=0.3*, *epsilon=0.1*, *alphaE=None*, *alphaA=None*, *expect=None*, *actorExpect=None*, ***kwargs*)
Bases: *model.modelTemplate.Model*

A basic, complete actor-critic model with decision making based on QLearnE

Name
The name of the class used when recording what has been used.

Type string

Parameters

- **alpha** (*float, optional*) – Learning rate parameter
- **alphaE** (*float, optional*) – Learning rate parameter for the update of the expectations. Default `lpha`
- **alphaA** (*float, optional*) – Learning rate parameter for the update of the actor. Default `lpha`
- **epsilon** (*float, optional*) – Noise parameter. The larger it is the less likely the model is to choose the highest expected reward
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`
- **expect** (*array of floats, optional*) – The initialisation of the expected reward. Default `ones((number_actions, number_cues)) * 5 / number_cues`

- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns **probabilities** – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters **actionValues** (*1D ndArray of floats*) –

Returns **probArray** – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.3 model.ACES module

Author Dominic Hunt

Reference Based on ideas we had.

class model.ACES.**ACES**(*alpha=0.3*, *epsilon=0.1*, *expect=None*, *actorExpect=None*, ***kwargs*)
Bases: *model.modelTemplate.Model*

A basic, complete actor-critic model with decision making based on QLearnE

Name
The name of the class used when recording what has been used.

Type string

Parameters

- **alpha** (*float, optional*) – Learning rate parameter
- **epsilon** (*float, optional*) – Noise parameter. The larger it is the less likely the model is to choose the highest expected reward
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`
- **expect** (*array of floats, optional*) – The initialisation of the expected reward. Default `ones((number_actions, number_cues)) * 5 / number_cues`
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns **probabilities** – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters **actionValues** (*1D ndArray of floats*) –

Returns **probArray** – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.4 model.BP module

Author Dominic Hunt

class model.BP.**BP**(*alpha=0.3*, *beta=4*, *dirichletInit=1*, *validRewards=array([0, 1])*, *invBeta=None*, ***kwargs*)

Bases: *model.modelTemplate.Model*

The Bayesian predictor model

Name

The name of the class used when recording what has been used.

Type string

Parameters

- **alpha** (*float, optional*) – Learning rate parameter
- **beta** (*float, optional*) – Sensitivity parameter for probabilities. Default 4
- **invBeta** (*float, optional*) – Inverse of sensitivity parameter. Defined as $\frac{1}{\beta+1}$. Default 0.2
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **validRewards** (*list, np.ndarray, optional*) – The different reward values that can occur in the task. Default array([0, 1])
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **dirichletInit** (*float, optional*) – The initial values for values of the dirichlet distribution. Normally 0, 1/2 or 1. Default 1
- **prior** (array of floats in [0, 1], optional) – Ignored in this case
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

actStimMerge(*dirichletVals*, *stimuli*)

actorStimulusProbs()

Calculates in the model-appropriate way the probability of each action.

Returns probabilities – The probabilities associated with the action choices

Return type 1D np.ndArray of floats

calcActExpectations(*dirichletVals*)

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters **actionValues** (*1D np.ndArray of floats*) –

Returns **probArray** – The probabilities associated with the actionValues

Return type 1D np.ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateExpectations(*dirichletVals*)

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.5 model.BPE module

Author Dominic Hunt

class model.BPE.**BPE**(*alpha=0.3*, *epsilon=0.1*, *dirichletInit=1*, *validRewards=array([0, 1])*, ***kwargs*)

Bases: *model.modelTemplate.Model*

The Bayesian predictor model

Name

The name of the class used when recording what has been used.

Type string

Parameters

- **alpha** (*float, optional*) – Learning rate parameter
- **epsilon** (*float, optional*) – Noise parameter. The larger it is the less likely the model is to choose the highest expected reward
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **validRewards** (*list,np.ndarray, optional*) – The different reward values that can occur in the task. Default `array([0, 1])`
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **dirichletInit** (*float, optional*) – The initial values for values of the dirichlet distribution. Normally 0, 1/2 or 1. Default 1
- **prior** (array of floats in `[0, 1]`, optional) – Ignored in this case
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

See also:

model.BP This model is heavily based on that one

actStimMerge(*dirichletVals*, *stimuli*)

actorStimulusProbs()

Calculates in the model-appropriate way the probability of each action.

Returns **probabilities** – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcActExpectations(*dirichletVals*)

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters **actionValues** (*1D ndArray of floats*) –

Returns **probArray** – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateExpectations(*dirichletVals*)

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.6 model.BPV module

Author Dominic Hunt

class model.BPV.**BPV**(*alpha=0.3*, *dirichletInit=1*, *validRewards=array([0, 1])*, ***kwargs*)
Bases: *model.modelTemplate.Model*

The Bayesian predictor model

Name
The name of the class used when recording what has been used.

Type string

Parameters

- **alpha** (*float, optional*) – Learning rate parameter
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **validRewards** (*list, np.ndarray, optional*) – The different reward values that can occur in the task. Default `array([0, 1])`
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **dirichletInit** (*float, optional*) – The initial values for values of the dirichlet distribution. Normally 0, 1/2 or 1. Default 1
- **prior** (array of floats in `[0, 1]`, optional) – Ignored in this case
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

actStimMerge(*dirichletVals*, *stimuli*)

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns **probabilities** – The probabilities associated with the action choices

Return type 1D np.ndarray of floats

calcActExpectations(*dirichletVals*)

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters **actionValues** (*1D np.ndarray of floats*) –

Returns **probArray** – The probabilities associated with the actionValues

Return type 1D np.ndarray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateExpectations(*dirichletVals*)

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.7 model.OpAL module

Author Dominic Hunt

Reference Based on the paper Opponent actor learning (OpAL): Modeling interactive effects of striatal dopamine on reinforcement learning and choice incentive. Collins, A. G. E., & Frank, M. J. (2014). Psychological Review, 121(3), 337–66. doi:10.1037/a0037015

class model.OpAL.**OpAL**(*alpha=0.3*, *beta=4*, *rho=0*, *invBeta=None*, *alphaCrit=None*, *betaGo=None*, *betaNogo=None*, *alphaGo=None*, *alphaNogo=None*, *alphaGoDiff=None*, *alphaNogoDiff=None*, *alphaGoNogoDiff=None*, *expect=None*, *expectGo=None*, ***kwargs*)

Bases: *model.modelTemplate.Model*

The Opponent actor learning model

Name

The name of the class used when recording what has been used.

Type string

currAction

The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **alpha** (*float, optional*) – Learning rate parameter, used as either the
- **alphaGoNogoDiff** (*float, optional*) – The difference between `alphaGo` and `alphaNogo`. Default is `None`. If not `None` will overwrite `alphaNogo` $\alpha_N = \alpha_G - \alpha_\delta$
- **alphaCrit** (*float, optional*) – The critic learning rate. Default is `alpha`
- **alphaGo** (*float, optional*) – Learning rate parameter for Go, the positive part of the actor learning Default is `alpha`
- **alphaNogo** (*float, optional*) – Learning rate parameter for Nogo, the negative part of the actor learning Default is `alpha`
- **alphaGoDiff** (*float, optional*) – The difference between `alphaCrit` and `alphaGo`. The default is `None` If not `None` and `alphaNogoDiff` is also not `None`, it will overwrite the `alphaGo` parameter $\alpha_G = \alpha_C + \alpha$
- **alphaNogoDiff** (*float, optional*) – The difference between `alphaCrit` and `alphaNogo`. The default is `None` If not `None` and `alphaGoDiff` is also not `None`, it will overwrite the `alphaNogo` parameter $\alpha_N = \alpha_C + \alpha$
- **beta** (*float, optional*) – Sensitivity parameter for probabilities. Also known as an exploration- exploitation parameter. Defined as β in the paper
- **invBeta** (*float, optional*) – Inverse of sensitivity parameter for the probabilities. Defined as $\frac{1}{\beta+1}$. Default `0.2`
- **rho** (*float, optional*) – The asymmetry between the actor weights. $\rho = \beta_G - \beta = \beta_N + \beta$
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.

- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`
- **expect** (*array of floats, optional*) – The initialisation of the the expected reward. Default `ones((number_actions, number_cues)) / number_critics`
- **expectGo** (*array of floats, optional*) – The initialisation of the the expected go and nogo. Default `ones((number_actions, number_cues)) / number_critics`
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

Notes

Actor: The chosen action is updated with

$$\delta_{d,t} = r_t - E_{d,t}$$
$$E_{d,t+1} = E_{d,t} + \alpha_E \delta_{d,t}$$

Critic: The chosen action is updated with

$$G_{d,t+1} = G_{d,t} + \alpha_G G_{d,t} \delta_{d,t}$$
$$N_{d,t+1} = N_{d,t} - \alpha_N N_{d,t} \delta_{d,t}$$

Probabilities: The probabilities for all actions are calculated using

$$A_{d,t} = (1+\rho) G_{d,t} - (1-\rho) N_{d,t}$$
$$P_{d,t} = \frac{e^{\beta A_{d,t}}}{\sum_{d \in D} e^{\beta A_{d,t}}}$$

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns probabilities – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters actionValues (*1D ndArray of floats*) –

Returns probArray – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value

- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.8 model.OpALE module

Author Dominic Hunt

Reference Based on the paper Opponent actor learning (OpAL): Modeling interactive effects of striatal dopamine on reinforcement learning and choice incentive. Collins, A. G. E., & Frank, M. J. (2014). Psychological Review, 121(3), 337–66. doi:10.1037/a0037015

class model.OpALE.**OpALE**(*alpha=0.3*, *epsilon=0.3*, *rho=0*, *alphaCrit=None*, *alphaGo=None*, *alphaNogo=None*, *alphaGoDiff=None*, *alphaNogoDiff=None*, *alphaGoNogoDiff=None*, *expect=None*, *expectGo=None*, ***kwargs*)
Bases: *model.modelTemplate.Model*

The Opponent actor learning model

Name
The name of the class used when recording what has been used.

Type string

currAction
The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **alpha** (*float, optional*) – Learning rate parameter, used as either the
- **alphaGoNogoDiff** (*float, optional*) – The difference between `alphaGo` and `alphaNogo`. Default is `None`. If not `None` will overwrite `alphaNogo` $\alpha_N = \alpha_G - \alpha_\delta$
- **alphaCrit** (*float, optional*) – The critic learning rate. Default is `alpha`
- **alphaGo** (*float, optional*) – Learning rate parameter for Go, the positive part of the actor learning Default is `alpha`
- **alphaNogo** (*float, optional*) – Learning rate parameter for Nogo, the negative part of the actor learning Default is `alpha`
- **alphaGoDiff** (*float, optional*) – The difference between `alphaCrit` and `alphaGo`. The default is `None` If not `None` and `alphaNogoDiff` is also not `None`, it will overwrite the `alphaGo` parameter $\alpha_G = \alpha_C + \alpha$
- **alphaNogoDiff** (*float, optional*) – The difference between `alphaCrit` and `alphaNogo`. The default is `None` If not `None` and `alphaGoDiff` is also not `None`, it will overwrite the `alphaNogo` parameter $\alpha_N = \alpha_C + \alpha$
- **epsilon** (*float, optional*) – Sensitivity parameter for probabilities. Also known as an exploration- exploitation parameter. Defined as ϵ in the paper
- **rho** (*float, optional*) – The asymmetry between the actor weights. $\rho = \epsilon_G - \epsilon = \epsilon_N + \epsilon$
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`
- **expect** (*array of floats, optional*) – The initialisation of the the expected reward. Default `ones((number_actions, number_cues)) / number_critics`
- **expectGo** (*array of floats, optional*) – The initialisation of the the expected go and nogo. Default `ones((number_actions, number_cues)) / number_critics`
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew

- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

Notes

Actor: The chosen action is updated with

$$\delta_{d,t} = r_t - E_{d,t}$$
$$E_{d,t+1} = E_{d,t} + \alpha_E \delta_{d,t}$$

Critic: The chosen action is updated with

$$G_{d,t+1} = G_{d,t} + \alpha_G G_{d,t} \delta_{d,t}$$
$$N_{d,t+1} = N_{d,t} - \alpha_N N_{d,t} \delta_{d,t}$$

Probabilities: The probabilities for all actions are calculated using

$$A_{d,t} = (1+\rho) G_{d,t} - (1-\rho) N_{d,t}$$
$$P_{d,t} = \frac{e^{\epsilon A_{d,t}}}{\sum_{d \in D} e^{\epsilon A_{d,t}}}$$

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns probabilities – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters actionValues (*1D ndArray of floats*) –

Returns probArray – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns results – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.9 model.OpALS module

Author Dominic Hunt

Reference Based on the paper Opponent actor learning (OpAL): Modeling interactive effects of striatal dopamine on reinforcement learning and choice incentive. Collins, A. G. E., & Frank, M. J. (2014). Psychological Review, 121(3), 337–66. doi:10.1037/a0037015

class model.OpALS.**OpALS**(*alpha=0.3*, *beta=4*, *rho=0*, *saturateVal=10*, *invBeta=None*, *alphaCrit=None*, *betaGo=None*, *betaNogo=None*, *alphaGo=None*, *alphaNogo=None*, *alphaGoDiff=None*, *alphaNogoDiff=None*, *alphaGoNogoDiff=None*, *expect=None*, *expectGo=None*, ***kwargs*)
Bases: *model.modelTemplate.Model*

The Opponent actor learning model modified to have saturation values

The saturation values are the same for the actor and critic learners

Name
The name of the class used when recording what has been used.

Type string

currAction
The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **alpha** (*float, optional*) – Learning rate parameter, used as either the
- **alphaGoNogoDiff** (*float, optional*) – The difference between alphaGo and alphaNogo. Default is None. If not None will overwrite alphaNogo $\alpha_N = \alpha_G - \alpha_\delta$
- **alphaCrit** (*float, optional*) – The critic learning rate. Default is alpha
- **alphaGo** (*float, optional*) – Learning rate parameter for Go, the positive part of the actor learning Default is alpha
- **alphaNogo** (*float, optional*) – Learning rate parameter for Nogo, the negative part of the actor learning Default is alpha

- **alphaGoDiff** (*float, optional*) – The difference between `alphaCrit` and `alphaGo`. The default is `None` If not `None` and `alphaNogoDiff` is also not `None`, it will overwrite the `alphaGo` parameter $\alpha_G = \alpha_C + \alpha$
- **alphaNogoDiff** (*float, optional*) – The difference between `alphaCrit` and `alphaNogo`. The default is `None` If not `None` and `alphaGoDiff` is also not `None`, it will overwrite the `alphaNogo` parameter $\alpha_N = \alpha_C + \alpha$
- **beta** (*float, optional*) – Sensitivity parameter for probabilities. Also known as an exploration- exploitation parameter. Defined as β in the paper
- **invBeta** (*float, optional*) – Inverse of sensitivity parameter for the probabilities. Defined as $\frac{1}{\beta+1}$. Default `0.2`
- **rho** (*float, optional*) – The asymmetry between the actor weights. $\rho = \beta_G - \beta = \beta_N + \beta$
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`
- **expect** (*array of floats, optional*) – The initialisation of the the expected reward. Default `ones((number_actions, number_cues)) / number_critics`
- **expectGo** (*array of floats, optional*) – The initialisation of the the expected go and nogo. Default `ones((number_actions, number_cues)) / number_critics`
- **saturateVal** (*float, optional*) – The saturation value for the model. Default is 10
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

Notes

Actor: The chosen action is updated with

$$\delta_{d,t} = r_t - E_{d,t}$$

$$E_{d,t+1} = E_{d,t} + \alpha_E \delta_{d,t}(1 - \frac{E_{d,t}}{S})$$

Critic: The chosen action is updated with

$$G_{d,t+1} = G_{d,t} + \alpha_G G_{d,t} \delta_{d,t} (1 - \frac{G_{d,t}}{S})$$
$$N_{d,t+1} = N_{d,t} - \alpha_N N_{d,t} \delta_{d,t} (1 - \frac{N_{d,t}}{S})$$

Probabilities: The probabilities for all actions are calculated using

$$A_{d,t} = (1 + \rho) G_{d,t} - (1 - \rho) N_{d,t}$$
$$P_{d,t} = \frac{e^{\beta A_{d,t}}}{\sum_{d \in D} e^{\beta A_{d,t}}}$$

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns **probabilities** – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters **actionValues** (*1D ndArray of floats*) –

Returns **probArray** – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.10 model.OpALSE module

Author Dominic Hunt

Reference Based on the paper Opponent actor learning (OpAL): Modeling interactive effects of striatal dopamine on reinforcement learning and choice incentive. Collins, A. G. E., & Frank, M. J. (2014). Psychological Review, 121(3), 337–66. doi:10.1037/a0037015

class model.OpALSE.**OpALSE**(*alpha=0.3*, *epsilon=0.3*, *rho=0*, *saturateVal=10*, *alphaCrit=None*, *alphaGo=None*, *alphaNogo=None*, *alphaGoDiff=None*, *alphaNogoDiff=None*, *alphaGoNogoDiff=None*, *expect=None*, *expectGo=None*, ***kwargs*)
Bases: *model.modelTemplate.Model*

The Opponent actor learning model modified to have saturation values

The saturation values are the same for the actor and critic learners

Name
The name of the class used when recording what has been used.

Type string

currAction
The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **alpha** (*float, optional*) – Learning rate parameter, used as either the
- **alphaGoNogoDiff** (*float, optional*) – The difference between `alphaGo` and `alphaNogo`. Default is `None`. If not `None` will overwrite `alphaNogo` $\alpha_N = \alpha_G - \alpha_\delta$
- **alphaCrit** (*float, optional*) – The critic learning rate. Default is `alpha`
- **alphaGo** (*float, optional*) – Learning rate parameter for Go, the positive part of the actor learning Default is `alpha`
- **alphaNogo** (*float, optional*) – Learning rate parameter for Nogo, the negative part of the actor learning Default is `alpha`
- **alphaGoDiff** (*float, optional*) – The difference between `alphaCrit` and `alphaGo`. The default is `None` If not `None` and `alphaNogoDiff` is also not `None`, it will overwrite the `alphaGo` parameter $\alpha_G = \alpha_C + \alpha$
- **alphaNogoDiff** (*float, optional*) – The difference between `alphaCrit` and `alphaNogo`. The default is `None` If not `None` and `alphaGoDiff` is also not `None`, it will overwrite the `alphaNogo` parameter $\alpha_N = \alpha_C + \alpha$

- **epsilon** (*float, optional*) – Sensitivity parameter for probabilities. Also known as an exploration- exploitation parameter. Defined as ϵ in the paper
- **rho** (*float, optional*) – The asymmetry between the actor weights. $\rho = \epsilon_G - \epsilon = \epsilon_N + \epsilon$
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`
- **expect** (*array of floats, optional*) – The initialisation of the the expected reward. Default `ones((number_actions, number_cues)) / number_critics`
- **expectGo** (*array of floats, optional*) – The initialisation of the the expected go and nogo. Default `ones((number_actions, number_cues)) / number_critics`
- **saturateVal** (*float, optional*) – The saturation value for the model. Default is 10
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

Notes

Actor: The chosen action is updated with

$$\delta_{d,t} = r_t - E_{d,t}$$
$$E_{d,t+1} = E_{d,t} + \alpha_E \delta_{d,t}(1 - \frac{E_{d,t}}{S})$$

Critic: The chosen action is updated with

$$G_{d,t+1} = G_{d,t} + \alpha_G G_{d,t} \delta_{d,t}(1 - \frac{G_{d,t}}{S})$$
$$N_{d,t+1} = N_{d,t} - \alpha_N N_{d,t} \delta_{d,t}(1 - \frac{N_{d,t}}{S})$$

Probabilities: The probabilities for all actions are calculated using

$$A_{d,t} = (1+\rho)G_{d,t} - (1-\rho)N_{d,t}$$
$$P_{d,t} = \frac{e^{\epsilon A_{d,t}}}{\sum_{d \in D} e^{\epsilon A_{d,t}}}$$

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns **probabilities** – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters **actionValues** (*1D ndArray of floats*) –

Returns **probArray** – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward

- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.11 model.OpAL_H module

Author Dominic Hunt

Reference Based on the paper Opponent actor learning (OpAL): Modeling interactive effects of striatal dopamine on reinforcement learning and choice incentive. Collins, A. G. E., & Frank, M. J. (2014). Psychological Review, 121(3), 337–66. doi:10.1037/a0037015

class model.OpAL_H.**OpAL_H**(*alpha=0.3*, *beta=4*, *rho=0*, *invBeta=None*, *alphaCrit=None*, *betaGo=None*, *betaNogo=None*, *alphaGo=None*, *alphaNogo=None*, *alphaGoDiff=None*, *alphaNogoDiff=None*, *alphaGoNogoDiff=None*, *expect=None*, *expectGo=None*, ***kwargs*)

Bases: *model.modelTemplate.Model*

The Opponent actor learning model without Hebbian learning

Name
The name of the class used when recording what has been used.

Type string

currAction
The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **alpha** (*float, optional*) – Learning rate parameter, used as either the
- **alphaGoNogoDiff** (*float, optional*) – The difference between `alphaGo` and `alphaNogo`. Default is `None`. If not `None` will overwrite `alphaNogo` $\alpha_N = \alpha_G - \alpha_\delta$
- **alphaCrit** (*float, optional*) – The critic learning rate. Default is `alpha`
- **alphaGo** (*float, optional*) – Learning rate parameter for Go, the positive part of the actor learning Default is `alpha`
- **alphaNogo** (*float, optional*) – Learning rate parameter for Nogo, the negative part of the actor learning Default is `alpha`
- **alphaGoDiff** (*float, optional*) – The difference between `alphaCrit` and `alphaGo`. The default is `None` If not `None` and `alphaNogoDiff` is also not `None`, it will overwrite the `alphaGo` parameter $\alpha_G = \alpha_C + \alpha$
- **alphaNogoDiff** (*float, optional*) – The difference between `alphaCrit` and `alphaNogo`. The default is `None` If not `None` and `alphaGoDiff` is also not `None`, it will overwrite the `alphaNogo` parameter $\alpha_N = \alpha_C + \alpha$
- **beta** (*float, optional*) – Sensitivity parameter for probabilities. Also known as an exploration- exploitation parameter. Defined as β in the paper
- **invBeta** (*float, optional*) – Inverse of sensitivity parameter for the probabilities. Defined as $\frac{1}{\beta+1}$. Default `0.2`
- **rho** (*float, optional*) – The asymmetry between the actor weights. $\rho = \beta_G - \beta = \beta_N + \beta$
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.

- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.

- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues

- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.

- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`

- **expect** (*array of floats, optional*) – The initialisation of the the expected reward. Default `ones((number_actions, number_cues)) / number_critics`

- **expectGo** (*array of floats, optional*) – The initialisation of the the expected go and nogo. Default `ones((number_actions, number_cues)) / number_critics`

- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim

- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew

- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

Notes

Actor: The chosen action is updated with

$$\delta_{d,t} = r_t - E_{d,t}$$
$$E_{d,t+1} = E_{d,t} + \alpha_E \delta_{d,t}$$

Critic: The chosen action is updated with

$$G_{d,t+1} = G_{d,t} + \alpha_G \delta_{d,t}$$
$$N_{d,t+1} = N_{d,t} - \alpha_N \delta_{d,t}$$

Probabilities: The probabilities for all actions are calculated using

$$A_{d,t} = (1+\rho) G_{d,t} - (1-\rho) N_{d,t}$$
$$P_{d,t} = \frac{e^{\beta A_{d,t}}}{\sum_{d \in D} e^{\beta A_{d,t}}}$$

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns probabilities – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters **actionValues** (*1D ndArray of floats*) –

Returns **probArray** – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.12 model.OpAL_HE module

Author Dominic Hunt

Reference Based on the paper Opponent actor learning (OpAL): Modeling interactive effects of striatal dopamine on reinforcement learning and choice incentive. Collins, A. G. E., & Frank, M. J. (2014). Psychological Review, 121(3), 337–66. doi:10.1037/a0037015

class model.OpAL_HE.**OpAL_HE**(*alpha=0.3*, *epsilon=0.3*, *rho=0*, *alphaCrit=None*, *alphaGo=None*, *alphaNogo=None*, *alphaGoDiff=None*, *alphaNogoDiff=None*, *alphaGoNogoDiff=None*, *expect=None*, *expectGo=None*, ***kwargs*)

Bases: *model.modelTemplate.Model*

The Opponent actor learning model without Hebbian learning

Name

The name of the class used when recording what has been used.

Type string

currAction

The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **alpha** (*float, optional*) – Learning rate parameter, used as either the
- **alphaGoNogoDiff** (*float, optional*) – The difference between `alphaGo` and `alphaNogo`. Default is `None`. If not `None` will overwrite `alphaNogo` $\alpha_N = \alpha_G - \alpha_\delta$
- **alphaCrit** (*float, optional*) – The critic learning rate. Default is `alpha`
- **alphaGo** (*float, optional*) – Learning rate parameter for Go, the positive part of the actor learning Default is `alpha`
- **alphaNogo** (*float, optional*) – Learning rate parameter for Nogo, the negative part of the actor learning Default is `alpha`
- **alphaGoDiff** (*float, optional*) – The difference between `alphaCrit` and `alphaGo`. The default is `None` If not `None` and `alphaNogoDiff` is also not `None`, it will overwrite the `alphaGo` parameter $\alpha_G = \alpha_C + \alpha$
- **alphaNogoDiff** (*float, optional*) – The difference between `alphaCrit` and `alphaNogo`. The default is `None` If not `None` and `alphaGoDiff` is also not `None`, it will overwrite the `alphaNogo` parameter $\alpha_N = \alpha_C + \alpha$
- **epsilon** (*float, optional*) – Sensitivity parameter for probabilities. Also known as an exploration- exploitation parameter. Defined as ϵ in the paper
- **rho** (*float, optional*) – The asymmetry between the actor weights. $\rho = \epsilon_G - \epsilon = \epsilon_N + \epsilon$
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`

- **expect** (*array of floats, optional*) – The initialisation of the the expected reward. Default `ones((number_actions, number_cues)) / number_critics`
- **expectGo** (*array of floats, optional*) – The initialisation of the the expected go and nogo. Default `ones((number_actions, number_cues)) / number_critics`
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

Notes

Actor: The chosen action is updated with

$$\delta_{d,t} = r_t - E_{d,t}$$
$$E_{d,t+1} = E_{d,t} + \alpha_E \delta_{d,t}$$

Critic: The chosen action is updated with

$$G_{d,t+1} = G_{d,t} + \alpha_G \delta_{d,t}$$
$$N_{d,t+1} = N_{d,t} - \alpha_N \delta_{d,t}$$

Probabilities: The probabilities for all actions are calculated using

$$A_{d,t} = (1+\rho)G_{d,t} - (1-\rho)N_{d,t}$$
$$P_{d,t} = \frac{e^{\epsilon A_{d,t}}}{\sum_{d \in D} e^{\epsilon A_{d,t}}}$$

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns probabilities – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters actionValues (*1D ndArray of floats*) –

Returns probArray – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.13 model.modelTemplate module

model.modelTemplate Module

Author Dominic Hunt

Classes

Model([number_actions, number_cues, ...])	The model class is a general template for a model.
Rewards(**kwargs)	This acts as an interface between the feedback from a task and the feedback a model can process
Stimulus(**kwargs)	Stimulus processor class.

Model

class model.modelTemplate.**Model**(*number_actions=2*, *number_cues=1*, *number_critics=None*, *action_codes=None*, *non_action=u'None'*, *prior=None*, *stimulus_shaper=None*, *stimulus_shaper_name=None*, *stimulus_shaper_properties=None*, *reward_shaper=None*, *reward_shaper_name=None*, *reward_shaper_properties=None*, *decision_function=None*, *decision_function_name=None*, *decision_function_properties=None*, ***kwargs*)

Bases: object

The model class is a general template for a model. It also contains universal methods used by all models.

Name
The name of the class used when recording what has been used.

Type string

currAction
The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **prior** (array of floats in `[0,1]`, optional) – The prior probability of of the states being the correct one. Default `ones((self.number_actions, self.number_cues)) / self.number_critics)`
- **stimulus_shaper_name** (*string, optional*) – The name of the function that transforms the stimulus into a form the model can understand and a string to identify it later. `stimulus_shaper` takes priority
- **reward_shaper_name** (*string, optional*) – The name of the function that transforms the reward into a form the model can understand. `rewards_shaper` takes priority
- **decision_function_name** (*string, optional*) – The name of the function that takes the internal values of the model and turns them in to a decision. `decision function` takes priority
- **stimulus_shaper** (*Stimulus class, optional*) – The class that transforms the stimulus into a form the model can understand and a string to identify it later. Default is Stimulus
- **reward_shaper** (*Rewards class, optional*) – The class that transforms the reward into a form the model can understand. Default is Rewards

- **decision_function** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is `weightProb(range(number_actions))`
- **stimulus_shaper_properties** (*list, optional*) – The valid parameters of the function. Used to filter the unlisted keyword arguments Default is `None`
- **reward_shaper_properties** (*list, optional*) – The valid parameters of the function. Used to filter the unlisted keyword arguments Default is `None`
- **decision_function_properties** (*list, optional*) – The valid parameters of the function. Used to filter the unlisted keyword arguments Default is `None`

Methods Summary

actStimMerge(actStimuliParam[, stimFilter])	Takes the parameter to be merged by stimuli and filters it by the stimuli values
action()	Returns the action of the model
actorStimulusProbs()	Calculates in the model-appropriate way the probability of each action.
calcProbabilities(actionValues)	Calculate the probabilities associated with the action
choiceReflection()	Allows the model to update its state once an action has been chosen.
chooseAction(probabilities, lastAction, ...)	Chooses the next action and returns the associated probabilities
delta(reward, expectation, action, stimuli)	Calculates the comparison between the reward and the expectation
feedback(response)	Receives the reaction to the action and processes it
get_name()	
lastChoiceReinforcement()	Allows the model to update the reward expectation for the previous trialstep given the choice made in this trialstep
observe(state)	Receives the latest observation and decides what to do with it
overrideActionChoice(action)	Provides a method for overriding the model action choice.
params()	Returns the parameters of the model
processEvent([action, response])	Integrates the information from a stimulus, action, response set, regardless of which of the three elements are present.
returnTaskState()	Returns all the relevant data for this model
rewardExpectation(stimuli)	Calculate the expected reward for each action based on the stimuli
setsimID(simID)	**param simID**
standardResultOutput()	Returns the relevant data expected from a model as well as the parameters for the current model
storeStandardResults()	Updates the store of standard results found across models
storeState()	Stores the state of all the important variables so that they can be accessed later

Continued on next page

Table 16 – continued from previous page

updateModel(delta, action, stimuli, ...)	**param delta** The difference between the reward and the expected reward

Methods Documentation

actStimMerge(*actStimuliParam*, *stimFilter=1*)
Takes the parameter to be merged by stimuli and filters it by the stimuli values

Parameters

- **actStimuliParam** (*list of floats*) –

  The list of values representing each action stimuli pair, where the stimuli will have their filtered values merged together.

- **stimFilter** (*array of floats or a float, optional*) – The list of active stimuli with their weightings or one weight for all. Default 1

Returns **actionParams** – The parameter values associated with each action

Return type list of floats

action()
Returns the action of the model

Returns **action**

Return type integer or None

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns **probabilities** – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the action

Parameters **actionValues** (*1D ndArray of floats*) –

Returns **probArray** – The probabilities associated with the actionValues

Return type 1D ndArray of floats

choiceReflection()
Allows the model to update its state once an action has been chosen.

chooseAction(*probabilities*, *lastAction*, *events*, *validActions*)
Chooses the next action and returns the associated probabilities

Parameters

- **probabilities** (*list of floats*) – The probabilities associated with each combinations
- **lastAction** (*int*) – The last chosen action
- **events** (*list of floats*) – The stimuli. If probActions is True then this will be unused as the probabilities will already be
- **validActions** (*1D list or array*) – The actions permitted during this trial-step

Returns

- **newAction** (*int*) – The chosen action

- **decProbabilities** (*list of floats*) – The weights for the different actions

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

feedback(*response*)
Receives the reaction to the action and processes it

Parameters **response** (*float*) – The response from the task after an action. Returns without doing anything if the value of response is *None*.

classmethod get_name()

lastChoiceReinforcement()
Allows the model to update the reward expectation for the previous trialstep given the choice made in this trialstep

observe(*state*)
Receives the latest observation and decides what to do with it

There are five possible states: Observation Observation Action Observation Action Feedback Action Feedback Observation Feedback

Parameters **state** (*tuple of ({int | float | tuple},{tuple of int | None})*) – The stimulus from the task followed by the tuple of valid actions. Passes the values onto a processing function, self._updateObservation‘‘.

overrideActionChoice(*action*)
Provides a method for overriding the model action choice. This is used when fitting models to participant actions.

Parameters **action** (*int*) – Action chosen by external source to same situation

params()
Returns the parameters of the model

Returns **parameters**

Return type dictionary

processEvent(*action=None*, *response=None*)
Integrates the information from a stimulus, action, response set, regardless of which of the three elements are present.

Parameters

- **stimuli** (*{int | float | tuple | None}*) – The stimuli received
- **action** (*int, optional*) – The chosen action of the model. Default `None`
- **response** (*float, optional*) – The response from the task after an action. Default `None`

returnTaskState()
Returns all the relevant data for this model

Returns **results**

Return type dictionary

rewardExpectation(*stimuli*)
Calculate the expected reward for each action based on the stimuli

This contains parts that are task dependent

Parameters **stimuli** (*{int | float | tuple}*) – The set of stimuli

Returns

- **expectedRewards** (*float*) – The expected reward for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

setsimID(*simID*)

Parameters **simID** (*float*) –

standardResultOutput()
Returns the relevant data expected from a model as well as the parameters for the current model

Returns **results** – A dictionary of details about the

Return type dictionary

storeStandardResults()
Updates the store of standard results found across models

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

Rewards

class model.modelTemplate.**Rewards**(***kwargs*)
Bases: object

This acts as an interface between the feedback from a task and the feedback a model can process

Name
The identifier of the function

Type string

Methods Summary

details()	
get_name()	
processFeedback(feedback, lastAction, stimuli)	Takes the feedback and turns it into a form to be processed by the model

Methods Documentation

details()

classmethod get_name()

processFeedback(*feedback*, *lastAction*, *stimuli*)
Takes the feedback and turns it into a form to be processed by the model

Parameters

- **feedback** –
- **lastAction** –
- **stimuli** –

Returns

Return type modelFeedback

Stimulus

class model.modelTemplate.**Stimulus**(***kwargs*)
Bases: object

Stimulus processor class. This acts as an interface between an observation and . Does nothing.

Name
The identifier of the function

Type string

Methods Summary

details()	
get_name()	
processStimulus(observation)	Takes the observation and turns it into a form the model can use

Methods Documentation

details()

classmethod get_name()

processStimulus(*observation*)
Takes the observation and turns it into a form the model can use

Parameters **observation** –

Returns

- **stimuliPresent** (*int or list of int*)
- **stimuliActivity** (*float or list of float*)

Class Inheritance Diagram

Author Dominic Hunt

class model.modelTemplate.**Model**(*number_actions=2*, *number_cues=1*, *number_critics=None*, *action_codes=None*, *non_action=u'None'*, *prior=None*, *stimulus_shaper=None*, *stimulus_shaper_name=None*, *stimulus_shaper_properties=None*, *reward_shaper=None*, *reward_shaper_name=None*, *reward_shaper_properties=None*, *decision_function=None*, *decision_function_name=None*, *decision_function_properties=None*, ***kwargs*)

Bases: `object`

The model class is a general template for a model. It also contains universal methods used by all models.

Name
The name of the class used when recording what has been used.

Type string

currAction
The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **prior** (array of floats in `[0,1]`, optional) – The prior probability of of the states being the correct one. Default `ones((self.number_actions, self.number_cues)) / self.number_critics)`
- **stimulus_shaper_name** (*string, optional*) – The name of the function that transforms the stimulus into a form the model can understand and a string to identify it later. `stimulus_shaper` takes priority
- **reward_shaper_name** (*string, optional*) – The name of the function that transforms the reward into a form the model can understand. `rewards_shaper` takes priority
- **decision_function_name** (*string, optional*) – The name of the function that takes the internal values of the model and turns them in to a decision. `decision function` takes priority
- **stimulus_shaper** (*Stimulus class, optional*) – The class that transforms the stimulus into a form the model can understand and a string to identify it later. Default is Stimulus
- **reward_shaper** (*Rewards class, optional*) – The class that transforms the reward into a form the model can understand. Default is Rewards
- **decision_function** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is `weightProb(range(number_actions))`

- **stimulus_shaper_properties** (*list, optional*) – The valid parameters of the function. Used to filter the unlisted keyword arguments Default is `None`
- **reward_shaper_properties** (*list, optional*) – The valid parameters of the function. Used to filter the unlisted keyword arguments Default is `None`
- **decision_function_properties** (*list, optional*) – The valid parameters of the function. Used to filter the unlisted keyword arguments Default is `None`

actStimMerge(*actStimuliParam*, *stimFilter=1*)
Takes the parameter to be merged by stimuli and filters it by the stimuli values

Parameters

- **actStimuliParam** (*list of floats*) –

  The list of values representing each action stimuli pair, where the stimuli will have their filtered values merged together.
- **stimFilter** (*array of floats or a float, optional*) – The list of active stimuli with their weightings or one weight for all. Default `1`

Returns **actionParams** – The parameter values associated with each action

Return type list of floats

action()
Returns the action of the model

Returns **action**

Return type integer or None

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns **probabilities** – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the action

Parameters **actionValues** (*1D ndArray of floats*) –

Returns **probArray** – The probabilities associated with the actionValues

Return type 1D ndArray of floats

choiceReflection()
Allows the model to update its state once an action has been chosen.

chooseAction(*probabilities*, *lastAction*, *events*, *validActions*)
Chooses the next action and returns the associated probabilities

Parameters

- **probabilities** (*list of floats*) – The probabilities associated with each combinations
- **lastAction** (*int*) – The last chosen action
- **events** (*list of floats*) – The stimuli. If probActions is True then this will be unused as the probabilities will already be
- **validActions** (*1D list or array*) – The actions permitted during this trial-step

Returns

- **newAction** (*int*) – The chosen action

- **decProbabilities** (*list of floats*) – The weights for the different actions

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

feedback(*response*)
Receives the reaction to the action and processes it

Parameters **response** (*float*) – The response from the task after an action. Returns without doing anything if the value of response is *None*.

classmethod get_name()

lastChoiceReinforcement()
Allows the model to update the reward expectation for the previous trialstep given the choice made in this trialstep

observe(*state*)
Receives the latest observation and decides what to do with it

There are five possible states: Observation Observation Action Observation Action Feedback Action Feedback Observation Feedback

Parameters **state** (*tuple of ({int | float | tuple},{tuple of int | None})*) – The stimulus from the task followed by the tuple of valid actions. Passes the values onto a processing function, self._updateObservation‘‘.

overrideActionChoice(*action*)
Provides a method for overriding the model action choice. This is used when fitting models to participant actions.

Parameters **action** (*int*) – Action chosen by external source to same situation

params()
Returns the parameters of the model

Returns **parameters**

Return type dictionary

processEvent(*action=None*, *response=None*)
Integrates the information from a stimulus, action, response set, regardless of which of the three elements are present.

Parameters

- **stimuli** (*{int | float | tuple | None}*) – The stimuli received
- **action** (*int, optional*) – The chosen action of the model. Default `None`
- **response** (*float, optional*) – The response from the task after an action. Default `None`

returnTaskState()
Returns all the relevant data for this model

Returns **results**

Return type dictionary

rewardExpectation(*stimuli*)
Calculate the expected reward for each action based on the stimuli

This contains parts that are task dependent

Parameters **stimuli** (*{int | float | tuple}*) – The set of stimuli

Returns

- **expectedRewards** (*float*) – The expected reward for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

setsimID(*simID*)

Parameters **simID** (*float*) –

standardResultOutput()
Returns the relevant data expected from a model as well as the parameters for the current model

Returns **results** – A dictionary of details about the

Return type dictionary

storeStandardResults()
Updates the store of standard results found across models

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

class model.modelTemplate.**Rewards**(***kwargs*)
Bases: object

This acts as an interface between the feedback from a task and the feedback a model can process

Name
The identifier of the function

Type string

details()

classmethod get_name()

processFeedback(*feedback*, *lastAction*, *stimuli*)
Takes the feedback and turns it into a form to be processed by the model

Parameters

- **feedback** –
- **lastAction** –
- **stimuli** –

Returns

Return type modelFeedback

class model.modelTemplate.**Stimulus**(***kwargs*)
Bases: object

Stimulus processor class. This acts as an interface between an observation and . Does nothing.

Name
The identifier of the function

Type string

details()

classmethod get_name()

processStimulus(*observation*)
Takes the observation and turns it into a form the model can use

Parameters **observation** –

Returns

- **stimuliPresent** (*int or list of int*)
- **stimuliActivity** (*float or list of float*)

6.7.2.14 model.qLearn module

Author Dominic Hunt

Reference Based on the paper Regulatory fit effects in a choice task Worthy, D. a, Maddox, W. T., & Markman, A. B. (2007). Psychonomic Bulletin & Review, 14(6), 1125–32. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/18229485

class model.qLearn.**QLearn**(*alpha=0.3*, *beta=4*, *invBeta=None*, *expect=None*, ***kwargs*)
Bases: *model.modelTemplate.Model*

The q-Learning algorithm

Name
The name of the class used when recording what has been used.

Type string

currAction
The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **alpha** (*float, optional*) – Learning rate parameter
- **beta** (*float, optional*) – Sensitivity parameter for probabilities. Also known as an exploration- exploitation parameter. Defined as β in the paper
- **invBeta** (*float, optional*) – Inverse of sensitivity parameter. Defined as $\frac{1}{\beta+1}$. Default 0.2
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues

- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`
- **expect** (*array of floats, optional*) – The initialisation of the expected reward. Default `ones((number_actions, number_cues)) * 5 / number_cues`
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns probabilities – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters actionValues (*1D ndArray of floats*) –

Returns probArray – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns results – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters observation (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.15 model.qLearn2 module

Author Dominic Hunt

Reference Modified version of that found in the paper The role of the ventromedial prefrontal cortex in abstract state-based inference during decision making in humans. Hampton, A. N., Bossaerts, P., & O'Doherty, J. P. (2006). The Journal of Neuroscience : The Official Journal of the Society for Neuroscience, 26(32), 8360–7. doi:10.1523/JNEUROSCI.1010-06.2006

Notes In the original paper this model used the Luce choice algorithm, rather than the logistic algorithm used here. This generalisation has meant that the variable nu is no longer possible to use.

class model.qLearn2.**QLearn2**(*alpha=0.3*, *beta=4*, *alphaPos=None*, *alphaNeg=None*, *invBeta=None*, *expect=None*, ***kwargs*)
Bases: *model.modelTemplate.Model*

The q-Learning algorithm modified to have different positive and negative reward prediction errors

Name
The name of the class used when recording what has been used.

Type string

currAction
The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **alpha** (*float, optional*) – Learning rate parameter. For this model only used when setting alphaPos and alphaNeg to the same value. Default 0.3
- **alphaPos** (*float, optional*) – The positive learning rate parameter. Used when RPE is positive. Default is alpha
- **alphaNeg** (*float, optional*) – The negative learning rate parameter. Used when RPE is negative. Default is alpha
- **beta** (*float, optional*) – Sensitivity parameter for probabilities
- **invBeta** (*float, optional*) – Inverse of sensitivity parameter. Defined as $\frac{1}{\beta+1}$. Default 0.2

- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`
- **expect** (*array of floats, optional*) – The initialisation of the the expected reward. Default `ones((number_actions, number_cues)) * 5 / number_cues`
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

See also:

model.QLearn This model is heavily based on that one

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns probabilities – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters actionValues (*1D ndArray of floats*) –

Returns probArray – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.16 model.qLearn2E module

Author Dominic Hunt

Reference Modified version of that found in the paper The role of the ventromedial prefrontal cortex in abstract state-based inference during decision making in humans. Hampton, A. N., Bossaerts, P., & O'Doherty, J. P. (2006). The Journal of Neuroscience : The Official Journal of the Society for Neuroscience, 26(32), 8360–7. doi:10.1523/JNEUROSCI.1010-06.2006

Notes In the original paper this model used the Luce choice algorithm, rather than the logistic algorithm used here. This generalisation has meant that the variable nu is no longer possible to use.

class model.qLearn2E.**QLearn2E**(*alpha=0.3*, *epsilon=0.1*, *alphaPos=None*, *alphaNeg=None*, *expect=None*, ***kwargs*)
Bases: *model.modelTemplate.Model*

The q-Learning algorithm modified to have different positive and negative reward prediction errors and use the Epsylon greedy method for claculating probabilities

Name
The name of the class used when recording what has been used.

Type string

currAction
The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **alpha** (*float, optional*) – Learning rate parameter. For this model only used when setting alphaPos and alphaNeg to the same value. Default 0.3
- **alphaPos** (*float, optional*) – The positive learning rate parameter. Used when RPE is positive. Default is alpha
- **alphaNeg** (*float, optional*) – The negative learning rate parameter. Used when RPE is negative. Default is alpha
- **epsilon** (*float, optional*) – Noise parameter. The larger it is the less likely the model is to choose the highest expected reward
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`
- **expect** (*array of floats, optional*) – The initialisation of the the expected reward. Default `ones((number_actions, number_cues)) * 5 / number_cues`
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

See also:

model.QLearn This model is heavily based on that one

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns probabilities – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters actionValues (*1D ndArray of floats*) –

Returns probArray – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.17 model.qLearnCorr module

Author Dominic Hunt

Reference Based on the QLearn model and the choice autocorrelation equation in the paper Trial-by-trial data analysis using computational models. Daw, N. D. (2011). Decision Making, Affect, and Learning: Attention and Performance XXIII (pp. 3–38). http://doi.org/10.1093/acprof:oso/9780199600434.003.0001

class model.qLearnCorr.**QLearnCorr**(*alpha=0.3*, *beta=4*, *kappa=0.1*, *invBeta=None*, *expect=None*, ***kwargs*)

Bases: *model.modelTemplate.Model*

The q-Learning algorithm

Name
The name of the class used when recording what has been used.

Type string

currAction
The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **alpha** (*float, optional*) – Learning rate parameter
- **beta** (*float, optional*) – Sensitivity parameter for probabilities
- **kappa** (*float, optional*) – The autocorelation parameter for which positive values promote sticking and negative values promote alternation
- **invBeta** (*float, optional*) – Inverse of sensitivity parameter. Defined as $\frac{1}{\beta+1}$. Default `0.2`
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`
- **expect** (*array of floats, optional*) – The initialisation of the the expected reward. Default `ones((number_actions, number_cues)) * 5 / number_cues`
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

See also:

model.QLearn This model is heavily based on that one

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns **probabilities** – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters **actionValues** (*1D ndArray of floats*) –

Returns **probArray** – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.18 model.qLearnE module

Author Dominic Hunt

Reference Based on the Epsilon-greedy method along with a past choice autocorrelation inspired by `QLearnCorr`

class `model.qLearnE.`**QLearnE**(*alpha=0.3*, *epsilon=0.1*, *expect=None*, ***kwargs*)
Bases: *model.modelTemplate.Model*

The q-Learning algorithm

Name
The name of the class used when recording what has been used.

Type string

currAction
The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **alpha** (*float, optional*) – Learning rate parameter
- **epsilon** (*float, optional*) – Noise parameter. The larger it is the less likely the model is to choose the highest expected reward
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`
- **expect** (*array of floats, optional*) – The initialisation of the expected reward. Default `ones((number_actions, number_cues)) * 5 / number_cues`
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

See also:

model.QLearn This model is heavily based on that one

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns **probabilities** – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters **actionValues** (*1D ndArray of floats*) –

Returns **probArray** – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.19 model.qLearnECorr module

Author Dominic Hunt

Reference Based on the Epsilon-greedy method along with a past choice autocorrelation inspired by `QLearnCorr`

class model.qLearnECorr.**QLearnECorr**(*alpha=0.3*, *epsilon=0.1*, *kappa=0.1*, *expect=None*, ***kwargs*)

Bases: *model.modelTemplate.Model*

The q-Learning algorithm

Name
The name of the class used when recording what has been used.

Type string

currAction
The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **alpha** (*float, optional*) – Learning rate parameter
- **kappa** (*float, optional*) – The autocorrelation parameter for which positive values promote sticking and negative values promote alternation
- **epsilon** (*float, optional*) – Noise parameter. The larger it is the less likely the model is to choose the highest expected reward
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`
- **expect** (*array of floats, optional*) – The initialisation of the the expected reward. Default `ones((number_actions, number_cues)) * 5 / number_cues`
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

See also:

model.QLearnCorr This model is heavily based on that one

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns **probabilities** – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters **actionValues** (*1D ndArray of floats*) –

Returns **probArray** – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep

- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.20 model.qLearnF module

Author Dominic Hunt

Reference Based on the paper Regulatory fit effects in a choice task Worthy, D. a, Maddox, W. T., & Markman, A. B. (2007). Psychonomic Bulletin & Review, 14(6), 1125–32. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/18229485

class model.qLearnF.**QLearnF**(*alpha=0.3*, *beta=4*, *gamma=0.3*, *invBeta=None*, *expect=None*, ***kwargs*)

Bases: *model.modelTemplate.Model*

The q-Learning algorithm

Name

The name of the class used when recording what has been used.

Type string

currAction

The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **alpha** (*float, optional*) – Learning rate parameter
- **beta** (*float, optional*) – Sensitivity parameter for probabilities
- **gamma** (*float, optional*) – future expectation discounting
- **invBeta** (*float, optional*) – Inverse of sensitivity parameter. Defined as $\frac{1}{\beta+1}$. Default `0.2`
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`
- **expect** (*array of floats, optional*) – The initialisation of the expected reward. Default `ones((number_actions, number_cues)) * 5 / number_cues`
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew

- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

See also:

model.QLearn This model is heavily based on that one

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns probabilities – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters actionValues (*1D ndArray of floats*) –

Returns probArray – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

lastChoiceReinforcement()
Allows the model to update its expectations once the action has been chosen.

returnTaskState()
Returns all the relevant data for this model

Returns results – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters observation (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.21 model.qLearnK module

Author Dominic Hunt

Reference Based on the paper Cortical substrates for exploratory decisions in humans. Daw, N. D., O'Doherty, J. P., Dayan, P., Dolan, R. J., & Seymour, B. (2006). Nature, 441(7095), 876–9. https://doi.org/10.1038/nature04766

class model.qLearnK.**QLearnK**(*beta=4*, *sigma=1*, *sigmaG=1*, *drift=1*, *sigmaA=None*, *alphaA=None*, *invBeta=None*, *expect=None*, ***kwargs*)

Bases: *model.modelTemplate.Model*

The q-Learning Kalman algorithm

Name
The name of the class used when recording what has been used.

Type string

currAction
The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **sigma** (*float, optional*) – Uncertainty scale measure
- **sigmaG** (*float, optional*) – Uncertainty measure growth
- **drift** (*float, optional*) – The drift rate
- **beta** (*float, optional*) – Sensitivity parameter for probabilities. Also known as an exploration- exploitation parameter. Defined as β in the paper
- **invBeta** (*float, optional*) – Inverse of sensitivity parameter. Defined as $\frac{1}{\beta+1}$. Default `0.2`
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`

- **expect** (*array of floats, optional*) – The initialisation of the expected reward. Default `ones((number_actions, number_cues)) * 5 / number_cues`
- **sigmaA** (*array of floats, optional*) – The initialisation of the uncertainty measure
- **alphaA** (*array of floats, optional*) – The initialisation of the learning rates
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns **probabilities** – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters **actionValues** (*1D ndArray of floats*) –

Returns **probArray** – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action

- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.22 model.qLearnMeta module

Author Dominic Hunt

Reference Based on the model QLearn as well as the paper: Meta-learning in Reinforcement Learning

class model.qLearnMeta.**QLearnMeta**(*alpha=0.3*, *tau=0.2*, *rewardD=None*, *rewardDD=None*, *expect=None*, ***kwargs*)

Bases: *model.modelTemplate.Model*

The q-Learning algorithm with a second-order adaptive beta

Name
The name of the class used when recording what has been used.

Type string

currAction
The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **alpha** (*float, optional*) – Learning rate parameter
- **tau** (*float, optional*) – Beta rate Sensitivity parameter for probabilities
- **invBeta** (*float, optional*) – Inverse of sensitivity parameter. Defined as $\frac{1}{\beta+1}$. Default `0.2`
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.

- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`
- **expect** (*array of floats, optional*) – The initialisation of the the expected reward. Default `ones((number_actions, number_cues)) * 5 / number_cues`
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.binary.eta

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns **probabilities** – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters **actionValues** (*1D ndArray of floats*) –

Returns **probArray** – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations

- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateBeta(*reward*, *action*)

Parameters **reward** (*float*) – The reward value

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.23 model.randomBias module

Author Dominic Hunt

class model.randomBias.**RandomBias**(*expect=None*, ***kwargs*)
Bases: *model.modelTemplate.Model*

A model replicating a participant who chooses randomly, but with a bias towards certain actions

Name
The name of the class used when recording what has been used.

Type string

currAction
The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns **probabilities** – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities()
Calculate the probabilities associated with the actions

Parameters **actionValues** (*1D ndArray of floats*) –

Returns **probArray** – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.24 model.td0 module

Author Dominic Hunt

Reference Based on the description on p134-135 of Reinforcement Learning, Sutton & Barto 1998

class model.td0.**TD0**(*alpha=0.3*, *beta=4*, *gamma=0.3*, *invBeta=None*, *expect=None*, ***kwargs*)
Bases: *model.modelTemplate.Model*

The td-Learning algorithm

Name
The name of the class used when recording what has been used.

Type string

currAction
The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **alpha** (*float, optional*) – Learning rate parameter
- **beta** (*float, optional*) – Sensitivity parameter for probabilities
- **gamma** (*float, optional*) – future expectation discounting
- **invBeta** (*float, optional*) – Inverse of sensitivity parameter. Defined as $\frac{1}{\beta+1}$. Default `0.2`
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`
- **expect** (*array of floats, optional*) – The initialisation of the expected reward. Default `ones((number_actions, number_cues)) * 5 / number_cues`
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns **probabilities** – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters **actionValues** (*1D ndArray of floats*) –

Returns **probArray** – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

lastChoiceReinforcement()
Allows the model to update its expectations once the action has been chosen.

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep
- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.25 model.tdE module

Author Dominic Hunt

Reference Based on the description on p134-135 of Reinforcement Learning, Sutton & Barto 1998

class model.tdE.**TDE**(*alpha=0.3*, *epsilon=0.1*, *gamma=0.3*, *expect=None*, ***kwargs*)
Bases: *model.modelTemplate.Model*

The td-Learning algorithm

Name
The name of the class used when recording what has been used.

Type string

currAction
The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **alpha** (*float, optional*) – Learning rate parameter
- **epsilon** (*float, optional*) – Sensitivity parameter for probabilities
- **gamma** (*float, optional*) – future expectation discounting
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`
- **expect** (*array of floats, optional*) – The initialisation of the expected reward. Default `ones((number_actions, number_cues)) * 5 / number_cues`
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

See also:

model.TD0 This model is heavily based on that one

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns **probabilities** – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters **actionValues** (*1D ndArray of floats*) –

Returns **probArray** – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

lastChoiceReinforcement()
Allows the model to update its expectations once the action has been chosen.

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep

- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.7.2.26 model.tdr module

Author Dominic Hunt

class model.tdr.**TDR**(*alpha=0.3*, *beta=4*, *tau=0.3*, *invBeta=None*, *expect=None*, *avReward=None*, ***kwargs*)

Bases: *model.modelTemplate.Model*

The td-Learning algorithm

Name

The name of the class used when recording what has been used.

Type string

currAction

The current action chosen by the model. Used to pass participant action to model when fitting

Type int

Parameters

- **alpha** (*float, optional*) – Learning rate parameter
- **beta** (*float, optional*) – Sensitivity parameter for probabilities
- **invBeta** (*float, optional*) – Inverse of sensitivity parameter. Defined as $\frac{1}{\beta+1}$. Default `0.2`
- **tau** (*float, optional*) – Learning rate for average reward
- **number_actions** (*integer, optional*) – The maximum number of valid actions the model can expect to receive. Default 2.
- **number_cues** (*integer, optional*) –

  The initial maximum number of stimuli the model can expect to receive. Default 1.
- **number_critics** (*integer, optional*) – The number of different reaction learning sets. Default number_actions*number_cues
- **action_codes** (*dict with string or int as keys and int values, optional*) – A dictionary used to convert between the action references used by the task or dataset and references used in the models to describe the order in which the action information is stored.
- **prior** (array of floats in `[0, 1]`, optional) – The prior probability of of the states being the correct one. Default `ones((number_actions, number_cues)) / number_critics)`
- **expect** (*array of floats, optional*) – The initialisation of the expected reward. Default `ones((number_actions, number_cues)) * 5 / number_cues`
- **stimFunc** (*function, optional*) – The function that transforms the stimulus into a form the model can understand and a string to identify it later. Default is blankStim
- **rewFunc** (*function, optional*) – The function that transforms the reward into a form the model can understand. Default is blankRew
- **decFunc** (*function, optional*) – The function that takes the internal values of the model and turns them in to a decision. Default is model.decision.discrete.weightProb

actorStimulusProbs()
Calculates in the model-appropriate way the probability of each action.

Returns **probabilities** – The probabilities associated with the action choices

Return type 1D ndArray of floats

calcProbabilities(*actionValues*)
Calculate the probabilities associated with the actions

Parameters **actionValues** (*1D ndArray of floats*) –

Returns **probArray** – The probabilities associated with the actionValues

Return type 1D ndArray of floats

delta(*reward*, *expectation*, *action*, *stimuli*)
Calculates the comparison between the reward and the expectation

Parameters

- **reward** (*float*) – The reward value
- **expectation** (*float*) – The expected reward value
- **action** (*int*) – The chosen action
- **stimuli** (*{int | float | tuple | None}*) – The stimuli received

Returns

Return type delta

lastChoiceReinforcement()
Allows the model to update its expectations once the action has been chosen.

returnTaskState()
Returns all the relevant data for this model

Returns **results** – The dictionary contains a series of keys including Name, Probabilities, Actions and Events.

Return type dict

rewardExpectation(*observation*)
Calculate the estimated reward based on the action and stimuli

This contains parts that are task dependent

Parameters **observation** (*{int | float | tuple}*) – The set of stimuli

Returns

- **actionExpectations** (*array of floats*) – The expected rewards for each action
- **stimuli** (*list of floats*) – The processed observations
- **activeStimuli** (*list of [0, 1] mapping to [False, True]*) – A list of the stimuli that were or were not present

storeState()
Stores the state of all the important variables so that they can be accessed later

updateModel(*delta*, *action*, *stimuli*, *stimuliFilter*)

Parameters

- **delta** (*float*) – The difference between the reward and the expected reward
- **action** (*int*) – The action chosen by the model in this trialstep
- **stimuli** (*list of float*) – The weights of the different stimuli in this trialstep

- **stimuliFilter** (*list of bool*) – A list describing if a stimulus cue is present in this trialstep

6.8 fitAlgs package

6.8.1 fitAlgs Package

6.8.2 Submodules

6.8.2.1 fitAlgs.basinhopping module

Author Dominic Hunt

class fitAlgs.basinhopping.**Basinhopping**(*method=None*, *number_start_points=4*, *allow_boundary_fits=True*, *boundary_fit_sensitivity=5*, ***kwargs*)

Bases: *fitAlgs.fitAlg.FitAlg*

The class for fitting data using scipy.optimise.basinhopping

Parameters

- **fit_sim** (*fitAlgs.fitSims.FitSim instance, optional*) – An instance of one of the fitting simulation methods. Default `fitAlgs.fitSims.FitSim`
- **fit_measure** (*string, optional*) – The name of the function used to calculate the quality of the fit. The value it returns provides the fitter with its fitting guide. Default `-loge`
- **fit_measure_args** (*dict, optional*) – The parameters used to initialise fit_measure and extra_fit_measures. Default `None`
- **extra_fit_measures** (*list of strings, optional*) – List of fit measures not used to fit the model, but to provide more information. Any arguments needed for these measures should be placed in fit_measure_args. Default `None`
- **bounds** (*dictionary of tuples of length two with floats, optional*) – The boundaries for methods that use bounds. If unbounded methods are specified then the bounds will be ignored. Default is `None`, which translates to boundaries of (0, np.inf) for each parameter.
- **boundary_excess_cost** (*basestring or callable returning a function, optional*) – The function is used to calculate the penalty for exceeding the boundaries. Default is `boundFunc.scalarBound()`
- **boundary_excess_cost_properties** (*dict, optional*) – The parameters for the boundary_excess_cost function. Default {}
- **method** (*string or list of strings, optional*) – The name of the fitting method or list of names of fitting methods or name of list of fitting methods. Valid names found in the notes. Default `unconstrained`
- **number_start_points** (*int, optional*) – The number of starting points generated for each parameter. Default 4
- **allow_boundary_fits** (*bool, optional*) – Defines if fits that reach a boundary should be considered the same way as those that do not. Default is True
- **boundSensitivity** (*int, optional*) – Defines the smallest number of decimal places difference (so the minimal difference) between a fit value and its related boundaries before a fit value is considered different from a boundary. The default is *5*. This is only valid if `allow_boundary_fits` is `False`

Name
The name of the fitting method

Type string

unconstrained
The list of valid unconstrained fitting methods

Type list

constrained
The list of valid constrained fitting methods

Type list

Notes

unconstrained = ['Nelder-Mead','Powell','CG','BFGS'] constrained = ['L-BFGS-B','TNC','SLSQP'] Custom fitting algorithms are also allowed in theory, but it has yet to be implemented.

For each fitting function a set of different starting parameters will be tried. These are the combinations of all the values of the different parameters. For each starting parameter provided a set of number_start_points starting points will be chosen, surrounding the starting point provided. If the starting point provided is less than one it will be assumed that the values cannot exceed 1, otherwise, unless otherwise told, it will be assumed that they can take any value and will be chosen to be eavenly spaced around the provided value.

See also:

fitAlgs.fitAlg.fitAlg The general fitting method class, from which this one inherits

filtAlgs.fitSims.fitSim The general fitting class

scipy.optimise.basinhopping The fitting class this wraps around

callback(*x*, *f*, *accept*)
Used for storing the state after each stage of fitter

Parameters

- **x** (*coordinates of the trial minimum*) –
- **f** (*function value of the trial minimum*) –
- **accept** (*whether or not that minimum was accepted*) –

constrained = [u'L-BFGS-B', u'TNC', u'SLSQP']

fit(*simulator*, *model_parameter_names*, *model_initial_parameters*)
Runs the model through the fitting algorithms and starting parameters and returns the best one.

Parameters

- **simulator** (*function*) – The function used by a fitting algorithm to generate a fit for given model parameters. One example is fitAlgs.fitAlg.fitness
- **model_parameter_names** (*list of strings*) – The list of initial parameter names
- **model_initial_parameters** (*list of floats*) – The list of the intial parameters

Returns

- **best_fit_parameters** (*list of floats*) – The best fitting parameters
- **fit_quality** (*float*) – The quality of the fit as defined by the quality function chosen.

- **testedParams** (*tuple of two lists and a dictionary*) – The two lists are a list containing the parameter values tested, in the order they were tested, and the fit qualities of these parameters. The dictionary contains the coordinates of the trial minimum, the function value of the trial minimum and whether or not that minimum was accepted. Each is stored in a list.

See also:

`fitAlgs.fitAlg.fitness()`

`unconstrained = [u'Nelder-Mead', u'Powell', u'CG', u'BFGS']`

6.8.2.2 fitAlgs.boundFunc module

fitAlgs.boundFunc Module

Author Dominic Hunt

Functions

infBound([base])	Boundary excess of `inf` when over bounds
scalarBound([base])	Boundary excess calculated as a scalar increase based on difference with bounds

infBound

`fitAlgs.boundFunc.`**`infBound`**(*base=0*)

Boundary excess of `inf` when over bounds

Parameters **`base`** (*`float, optional`*) – The cost at the boundary. Default 0

Returns **cost** – Calculates the cost of exceeding the bounday using the parameters and the boundaries, and returns the cost.

Return type function

Examples

```
>>> cst = infBound(base = 160)
>>> cst([0.5, 2], [(0, 1), (0, 5)])
160
>>> cst([0.5, 7], [(0, 1), (0, 5)])
inf
```

scalarBound

`fitAlgs.boundFunc.`**`scalarBound`**(*base=0*)

Boundary excess calculated as a scalar increase based on difference with bounds

Parameters **`base`** (*`float, optional`*) – The cost at the boundary. Default 0

Returns **cost** – Calculates the cost of exceeding the boundary using the parameters and the boundaries, and returns the cost.

Return type function

Examples

```
>>> cst = scalarBound(base=160)
>>> cst([0.5, 2], [(0, 1), (0, 5)])
160.0
>>> cst([0.5, 7], [(0, 1), (0, 5)])
162.0
```

Author Dominic Hunt

fitAlgs.boundFunc.**infBound**(*base=0*)

Boundary excess of `inf` when over bounds

Parameters **base** (*float, optional*) – The cost at the boundary. Default 0

Returns **cost** – Calculates the cost of exceeding the bounday using the parameters and the boundaries, and returns the cost.

Return type function

Examples

```
>>> cst = infBound(base = 160)
>>> cst([0.5, 2], [(0, 1), (0, 5)])
160
>>> cst([0.5, 7], [(0, 1), (0, 5)])
inf
```

fitAlgs.boundFunc.**scalarBound**(*base=0*)

Boundary excess calculated as a scalar increase based on difference with bounds

Parameters **base** (*float, optional*) – The cost at the boundary. Default 0

Returns **cost** – Calculates the cost of exceeding the boundary using the parameters and the boundaries, and returns the cost.

Return type function

Examples

```
>>> cst = scalarBound(base=160)
>>> cst([0.5, 2], [(0, 1), (0, 5)])
160.0
>>> cst([0.5, 7], [(0, 1), (0, 5)])
162.0
```

6.8.2.3 fitAlgs.evolutionary module

Author Dominic Hunt

class fitAlgs.evolutionary.**Evolutionary**(*strategy=None, polish=False, population_size=20, tolerance=0.01, **kwargs*)

Bases: *fitAlgs.fitAlg.FitAlg*

The class for fitting data using scipy.optimise.differential_evolution

Parameters

- **fit_sim** (*fitAlgs.fitSims.FitSim instance, optional*) – An instance of one of the fitting simulation methods. Default `fitAlgs.fitSims.FitSim`

- **fit_measure** (*string, optional*) – The name of the function used to calculate the quality of the fit. The value it returns provides the fitter with its fitting guide. Default `-loge`
- **fit_measure_args** (*dict, optional*) – The parameters used to initialise fit_measure and extra_fit_measures. Default `None`
- **extra_fit_measures** (*list of strings, optional*) – List of fit measures not used to fit the model, but to provide more information. Any arguments needed for these measures should be placed in fit_measure_args. Default `None`
- **bounds** (*dictionary of tuples of length two with floats, optional*) – The boundaries for methods that use bounds. If unbounded methods are specified then the bounds will be ignored. Default is `None`, which translates to boundaries of (0, np.inf) for each parameter.
- **boundary_excess_cost** (*basestring or callable returning a function, optional*) – The function is used to calculate the penalty for exceeding the boundaries. Default is `boundFunc.scalarBound()`
- **boundary_excess_cost_properties** (*dict, optional*) – The parameters for the boundary_excess_cost function. Default {}
- **strategy** (*string or list of strings, optional*) – The name of the fitting strategy or list of names of fitting strategies or name of a list of fitting strategies. Valid names found in the notes. Default `best1bin`
- **polish** (*bool, optional*) – If True (default), then scipy.optimize.minimize with the `L-BFGS-B` method is used to polish the best population member at the end, which can improve the minimization slightly. Default `False`
- **population_size** (*int, optional*) – A multiplier for setting the total population size. The population has popsize * len(x) individuals. Default 20
- **tolerance** (*float, optional*) – When the mean of the population energies, multiplied by tol, divided by the standard deviation of the population energies is greater than 1 the solving process terminates: convergence = mean(pop) * tol / stdev(pop) > 1 Default 0.01

Name
The name of the fitting strategies

Type string

strategySet
The list of valid fitting strategies. Currently these are: 'best1bin', 'best1exp', 'rand1exp', 'randtobest1exp', 'best2exp', 'rand2exp', 'randtobest1bin', 'best2bin', 'rand2bin', 'rand1bin' For all strategies, use 'all'

Type list

See also:

fitAlgs.fitAlg.FitAlg The general fitting strategy class, from which this one inherits

fitAlgs.fitSims.FitSim The general class for seeing how a parameter combination perform

scipy.optimise.differential_evolution The fitting method this wraps around

callback (*xk*, *convergence*)
Used for storing the state after each stage of fitting

Parameters

- **xk** (*coordinates of best fit*) –
- **convergence** (*the proportion of the points from the iteration that have converged*) –

fit (*simulator*, *model_parameter_names*, *model_initial_parameters*)
Runs the model through the fitting algorithms and starting parameters and returns the best one.

Parameters

- **simulator** (*function*) – The function used by a fitting algorithm to generate a fit for given model parameters. One example is fitAlgs.fitSim.fitness
- **model_parameter_names** (*list of strings*) – The list of initial parameter names
- **model_initial_parameters** (*list of floats*) – The list of the initial parameters

Returns

- **best_fit_parameters** (*list of floats*) – The best fitting parameters
- **fit_quality** (*float*) – The quality of the fit as defined by the quality function chosen.
- **testedParams** (*tuple of two lists and a dictionary*) – The two lists are a list containing the parameter values tested, in the order they were tested, and the fit qualities of these parameters. The dictionary contains the parameters and convergence values from each iteration, stored in two lists.

See also:

`fitAlgs.fitAlg.fitness()`

validStrategySet = [u'best1bin', u'best1exp', u'rand1exp', u'randtobest1exp', u'best

6.8.2.4 fitAlgs.fitAlg module

Author Dominic Hunt

class fitAlgs.fitAlg.**FitAlg**(*fit_sim=None*, *fit_measure=u'-loge'*, *fit_measure_args=None*, *extra_fit_measures=None*, *bounds=None*, *boundary_excess_cost=None*, *boundary_excess_cost_properties=None*, *bound_ratio=1e-06*, *calculate_covariance=False*, ***kwargs*)
Bases: object

The abstract class for fitting data

Parameters

- **fit_sim** (*fitAlgs.fitSims.FitSim instance, optional*) – An instance of one of the fitting simulation methods. Default `fitAlgs.fitSims.FitSim`
- **fit_measure** (*string, optional*) – The name of the function used to calculate the quality of the fit. The value it returns provides the fitter with its fitting guide. Default `-loge`
- **fit_measure_args** (*dict, optional*) – The parameters used to initialise fit_measure and extra_fit_measures. Default `None`
- **extra_fit_measures** (*list of strings, optional*) – List of fit measures not used to fit the model, but to provide more information. Any arguments needed for these measures should be placed in fit_measure_args. Default `None`
- **bounds** (*dictionary of tuples of length two with floats, optional*) – The boundaries for methods that use bounds. If unbounded methods are specified then the bounds will be ignored. Default is `None`, which translates to boundaries of (0, np.inf) for each parameter.

- **boundary_excess_cost** (*basestring or callable returning a function, optional*) – The function is used to calculate the penalty for exceeding the boundaries. Default is `boundFunc.scalarBound()`
- **boundary_excess_cost_properties** (*dict, optional*) – The parameters for the boundary_excess_cost function. Default {}
- **calculate_covariance** (*bool, optional*) – Is the covariance calculated. Default `False`

Name
The name of the fitting method

Type string

See also:

fitAlgs.fitSims.fitSim The general fitting class

covariance(*model_parameter_names*, *paramvals*, *fitinfo*)
The covariance at a point

Parameters

- **paramvals** (*array or list*) – The parameters at which the
- **fitinfo** (*dict*) – The

Returns **covariance** – The covariance at the point paramvals

Return type float

extra_measures(**model_parameter_values*)

Parameters ***model_parameter_values** (*array of floats*) – The parameters proposed by the fitting algorithm

Returns **fit_quality** – The fit quality value calculated using the fit quality functions described in extraMeasures

Return type dict of float

find_name()
Returns the name of the class

fit(*simulator*, *model_parameter_names*, *model_initial_parameters*)
Runs the model through the fitting algorithms and starting parameters and returns the best one. This is the abstract version that always returns `(0,0)`

Parameters

- **simulator** (*function*) – The function used by a fitting algorithm to generate a fit for given model parameters. One example is `fitAlgs.fitAlg.fitness`
- **model_parameter_names** (*list of strings*) – The list of initial parameter names
- **model_initial_parameters** (*list of floats*) – The list of the initial parameters

Returns

- **best_fit_parameters** (*list of floats*) – The best fitting parameters
- **fit_quality** (*float*) – The quality of the fit as defined by the quality function chosen.
- **tested_parameters** (*tuple of two lists*) – The two lists are a list containing the parameter values tested, in the order they were tested, and the fit qualities of these parameters.

See also:

`fitAlgs.fitAlg.fitness()`

fitness (**params*)
Generates a fit quality value used by the fitting function. This is the function passed to the fitting function.

Parameters ***params** (*array of floats*) – The parameters proposed by the fitting algorithm

Returns **fit_quality** – The fit quality value calculated using the fitQualFunc function

Return type float

See also:

fitAlgs.qualityFunc() the module of fitQualFunc functions

fitAlg.invalidParams() Checks if the parameters are valid and if not returns `inf`

fitAlgs.fitSims.fitSim.fitness() Runs the model simulation and returns the values used to calculate the fit quality

info()
The information relating to the fitting method used

Includes information on the fitting algorithm used

Returns **info** – The fitSims info and the fitAlgs.fitAlg info

Return type (dict,dict)

See also:

`fitAlg.fitSims.fitSim.info()`

invalid_parameters (**params*)
Identifies if the parameters passed are within the bounds provided

If they are not returns `inf`

Parameters **params** (*list of floats*) – Parameters to be passed to the sim

Returns **validity** – If the parameters are valid or not

Return type Bool

Notes

No note

Examples

```
>>> a = FitAlg(bounds={1:(0,5), 2:(0,2), 3:(-1,1)})
>>> a.set_bounds([3, 1])
>>> a.invalid_parameters(0, 0)
False
>>> a.invalid_parameters(2, 0)
True
>>> a.invalid_parameters(0, -1)
True
>>> a.invalid_parameters(6, 6)
True
```

participant (*model*, *model_parameters*, *model_properties*, *participant_data*)
Fit participant data to a model for a given task

Parameters

- **model** (*model.modelTemplate.Model inherited class*) – The model you wish to try and fit values to
- **model_parameters** (*dict*) – The model initial parameters
- **model_properties** (*dict*) – The model static properties
- **participant_data** (*dict*) – The participant data

Returns

- **model** (*model.modelTemplate.Model inherited class instance*) – The model with the best fit parameters
- **fit_quality** (*float*) – Specifies the fit quality for this participant to the model
- **fitting_data** (*tuple of OrderedDict and list*) – They are an ordered dictionary containing the parameter values tested, in the order they were tested, and the fit qualities of these parameters.

set_bounds (*model_parameter_names*)
Checks if the bounds have changed

Parameters **model_parameter_names** (*list of strings*) – An ordered list of the names of the parameters to be fitted

Examples

```
>>> a = FitAlg(bounds={1: (0, 5), 2: (0, 2), 3: (-1, 1)})
>>> a.boundaries
{1: (0, 5), 2: (0, 2), 3: (-1, 1)}
>>> a.set_bounds([])
>>> a.boundaries
{1: (0, 5), 2: (0, 2), 3: (-1, 1)}
>>> a.boundary_names
[]
>>> a.set_bounds([3,1])
>>> a.boundary_values
[(-1, 1), (0, 5)]
>>> a.set_bounds([2,1])
>>> a.boundary_values
[(0, 2), (0, 5)]
```

classmethod startParams (*initial_parameters*, *bounds=None*, *number_starting_points=3*)
Defines a list of different starting parameters to run the minimization over

Parameters

- **initial_parameters** (*list of floats*) – The initial starting values proposed
- **bounds** (*list of tuples of length two with floats, optional*) – The boundaries for methods that use bounds. If unbounded methods are specified then the bounds will be ignored. Default is `None`, which translates to boundaries of (0,float('Inf')) for each parameter.
- **number_starting_points** (*int*) – The number of starting parameter values to be calculated around each initial point

Returns **startParamSet** – The generated starting parameter combinations

Return type list of list of floats

See also:

FitAlg.start_parameter_values() Used in this function

Examples

```
>>> FitAlg.startParams([0.5,0.5], number_starting_points=2)
array([[0.33333333, 0.33333333],
       [0.66666667, 0.33333333],
       [0.33333333, 0.66666667],
       [0.66666667, 0.66666667]])
```

static start_parameter_values(*initial*, *boundary_min=-inf*, *boundary_max=inf*, *number_starting_points=3*)

Provides a set of starting points

Parameters

- **initial** (*float*) – The initial starting value proposed
- **boundary_min** (*float, optional*) – The minimum value of the parameter. Default is `float('-Inf')`
- **boundary_max** (*float, optional*) – The maximum value of the parameter. Default is `float('Inf')`
- **number_starting_points** (*int*) – The number of starting parameter values to be calculated around the inital point

Returns **startParams** – The generated starting parameters

Return type list of floats

Notes

For each starting parameter provided a set of numStartPoints starting points will be chosen, surrounding the starting point provided. If the starting point provided is less than one but greater than zero it will be assumed that the values cannot leave those bounds, otherwise, unless otherwise told, it will be assumed that they can take any positive value and will be chosen to be eavenly spaced around the provided value.

Examples

```
>>> FitAlg.start_parameter_values(0.5)
array([0.25, 0.5 , 0.75])
>>> FitAlg.start_parameter_values(5)
array([2.5, 5. , 7.5])
>>> FitAlg.start_parameter_values(-5)
array([2.5, 5. , 7.5])
>>> FitAlg.start_parameter_values(5, boundary_min = 0, boundary_max = 7)
array([4., 5., 6.])
>>> FitAlg.start_parameter_values(5, boundary_min = -3, boundary_max = 30)
array([1., 5., 9.])
>>> FitAlg.start_parameter_values(5, boundary_min = 0, boundary_max = 30)
array([2.5, 5. , 7.5])
>>> FitAlg.start_parameter_values(5, boundary_min = 3, boundary_max = 30,␣
↪number_starting_points = 7)
array([3.5, 4. , 4.5, 5. , 5.5, 6. , 6.5])
```

fitAlgs.fitAlg.**covariance**(*jac*)

Calculates the covariance based on the estimated jacobian

Inspired by how this is calculated in scipy.optimise.curve_fit, as found at https://github.com/scipy/scipy/blob/2526df72e5d4ca8bad6e2f4b3cbdfbc33e805865/scipy/optimize/minpack.py#L739

6.8.2.5 fitAlgs.fitSims module

fitAlgs.fitSims Module

Author Dominic Hunt

Classes

ActionError	
FitSim([participant_choice_property, ...])	A class for fitting data by passing the participant data through the model.
FitSubsetError	
StimuliError	

ActionError

exception fitAlgs.fitSims.**ActionError**

FitSim

class fitAlgs.fitSims.**FitSim**(*participant_choice_property=u'Actions'*, *participant_reward_property=u'Rewards'*, *model_fitting_variable=u'ActionProb'*, *task_stimuli_property=None*, *fit_subset=None*, *action_options_property=None*, *float_error_response_value=1e-100*)

Bases: object

A class for fitting data by passing the participant data through the model.

This has been setup for fitting action-response models

Parameters

- **participant_choice_property** (*string, optional*) – The participant data key of their action choices. Default 'Actions'
- **participant_reward_property** (*string, optional*) – The participant data key of the participant reward data. Default 'Rewards'
- **model_fitting_variable** (*string, optional*) – The key to be compared in the model data. Default 'ActionProb'
- **task_stimuli_property** (*list of strings or None, optional*) – The keys containing the stimuli seen by the participant before taking a decision on an action. Default None
- **action_options_property** (*string or None or list of ints, optional*) – The name of the key in partData where the list of valid actions can be found. If None then the action list is considered to stay constant. If a list then the list will be taken as the list of actions that can be taken at each instance. Default None

- **float_error_response_value** (*float, optional*) – If a floating point error occurs when running a fit the fitter function will return a value for each element of fpRespVal. Default is `1/1e100`
- **fit_subset** (`float('Nan')`, `None`, `"rewarded"`, `"unrewarded"`, `"all"` or list of int, optional) – Describes which, if any, subset of trials will be used to evaluate the performance of the model. This can either be described as a list of trial numbers or, by passing - `"all"` for fitting all trials - `float('Nan')` or `"unrewarded"` for all those trials whose feedback was `float('Nan')` - `"rewarded"` for those who had feedback that was not `float('Nan')` Default `None`, which means all trials will be used.

Name
The name of the fitting type

Type string

See also:

fitAlgs.fitAlg.FitAlg The general fitting class

Methods Summary

find_name()	Returns the name of the class
fitness(*model_parameters)	Used by a fitter to generate the list of values characterising how well the model parameters describe the participants actions.
fitted_model(*model_parameters)	Simulating a model run with specific parameter values
get_model_parameters(*model_parameters)	Compiles the model parameter arguments based on the model parameters
get_model_properties(*model_parameters)	Compiles the kwarg model arguments based on the model_parameters and previously specified other parameters
info()	The dictionary describing the fitters algorithm chosen
participant_sequence_generation(...)	Finds the stimuli in the participant data and returns formatted observations
prepare_sim(model, model_parameters, ...)	Set up the simulation of a model following the behaviour of a participant

Methods Documentation

find_name()
Returns the name of the class

fitness(**model_parameters*)
Used by a fitter to generate the list of values characterising how well the model parameters describe the participants actions.

Parameters **model_parameters** (*list of floats*) – A list of the parameters used by the model in the order previously defined

Returns **model_performance** – The choices made by the model that will be used to characterise the quality of the fit.

Return type list of floats

See also:

fitAlgs.fitSims.FitSim.participant() Fits participant data

fitAlgs.fitAlg.fitAlg() The general fitting class

fitAlgs.fitAlg.fitAlg.fitness() The function that this one is called by

fitted_model(**model_parameters*)
Simulating a model run with specific parameter values

Parameters ***model_parameters** (*floats*) – The model parameters provided in the order defined in the model setup

Returns **model_instance**

Return type model.modelTemplate.Model class instance

get_model_parameters(**model_parameters*)
Compiles the model parameter arguments based on the model parameters

Parameters **model_parameters** (*list of floats*) – The parameter values in the order extracted from the modelSetup parameter dictionary

Returns **parameters** – The kwarg model parameter arguments

Return type dict

get_model_properties(**model_parameters*)
Compiles the kwarg model arguments based on the model_parameters and previously specified other parameters

Parameters **model_parameters** (*list of floats*) – The parameter values in the order extracted from the modelSetup parameter dictionary

Returns **model_properties** – The kwarg model arguments

Return type dict

info()
The dictionary describing the fitters algorithm chosen

Returns **fitInfo** – The dictionary of fitters class information

Return type dict

static participant_sequence_generation(*participant_data*, *choice_property*, *reward_property*, *stimuli_property*, *action_options_property*)
Finds the stimuli in the participant data and returns formatted observations

Parameters

- **participant_data** (*dict*) – The participant data
- **choice_property** (*string*) – The participant data key of their action choices.
- **reward_property** (*string*) – The participant data key of the participant reward data
- **stimuli_property** (*string or None or list of strings*) – A list of the keys in partData representing participant stimuli
- **action_options_property** (*string or None or list of strings, ints or None*) – The name of the key in partData where the list of valid actions can be found. If `None` then the action list is considered to stay constant. If a list then the list will be taken as the list of actions that can be taken at every trialstep. If the list is shorter than the number of trialsteps, then it will be considered to be a list of valid actions for each trialstep.

Returns **participant_sequence** – Each list element contains the observation, action and feedback for each trial taken by the participant

Return type list of three element tuples

prepare_sim(*model*, *model_parameters*, *model_properties*, *participant_data*)
Set up the simulation of a model following the behaviour of a participant

Parameters

- **model** (*model.modelTemplate.Model inherited class*) – The model you wish to try and fit values to
- **model_parameters** (*dict*) – The model initial parameters
- **model_properties** (*dict*) – The model static properties
- **participant_data** (*dict*) – The participant data

Returns

Return type fitness

FitSubsetError

exception fitAlgs.fitSims.**FitSubsetError**

StimuliError

exception fitAlgs.fitSims.**StimuliError**

Class Inheritance Diagram

Author Dominic Hunt

exception fitAlgs.fitSims.**ActionError**
Bases: exceptions.Exception

class fitAlgs.fitSims.**FitSim**(*participant_choice_property=u'Actions'*, *participant_reward_property=u'Rewards'*, *model_fitting_variable=u'ActionProb'*, *task_stimuli_property=None*, *fit_subset=None*, *action_options_property=None*, *float_error_response_value=1e-100*)
Bases: object

A class for fitting data by passing the participant data through the model.

This has been setup for fitting action-response models

Parameters

- **participant_choice_property** (*string, optional*) – The participant data key of their action choices. Default 'Actions'
- **participant_reward_property** (*string, optional*) – The participant data key of the participant reward data. Default 'Rewards'
- **model_fitting_variable** (*string, optional*) – The key to be compared in the model data. Default 'ActionProb'
- **task_stimuli_property** (*list of strings or None, optional*) – The keys containing the stimuli seen by the participant before taking a decision on an action. Default None

- **action_options_property** (*string or None or list of ints, optional*) – The name of the key in partData where the list of valid actions can be found. If `None` then the action list is considered to stay constant. If a list then the list will be taken as the list of actions that can be taken at each instance. Default `None`
- **float_error_response_value** (*float, optional*) – If a floating point error occurs when running a fit the fitter function will return a value for each element of fpRespVal. Default is `1/1e100`
- **fit_subset** (`float('Nan')`, `None`, `"rewarded"`, `"unrewarded"`, `"all"` or list of int, optional) – Describes which, if any, subset of trials will be used to evaluate the performance of the model. This can either be described as a list of trial numbers or, by passing - `"all"` for fitting all trials - `float('Nan')` or `"unrewarded"` for all those trials whose feedback was `float('Nan')` - `"rewarded"` for those who had feedback that was not `float('Nan')` Default `None`, which means all trials will be used.

Name
The name of the fitting type

Type string

See also:

fitAlgs.fitAlg.FitAlg The general fitting class

find_name()
Returns the name of the class

fitness(**model_parameters*)
Used by a fitter to generate the list of values characterising how well the model parameters describe the participants actions.

Parameters **model_parameters** (*list of floats*) – A list of the parameters used by the model in the order previously defined

Returns **model_performance** – The choices made by the model that will be used to characterise the quality of the fit.

Return type list of floats

See also:

fitAlgs.fitSims.FitSim.participant() Fits participant data

fitAlgs.fitAlg.fitAlg() The general fitting class

fitAlgs.fitAlg.fitAlg.fitness() The function that this one is called by

fitted_model(**model_parameters*)
Simulating a model run with specific parameter values

Parameters ***model_parameters** (*floats*) – The model parameters provided in the order defined in the model setup

Returns **model_instance**

Return type model.modelTemplate.Model class instance

get_model_parameters(**model_parameters*)
Compiles the model parameter arguments based on the model parameters

Parameters **model_parameters** (*list of floats*) – The parameter values in the order extracted from the modelSetup parameter dictionary

Returns **parameters** – The kwarg model parameter arguments

Return type dict

get_model_properties(**model_parameters*)
Compiles the kwarg model arguments based on the model_parameters and previously specified other parameters

Parameters **model_parameters** (*list of floats*) – The parameter values in the order extracted from the modelSetup parameter dictionary

Returns **model_properties** – The kwarg model arguments

Return type dict

info()
The dictionary describing the fitters algorithm chosen

Returns **fitInfo** – The dictionary of fitters class information

Return type dict

static participant_sequence_generation(*participant_data*, *choice_property*, *reward_property*, *stimuli_property*, *action_options_property*)
Finds the stimuli in the participant data and returns formatted observations

Parameters

- **participant_data** (*dict*) – The participant data
- **choice_property** (*string*) – The participant data key of their action choices.
- **reward_property** (*string*) – The participant data key of the participant reward data
- **stimuli_property** (*string or None or list of strings*) – A list of the keys in partData representing participant stimuli
- **action_options_property** (*string or None or list of strings, ints or None*) – The name of the key in partData where the list of valid actions can be found. If `None` then the action list is considered to stay constant. If a list then the list will be taken as the list of actions that can be taken at every trialstep. If the list is shorter than the number of trialsteps, then it will be considered to be a list of valid actions for each trialstep.

Returns **participant_sequence** – Each list element contains the observation, action and feedback for each trial taken by the participant

Return type list of three element tuples

prepare_sim(*model*, *model_parameters*, *model_properties*, *participant_data*)
Set up the simulation of a model following the behaviour of a participant

Parameters

- **model** (*model.modelTemplate.Model inherited class*) – The model you wish to try and fit values to
- **model_parameters** (*dict*) – The model initial parameters
- **model_properties** (*dict*) – The model static properties
- **participant_data** (*dict*) – The participant data

Returns

Return type fitness

exception fitAlgs.fitSims.**FitSubsetError**
Bases: exceptions.Exception

exception fitAlgs.fitSims.**StimuliError**
Bases: exceptions.Exception

6.8.2.6 fitAlgs.leastsq module

Author Dominic Hunt

class fitAlgs.leastsq.**Leastsq**(*method=u'dogbox'*, *jacobian_method=u'3-point'*, ***kwargs*)
Bases: *fitAlgs.fitAlg.FitAlg*

Fits data based on the least squared optimizer scipy.optimize.least_squares

Not properly developed and will not be documented until upgrade

Parameters

- **fit_sim** (*fitAlgs.fitSims.FitSim instance, optional*) – An instance of one of the fitting simulation methods. Default `fitAlgs.fitSims.FitSim`
- **fit_measure** (*string, optional*) – The name of the function used to calculate the quality of the fit. The value it returns provides the fitter with its fitting guide. Default `-loge`
- **fit_measure_args** (*dict, optional*) – The parameters used to initialise fit_measure and extra_fit_measures. Default `None`
- **extra_fit_measures** (*list of strings, optional*) – List of fit measures not used to fit the model, but to provide more information. Any arguments needed for these measures should be placed in fit_measure_args. Default `None`
- **bounds** (*dictionary of tuples of length two with floats, optional*) – The boundaries for methods that use bounds. If unbounded methods are specified then the bounds will be ignored. Default is `None`, which translates to boundaries of (0, np.inf) for each parameter.
- **boundary_excess_cost** (*basestring or callable returning a function, optional*) – The function is used to calculate the penalty for exceeding the boundaries. Default is `boundFunc.scalarBound()`
- **boundary_excess_cost_properties** (*dict, optional*) – The parameters for the boundary_excess_cost function. Default {}
- **method** (*{'trf', 'dogbox', 'lm'}, optional*) – Algorithm to perform minimization. Default `dogbox`

Name
The name of the fitting method

Type string

See also:

fitAlgs.fitAlg.fitAlg The general fitting method class, from which this one inherits

fitAlgs.fitSims.fitSim The general fitting class

scipy.optimize.least_squares The fitting class this wraps around

fit(*simulator*, *model_parameter_names*, *model_initial_parameters*)
Runs the model through the fitting algorithms and starting parameters and returns the best one.

Parameters

- **simulator** (*function*) – The function used by a fitting algorithm to generate a fit for given model parameters. One example is fitAlg.fitness
- **model_parameter_names** (*list of strings*) – The list of initial parameter names
- **model_initial_parameters** (*list of floats*) – The list of the initial parameters

Returns

- **fitParams** (*list of floats*) – The best fitting parameters
- **fit_quality** (*float*) – The quality of the fit as defined by the quality function chosen.
- **testedParams** (*tuple of two lists*) – The two lists are a list containing the parameter values tested, in the order they were tested, and the fit qualities of these parameters.

See also:

`fitAlgs.fitAlg.fitness()`

6.8.2.7 fitAlgs.minimize module

Author Dominic Hunt

class fitAlgs.minimize.**Minimize**(*method=None*, *number_start_points=4*, *allow_boundary_fits=True*, *boundary_fit_sensitivity=5*, ***kwargs*)

Bases: *fitAlgs.fitAlg.FitAlg*

The class for fitting data using scipy.optimise.minimize

Parameters

- **fit_sim** (*fitAlgs.fitSims.FitSim instance, optional*) – An instance of one of the fitting simulation methods. Default `fitAlgs.fitSims.FitSim`
- **fit_measure** (*string, optional*) – The name of the function used to calculate the quality of the fit. The value it returns provides the fitter with its fitting guide. Default `-loge`
- **fit_measure_args** (*dict, optional*) – The parameters used to initialise fit_measure and extra_fit_measures. Default `None`
- **extra_fit_measures** (*list of strings, optional*) – List of fit measures not used to fit the model, but to provide more information. Any arguments needed for these measures should be placed in fit_measure_args. Default `None`
- **bounds** (*dictionary of tuples of length two with floats, optional*) – The boundaries for methods that use bounds. If unbounded methods are specified then the bounds will be ignored. Default is `None`, which translates to boundaries of (0, np.inf) for each parameter.
- **boundary_excess_cost** (*basestring or callable returning a function, optional*) – The function is used to calculate the penalty for exceeding the boundaries. Default is `boundFunc.scalarBound()`
- **boundary_excess_cost_properties** (*dict, optional*) – The parameters for the boundary_excess_cost function. Default {}
- **method** (*string or list of strings, optional*) – The name of the fitting method or list of names of fitting methods or name of list of fitting methods. Valid names found in the notes. Default `unconstrained`
- **number_start_points** (*int, optional*) – The number of starting points generated for each parameter. Default 4
- **allow_boundary_fits** (*bool, optional*) – Defines if fits that reach a boundary should be considered the same way as those that do not. Default is True
- **boundary_fit_sensitivity** (*int, optional*) – Defines the smallest number of decimal places difference (so the minimal difference) between a parameter value and its related boundaries before a parameter value is considered different from a boundary. The default is *5*. This is only valid if `allow_boundary_fits` is `False`

Name
The name of the fitting method

Type string

unconstrained
The list of valid unconstrained fitting methods

Type list

constrained
The list of valid constrained fitting methods

Type list

Notes

unconstrained = ['Nelder-Mead','Powell','CG','BFGS'] constrained = ['L-BFGS-B','TNC','SLSQP'] Custom fitting algorithms are also allowed in theory, but it has yet to be implemented.

For each fitting function a set of different starting parameters will be tried. These are the combinations of all the values of the different parameters. For each starting parameter provided a set of number_start_points starting points will be chosen, surrounding the starting point provided. If the starting point provided is less than one it will be assumed that the values cannot exceed 1, otherwise, unless otherwise told, it will be assumed that they can take any value and will be chosen to be eavenly spaced around the provided value.

See also:

fitAlgs.fitAlg.fitAlg The general fitting method class, from which this one inherits

fitAlgs.fitSims.fitSim The general fitSim class

scipy.optimise.minimize The fitting class this wraps around

constrained = [u'L-BFGS-B', u'TNC', u'SLSQP']

fit(*simulator*, *model_parameter_names*, *model_initial_parameters*)
Runs the model through the fitting algorithms and starting parameters and returns the best one.

Parameters

- **simulator** (*function*) – The function used by a fitting algorithm to generate a fit for given model parameters. One example is fitAlgs.fitAlg.fitness
- **model_parameter_names** (*list of strings*) – The list of initial parameter names
- **model_initial_parameters** (*list of floats*) – The list of the initial parameters

Returns

- **best_fit_parameters** (*list of floats*) – The best fitting parameters
- **fit_quality** (*float*) – The quality of the fit as defined by the quality function chosen.
- **testedParams** (*tuple of two lists*) – The two lists are a list containing the parameter values tested, in the order they were tested, and the fit qualities of these parameters.

See also:

`fitAlgs.fitAlg.fitness()`

unconstrained = [u'Nelder-Mead', u'Powell', u'CG', u'BFGS']

6.8.2.8 fitAlgs.qualityFunc module

fitAlgs.qualityFunc Module

Author Dominic Hunt

Functions

BIC2(**kwargs)	Generates a function that calculates the Bayesian Information Criterion (BIC)
BIC2norm(**kwargs)	**param numParams** The number of parameters used by the model used for the fits process. Default 2
BIC2normBoot(**kwargs)	An attempt at looking what would happen if the samples were resampled.
WBIC2(**kwargs)	Unfinished WBIC implementation
bayesFactor(**kwargs)	:math:‘2^{
bayesInv(**kwargs)	**param numParams** The number of parameters used by the model used for the fitters process. Default 2
bayesRand(**kwargs)	
logAverageProb(modVals)	Generates a fit quality value based on $\sum -2\log_2(\vec{x})$
logeprob(modVals)	Generates a fit quality value based on :math:‘f_{mathrm{mod}}left(vec x
logprob(modVals)	Generates a fit quality value based on :math:‘f_{mathrm{mod}}left(vec x
maxprob(modVals)	Generates a fit quality value based on $\sum 1 - \vec{x}$
qualFuncIdent(value, **kwargs)	
r2(**kwargs)	
simpleSum(modVals)	Generates a fit quality value based on $\sum \vec{x}$

BIC2

fitAlgs.qualityFunc.**BIC2**(***kwargs*)

Generates a function that calculates the Bayesian Information Criterion (BIC)

:math:‘lambda mathrm{log}_2(T)+ f_{mathrm{mod}}left(vec x

ight)‘

kwargs

BIC2norm

fitAlgs.qualityFunc.**BIC2norm**(***kwargs*)

Parameters

- **numParams** (*int, optional*) – The number of parameters used by the model used for the fits process. Default 2

- **qualityThreshold** (*float, optional*) – The BIC minimum fit quality criterion used for determining if a fit is valid. Default 20.0
- **number_actions** (*int or list of ints the length of the number of trials being fitted, optional*) – The number of actions the participant can choose between for each trialstep of the task. May need to be specified for each trial if the number of action choices varies between trials. Default 2
- **randActProb** (*float or list of floats the length of the number of trials being fitted. Optional*) – The prior probability of an action being randomly chosen. May need to be specified for each trial if the number of action choices varies between trials. Default `1/number_actions`

BIC2normBoot

`fitAlgs.qualityFunc.`**`BIC2normBoot`**(***kwargs*)

An attempt at looking what would happen if the samples were resampled. It was hoped that by doing this, the difference between different sample distributions would become more pronounced. This was not found to be true.

Parameters

- **numParams** (*int, optional*) – The number of parameters used by the model used for the fits process. Default 2
- **qualityThreshold** (*float, optional*) – The BIC minimum fit quality criterion used for determining if a fit is valid. Default 20.0
- **number_actions** (*int or list of ints the length of the number of trials being fitted, optional*) – The number of actions the participant can choose between for each trialstep of the task. May need to be specified for each trial if the number of action choices varies between trials. Default 2
- **randActProb** (*float or list of floats the length of the number of trials being fitted. Optional*) – The prior probability of an action being randomly chosen. May need to be specified for each trial if the number of action choices varies between trials. Default `1/number_actions`
- **numSamples** (*int, optional*) – The number of samples that will be randomly resampled from `modVals`. Default 100
- **sampleLen** (*int, optional*) – The length of the random sample. Default 1

WBIC2

`fitAlgs.qualityFunc.`**`WBIC2`**(***kwargs*)

Unfinished WBIC implementation

bayesFactor

`fitAlgs.qualityFunc.`**`bayesFactor`**(***kwargs*)

:math:‘2^{

rac{x}{2}}‘

kwargs

bayesInv

fitAlgs.qualityFunc.**bayesInv**(***kwargs*)

Parameters

- **numParams** (*int, optional*) – The number of parameters used by the model used for the fitters process. Default 2
- **qualityThreshold** (*float, optional*) – The BIC minimum fit quality criterion used for determining if a fit is valid. Default 20.0
- **number_actions** (*int or list of ints the length of the number of trials being fitted, optional*) – The number of actions the participant can choose between for each trialstep of the task. May need to be specified for each trial if the number of action choices varies between trials. Default 2
- **randActProb** (*float or list of floats the length of the number of trials being fitted. Optional*) – The prior probability of an action being randomly chosen. May need to be specified for each trial if the number of action choices varies between trials. Default `1/number_actions`

bayesRand

fitAlgs.qualityFunc.**bayesRand**(***kwargs*)

logAverageProb

fitAlgs.qualityFunc.**logAverageProb**(*modVals*)

Generates a fit quality value based on $\sum -2\log_2(\vec{x})$

Returns **fit** – The sum of the model values returned

Return type float

logeprob

fitAlgs.qualityFunc.**logeprob**(*modVals*)

Generates a fit quality value based on :math:‘f_{mathrm{mod}}left(vec x

ight) = sum -mathrm{log}_e(vec x)‘

fit [float] The sum of the model values returned

logprob

fitAlgs.qualityFunc.**logprob**(*modVals*)

Generates a fit quality value based on :math:‘f_{mathrm{mod}}left(vec x

ight) = sum -2mathrm{log}_2(vec x)‘

fit [float] The sum of the model values returned

maxprob

fitAlgs.qualityFunc.**maxprob**(*modVals*)
Generates a fit quality value based on $\sum 1 - \vec{x}$

Returns **fit** – The sum of the model values returned

Return type float

qualFuncIdent

fitAlgs.qualityFunc.**qualFuncIdent**(*value*, ***kwargs*)

r2

fitAlgs.qualityFunc.**r2**(***kwargs*)

simpleSum

fitAlgs.qualityFunc.**simpleSum**(*modVals*)
Generates a fit quality value based on $\sum \vec{x}$

Returns **fit** – The sum of the model values returned

Return type float

Author Dominic Hunt

fitAlgs.qualityFunc.**BIC2**(***kwargs*)

Generates a function that calculates the Bayesian Information Criterion (BIC)

:math:‘lambda mathrm{log}_2(T)+ f_{mathrm{mod}}left(vec x

ight)‘

kwargs

fitAlgs.qualityFunc.**BIC2norm**(***kwargs*)

Parameters

- **numParams** (*int, optional*) – The number of parameters used by the model used for the fits process. Default 2
- **qualityThreshold** (*float, optional*) – The BIC minimum fit quality criterion used for determining if a fit is valid. Default 20.0
- **number_actions** (*int or list of ints the length of the number of trials being fitted, optional*) – The number of actions the participant can choose between for each trialstep of the task. May need to be specified for each trial if the number of action choices varies between trials. Default 2
- **randActProb** (*float or list of floats the length of the number of trials being fitted. Optional*) – The prior probability of an action being randomly chosen. May need to be specified for each trial if the number of action choices varies between trials. Default `1/number_actions`

fitAlgs.qualityFunc.**BIC2normBoot**(***kwargs*)
An attempt at looking what would happen if the samples were resampled. It was hoped that by doing this, the difference between different sample distributions would become more pronounced. This was not found to be true.

Parameters

- **numParams** (*int, optional*) – The number of parameters used by the model used for the fits process. Default 2
- **qualityThreshold** (*float, optional*) – The BIC minimum fit quality criterion used for determining if a fit is valid. Default 20.0
- **number_actions** (*int or list of ints the length of the number of trials being fitted, optional*) – The number of actions the participant can choose between for each trialstep of the task. May need to be specified for each trial if the number of action choices varies between trials. Default 2
- **randActProb** (*float or list of floats the length of the number of trials being fitted. Optional*) – The prior probability of an action being randomly chosen. May need to be specified for each trial if the number of action choices varies between trials. Default `1/number_actions`
- **numSamples** (*int, optional*) – The number of samples that will be randomly resampled from `modVals`. Default 100
- **sampleLen** (*int, optional*) – The length of the random sample. Default 1

`fitAlgs.qualityFunc.`**`WBIC2`**(***kwargs*)
Unfinished WBIC implementation

`fitAlgs.qualityFunc.`**`bayesFactor`**(***kwargs*)

:math:‘2^{

rac{x}{2}}‘

kwargs

`fitAlgs.qualityFunc.`**`bayesInv`**(***kwargs*)

Parameters

- **numParams** (*int, optional*) – The number of parameters used by the model used for the fitters process. Default 2
- **qualityThreshold** (*float, optional*) – The BIC minimum fit quality criterion used for determining if a fit is valid. Default 20.0
- **number_actions** (*int or list of ints the length of the number of trials being fitted, optional*) – The number of actions the participant can choose between for each trialstep of the task. May need to be specified for each trial if the number of action choices varies between trials. Default 2
- **randActProb** (*float or list of floats the length of the number of trials being fitted. Optional*) – The prior probability of an action being randomly chosen. May need to be specified for each trial if the number of action choices varies between trials. Default `1/number_actions`

`fitAlgs.qualityFunc.`**`bayesRand`**(***kwargs*)

`fitAlgs.qualityFunc.`**`logAverageProb`**(*modVals*)
Generates a fit quality value based on $\sum -2\log_2(\vec{x})$

Returns **fit** – The sum of the model values returned

Return type float

`fitAlgs.qualityFunc.`**`logeprob`**(*modVals*)

Generates a fit quality value based on :math:‘f_{mathrm{mod}}left(vec x

ight) = sum -mathrm{log}_e(vec x)‘

fit [float] The sum of the model values returned

`fitAlgs.qualityFunc.`**`logprob`**(*modVals*)

Generates a fit quality value based on :math:‘f_{mathrm{mod}}left(vec x

ight) = sum -2mathrm{log}_2(vec x)‘

fit [float] The sum of the model values returned

fitAlgs.qualityFunc.**maxprob**(*modVals*)
Generates a fit quality value based on $\sum 1 - \vec{x}$

Returns **fit** – The sum of the model values returned

Return type float

fitAlgs.qualityFunc.**qualFuncIdent**(*value*, ***kwargs*)

fitAlgs.qualityFunc.**r2**(***kwargs*)

fitAlgs.qualityFunc.**simpleSum**(*modVals*)
Generates a fit quality value based on $\sum \vec{x}$

Returns **fit** – The sum of the model values returned

Return type float

6.9 outputting module

6.9.1 outputting Module

Author Dominic Hunt

6.9.1.1 Functions

date()	Calculate today's date as a string in the form <year>-<month>-<day> and returns it
dictKeyGen(store[, maxListLen, returnList, ...])	Identifies the columns necessary to convert a dictionary into a table
fancy_logger([log_file, log_level, ...])	Sets up the style of logging for all the simulations
file_name_generator([output_folder])	Keeps track of filenames that have been used and generates the next unused one
flatDictKeySet(store[, selectKeys])	Generates a dictionary of keys and identifiers for the new dictionary, including only the keys in the keys list.
folder_path_cleaning(folder)	Modifies string file names from Windows format to Unix format if necessary and makes sure there is a / at the end.
folder_setup(label, date_string[, ...])	Identifies and creates the folder the data will be stored in
listKeyGen(data[, maxListLen, returnList, ...])	Identifies the columns necessary to convert a list into a table
listSelection(data, loc)	Allows numpy array-like referencing of lists
newFlatDict(store[, selectKeys, labelPrefix])	Takes a list of dictionaries and returns a dictionary of 1D lists.
newListDict(store[, labelPrefix, maxListLen])	Takes a dictionary of numbers, strings, lists and arrays and returns a dictionary of 1D arrays.
pad(values, maxListLen)	Pads a list with None
pickleLog(results, file_name_gen[, label])	Stores the data in the appropriate pickle file in a Pickle subfolder of the outputting folder
pickle_write(data, handle, file_name_gen)	Writes the data to a pickle file

date

outputting.**date**()
Calculate today's date as a string in the form <year>-<month>-<day> and returns it

Returns **todayDate** – The current date in the format <year>-<month>-<day>

Return type basestring

dictKeyGen

outputting.**dictKeyGen**(*store*, *maxListLen=None*, *returnList=False*, *abridge=False*)
Identifies the columns necessary to convert a dictionary into a table

Parameters

- **store** (*dict*) – The dictionary to be broken down into keys
- **maxListLen** (*int or float with no decimal places or None, optional*) – The length of the longest expected list. Only useful if returnList is `True`. Default `None`
- **returnList** (*bool, optional*) – Defines if the lists will be broken into 1D lists or values. Default `False`, lists will be broken into values
- **abridge** (*bool, optional*) – Defines if the final dataset will be a summary or the whole lot. If it is a summary, lists of more than 10 elements are removed. Default `False`, not abridged

Returns

- **keySet** (*OrderedDict with values of OrderedDict, list or None*) – The dictionary of keys to be extracted
- **maxListLen** (*int or float with no decimal places or None, optional*) – If returnList is `True` this should be the length of the longest list. If returnList is `False` this should return its original value

Examples

```
>>> store = {'string': 'string'}
>>> dictKeyGen(store)
(OrderedDict([('string', None)]), 1)
>>> store = {'num': 23.6}
>>> dictKeyGen(store)
(OrderedDict([('num', None)]), 1)
>>> store = {'array': np.array([[1, 2, 3, 4, 5, 6], [7, 8, 9, 10, 11, 12]])}
>>> dictKeyGen(store, returnList=True, abridge=True)
(OrderedDict([(u'array', array([[0],
       [1]]))]), 6L)
>>> store = {'dict': {1: "a", 2: "b"}}
>>> dictKeyGen(store, maxListLen=7, returnList=True, abridge=True)
(OrderedDict([('dict', OrderedDict([(1, None), (2, None)]))]), 7)
```

fancy_logger

outputting.**fancy_logger**(*log_file=None*, *log_level=20*, *numpy_error_level=u'log'*)
Sets up the style of logging for all the simulations

Parameters

- **date_string** (*basestring*) – The date the log will start at
- **log_file** (*string, optional*) – Provides the path the log will be written to. Default "./log.txt"
- **log_level** (*{logging.DEBUG, logging.INFO, logging.WARNING, logging.ERROR, logging.CRITICAL}*) – Defines the level of the log. Default logging.INFO
- **numpy_error_level** (*{'log', 'raise'}*) – Defines the response to numpy errors. Default `log`. See numpy.seterr

Returns **close_loggers** – Closes the logging systems that have been set up

Return type function

See also:

logging() The Python standard logging library

numpy.seterr() The function npErrResp is passed to for defining the response to numpy errors

file_name_generator

outputting.**file_name_generator**(*output_folder=None*)

Keeps track of filenames that have been used and generates the next unused one

Parameters **output_folder** (*string, optional*) – The folder into which the new file will be placed. Default is the current working directory

Returns **new_file_name** – Creates a new file with the name <handle> and the extension <extension>. It takes two string parameters: (`handle`, `extension`) and returns one `fileName` string

Return type function

Examples

```
>>> file_name_gen = file_name_generator("./")
>>> file_name_gen("a", "b")
'./a.b'
>>> file_name_gen("a", "b")
'./a_1.b'
>>> file_name_gen("", "")
'./'
>>> file_name_gen = file_name_generator()
>>> fileName = file_name_gen("", "")
>>> fileName == os.getcwd()
False
```

flatDictKeySet

outputting.**flatDictKeySet**(*store*, *selectKeys=None*)

Generates a dictionary of keys and identifiers for the new dictionary, including only the keys in the keys list. Any keys with lists will be split into a set of keys, one for each element in the original key.

These are named <key><location>

Parameters

- **store** (*list of dicts*) – The dictionaries would be expected to have many of the same keys. Any dictionary keys containing lists in the input have been split into multiple numbered keys
- **selectKeys** (*list of strings, optional*) – The keys whose data will be included in the return dictionary. Default `None`, which results in all keys being returned

Returns **keySet** – The dictionary of keys to be extracted

Return type OrderedDict with values of OrderedDict, list or None

See also:

`reframeListDicts()`, *newFlatDict()*

folder_path_cleaning

`outputting.`**`folder_path_cleaning`**(*folder*)

Modifies string file names from Windows format to Unix format if necessary and makes sure there is a `/` at the end.

Parameters **folder** (*string*) – The folder path

Returns **folder_path** – The folder path

Return type basestring

folder_setup

`outputting.`**`folder_setup`**(*label*, *date_string*, *pickle_data=False*, *base_path=None*)

Identifies and creates the folder the data will be stored in

Folder will be created as "./Outputs/<sim_label>_<date>/". If that had previously been created then it is created as "./Outputs/<sim_label>_<date>_no_<#>/", where "<#>" is the first available integer.

A subfolder is also created with the name `Pickle` if pickle is true.

Parameters

- **label** (*basestring*) – The label for the simulation
- **date_string** (*basestring*) – The date identifier
- **pickle_data** (*bool, optional*) – If true the data for each model, task and participant is recorded. Default is `False`
- **base_path** (*basestring, optional*) – The path into which the new folder will be placed. Default is current working directory

Returns **folder_name** – The folder path that has just been created

Return type string

See also:

`newFile()` Creates a new file

`saving()` Creates the log system

listKeyGen

`outputting.`**`listKeyGen`**(*data*, *maxListLen=None*, *returnList=False*, *abridge=False*)

Identifies the columns necessary to convert a list into a table

Parameters

- **data** (*numpy.ndarray or list*) – The list to be broken down
- **maxListLen** (*int or float with no decimal places or None, optional*) – The length of the longest expected list. Only useful if returnList is `True`. Default `None`
- **returnList** (*bool, optional*) – Defines if the lists will be broken into 1D lists or values. Default `False`, lists will be broken into values
- **abridge** (*bool, optional*) – Defines if the final dataset will be a summary or the whole lot. If it is a summary, lists of more than 10 elements are removed. Default `False`, not abridged

Returns

- **returnList** (*None or list of tuples of ints or ints*) – The list of co-ordinates for the elements to be extracted from the data. If None the list is used as-is.
- **maxListLen** (*int or float with no decimal places or None, optional*) – If returnList is `True` this should be the length of the longest list. If returnList is `False` this should return its original value

Examples

```
>>> listKeyGen([[1, 2, 3, 4, 5, 6], [4, 5, 6, 7, 8, 9]], maxListLen=None,␣
↪returnList=False, abridge=False)
(array([[0, 0], [1, 0], [0, 1], [1, 1], [0, 2], [1, 2], [0, 3], [1, 3], [0, 4],
↪ [1, 4], [0, 5], [1, 5]]), 1)
>>> listKeyGen([[1, 2, 3, 4, 5, 6], [4, 5, 6, 7, 8, 9]], maxListLen=None,␣
↪returnList=False, abridge=True)
(None, None)
>>> listKeyGen([[1, 2, 3, 4, 5, 6], [4, 5, 6, 7, 8, 9]], maxListLen=None,␣
↪returnList=True, abridge=True)
(array([[0],
       [1]]), 6L)
```

listSelection

`outputting.`**`listSelection`**(*data*, *loc*)

Allows numpy array-like referencing of lists

Parameters

- **data** (*list*) – The data to be referenced
- **loc** (*tuple of integers*) – The location to be referenced

Returns **selection** – The referenced subset

Return type list

Examples

```
>>> listSelection([1, 2, 3], (0,))
1
>>> listSelection([[1, 2, 3], [4, 5, 6]], (0,))
[1, 2, 3]
>>> listSelection([[1, 2, 3], [4, 5, 6]], (0, 2))
3
```

newFlatDict

outputting.**newFlatDict**(*store*, *selectKeys=None*, *labelPrefix=u''*)

Takes a list of dictionaries and returns a dictionary of 1D lists.

If a dictionary did not have that key or list element, then 'None' is put in its place

Parameters

- **store** (*list of dicts*) – The dictionaries would be expected to have many of the same keys. Any dictionary keys containing lists in the input have been split into multiple numbered keys
- **selectKeys** (*list of strings, optional*) – The keys whose data will be included in the return dictionary. Default `None`, which results in all keys being returned
- **labelPrefix** (*string*) – An identifier to be added to the beginning of each key string.

Returns **newStore** – The new dictionary with the keys from the keySet and the values as 1D lists with 'None' if the keys, value pair was not found in the store.

Return type dict

Examples

```
>>> store = [{'list': [1, 2, 3, 4, 5, 6]}]
>>> newFlatDict(store)
OrderedDict([('list_[0]', [1]), ('list_[1]', [2]), ('list_[2]', [3]), ('list_
↪[3]', [4]), ('list_[4]', [5]), ('list_[5]', [6])])
>>> store = [{'string': 'string'}]
>>> newFlatDict(store)
OrderedDict([(u'string', ["u'string'"])])
>>> store = [{'dict': {1: {3: "a"}, 2: "b"}}]
>>> newFlatDict(store)
OrderedDict([(u'dict_1_3', ["u'a'"]), (u'dict_2', ["u'b'"])])
```

newListDict

outputting.**newListDict**(*store*, *labelPrefix=u''*, *maxListLen=0*)

Takes a dictionary of numbers, strings, lists and arrays and returns a dictionary of 1D arrays.

If there is a single value, then a list is created with that value repeated

Parameters

- **store** (*dict*) – A dictionary of numbers, strings, lists, dictionaries and arrays
- **labelPrefix** (*string*) – An identifier to be added to the beginning of each key string. Default empty string

Returns **newStore** – The new dictionary with the keys from the keySet and the values as 1D lists.

Return type dict

Examples

```
>>> store = {'list': [1, 2, 3, 4, 5, 6]}
>>> newListDict(store)
OrderedDict([('list', [1, 2, 3, 4, 5, 6])])
>>> store = {'string': 'string'}
>>> newListDict(store)
OrderedDict([('string', ['string'])])
>>> store = {'dict': {1: {3: "a"}, 2: "b"}}
>>> newListDict(store)
OrderedDict([(u'dict_1_3', ['a']), (u'dict_2', ['b'])])
```

pad

outputting.**pad**(*values*, *maxListLen*)

Pads a list with None

Parameters

- **values** (*list*) – The list to be extended
- **maxListLen** (*int*) – The number of elements the list needs to have

pickleLog

outputting.**pickleLog**(*results*, *file_name_gen*, *label=u''*)

Stores the data in the appropriate pickle file in a Pickle subfolder of the outputting folder

Parameters

- **results** (*dict*) – The data to be stored
- **file_name_gen** (*function*) – Creates a new file with the name <handle> and the extension <extension>. It takes two string parameters: (`handle`, `extension`) and returns one `fileName` string
- **label** (*string, optional*) – A label for the results file

pickle_write

outputting.**pickle_write**(*data*, *handle*, *file_name_gen*)

Writes the data to a pickle file

Parameters

- **data** (*object*) – Data to be written to the file
- **handle** (*string*) – The name of the file
- **file_name_gen** (*function*) – Creates a new file with the name <handle> and the extension <extension>. It takes two string parameters: (`handle`, `extension`) and returns one `fileName` string

6.9.1.2 Classes

LoggerWriter(writer)	Fake file-like stream object that redirects writes to a logger instance.
Saving([label, output_path, config, ...])	Creates the folder structure for the saved data and created the log file as `log.txt`

LoggerWriter

class outputting.**LoggerWriter**(*writer*)

Bases: object

Fake file-like stream object that redirects writes to a logger instance. Taken from https://stackoverflow.com/a/51612402

Parameters **writer** (*logging function*) –

Methods Summary

flush()
write(message)

Methods Documentation

flush()

write(*message*)

Saving

class outputting.**Saving**(*label=None*, *output_path=None*, *config=None*, *config_file=None*, *pickle_store=False*, *min_log_level=u'INFO'*, *numpy_error_level=u'log'*)

Bases: object

Creates the folder structure for the saved data and created the log file as `log.txt`

Parameters

- **label** (*string, optional*) – The label for the simulation. Default `None` will mean no data is saved to files.
- **output_path** (*string, optional*) – The path that will be used for the run output. Default `None`
- **config** (*dict, optional*) – The parameters of the running simulation/fitting. This is used to create a YAML configuration file. Default `None`
- **config_file** (*string, optional*) – The file name and path of a `.yaml` configuration file. Default `None`
- **pickle_store** (*bool, optional*) – If true the data for each model, task and participant is recorded. Default is `False`
- **min_log_level** (*basestring, optional*) – Defines the level of the log from (`DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`). Default `INFO` See https://docs.python.org/3/library/logging.html#levels
- **numpy_error_level** (*{'log', 'raise'}*) – Defines the response to numpy errors. Default `log`. See numpy.seterr

Returns **file_name_gen** – Creates a new file with the name <handle> and the extension <extension>. It takes two string parameters: (`handle`, `extension`) and returns one `fileName` string

Return type function

See also:

folderSetup creates the folders

6.9.1.3 Class Inheritance Diagram

Author Dominic Hunt

class outputting.**LoggerWriter**(*writer*)
Bases: object

Fake file-like stream object that redirects writes to a logger instance. Taken from https://stackoverflow.com/a/51612402

Parameters **writer** (*logging function*) –

flush()

write(*message*)

class outputting.**Saving**(*label=None*, *output_path=None*, *config=None*, *config_file=None*, *pickle_store=False*, *min_log_level=u'INFO'*, *numpy_error_level=u'log'*)
Bases: object

Creates the folder structure for the saved data and created the log file as `log.txt`

Parameters

- **label** (*string, optional*) – The label for the simulation. Default `None` will mean no data is saved to files.
- **output_path** (*string, optional*) – The path that will be used for the run output. Default `None`
- **config** (*dict, optional*) – The parameters of the running simulation/fitting. This is used to create a YAML configuration file. Default `None`
- **config_file** (*string, optional*) – The file name and path of a `.yaml` configuration file. Default `None`
- **pickle_store** (*bool, optional*) – If true the data for each model, task and participant is recorded. Default is `False`
- **min_log_level** (*basestring, optional*) – Defines the level of the log from (`DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`). Default `INFO` See https://docs.python.org/3/library/logging.html#levels
- **numpy_error_level** (*{'log', 'raise'}*) – Defines the response to numpy errors. Default `log`. See numpy.seterr

Returns **file_name_gen** – Creates a new file with the name <handle> and the extension <extension>. It takes two string parameters: (`handle`, `extension`) and returns one `fileName` string

Return type function

See also:

folderSetup creates the folders

outputting.**date**()
Calculate today's date as a string in the form <year>-<month>-<day> and returns it

Returns **todayDate** – The current date in the format <year>-<month>-<day>

Return type basestring

outputting.**dictKeyGen**(*store*, *maxListLen=None*, *returnList=False*, *abridge=False*)
Identifies the columns necessary to convert a dictionary into a table

Parameters

- **store** (*dict*) – The dictionary to be broken down into keys
- **maxListLen** (*int or float with no decimal places or None, optional*) – The length of the longest expected list. Only useful if returnList is `True`. Default `None`
- **returnList** (*bool, optional*) – Defines if the lists will be broken into 1D lists or values. Default `False`, lists will be broken into values
- **abridge** (*bool, optional*) – Defines if the final dataset will be a summary or the whole lot. If it is a summary, lists of more than 10 elements are removed. Default `False`, not abridged

Returns

- **keySet** (*OrderedDict with values of OrderedDict, list or None*) – The dictionary of keys to be extracted
- **maxListLen** (*int or float with no decimal places or None, optional*) – If returnList is `True` this should be the length of the longest list. If returnList is `False` this should return its original value

Examples

```
>>> store = {'string': 'string'}
>>> dictKeyGen(store)
(OrderedDict([('string', None)]), 1)
>>> store = {'num': 23.6}
>>> dictKeyGen(store)
(OrderedDict([('num', None)]), 1)
>>> store = {'array': np.array([[1, 2, 3, 4, 5, 6], [7, 8, 9, 10, 11, 12]])}
>>> dictKeyGen(store, returnList=True, abridge=True)
(OrderedDict([(u'array', array([[0],
       [1]]))]), 6L)
>>> store = {'dict': {1: "a", 2: "b"}}
>>> dictKeyGen(store, maxListLen=7, returnList=True, abridge=True)
(OrderedDict([('dict', OrderedDict([(1, None), (2, None)]))]), 7)
```

outputting.**fancy_logger**(*log_file=None*, *log_level=20*, *numpy_error_level=u'log'*)

Sets up the style of logging for all the simulations

Parameters

- **date_string** (*basestring*) – The date the log will start at
- **log_file** (*string, optional*) – Provides the path the log will be written to. Default "./log.txt"
- **log_level** (*{logging.DEBUG, logging.INFO, logging.WARNING, logging.ERROR, logging.CRITICAL}*) – Defines the level of the log. Default logging.INFO
- **numpy_error_level** (*{'log', 'raise'}*) – Defines the response to numpy errors. Default `log`. See numpy.seterr

Returns **close_loggers** – Closes the logging systems that have been set up

Return type function

See also:

logging() The Python standard logging library

numpy.seterr() The function npErrResp is passed to for defining the response to numpy errors

outputting.**file_name_generator**(*output_folder=None*)

Keeps track of filenames that have been used and generates the next unused one

Parameters **output_folder** (*string, optional*) – The folder into which the new file will be placed. Default is the current working directory

Returns **new_file_name** – Creates a new file with the name <handle> and the extension <extension>. It takes two string parameters: (`handle`, `extension`) and returns one `fileName` string

Return type function

Examples

```
>>> file_name_gen = file_name_generator("./")
>>> file_name_gen("a", "b")
'./a.b'
>>> file_name_gen("a", "b")
'./a_1.b'
>>> file_name_gen("", "")
'./'
>>> file_name_gen = file_name_generator()
>>> fileName = file_name_gen("", "")
>>> fileName == os.getcwd()
False
```

outputting.**flatDictKeySet**(*store*, *selectKeys=None*)

Generates a dictionary of keys and identifiers for the new dictionary, including only the keys in the keys list. Any keys with lists will be split into a set of keys, one for each element in the original key.

These are named <key><location>

Parameters

- **store** (*list of dicts*) – The dictionaries would be expected to have many of the same keys. Any dictionary keys containing lists in the input have been split into multiple numbered keys
- **selectKeys** (*list of strings, optional*) – The keys whose data will be included in the return dictionary. Default `None`, which results in all keys being returned

Returns **keySet** – The dictionary of keys to be extracted

Return type OrderedDict with values of OrderedDict, list or None

See also:

`reframeListDicts()`, *newFlatDict()*

outputting.**folder_path_cleaning**(*folder*)

Modifies string file names from Windows format to Unix format if necessary and makes sure there is a `/` at the end.

Parameters **folder** (*string*) – The folder path

Returns **folder_path** – The folder path

Return type basestring

outputting.**folder_setup**(*label*, *date_string*, *pickle_data=False*, *base_path=None*)

Identifies and creates the folder the data will be stored in

Folder will be created as "./Outputs/<sim_label>_<date>/". If that had previously been created then it is created as "./Outputs/<sim_label>_<date>_no_<#>/", where "<#>" is the first available integer.

A subfolder is also created with the name `Pickle` if pickle is true.

Parameters

- **label** (*basestring*) – The label for the simulation
- **date_string** (*basestring*) – The date identifier
- **pickle_data** (*bool, optional*) – If true the data for each model, task and participant is recorded. Default is `False`
- **base_path** (*basestring, optional*) – The path into which the new folder will be placed. Default is current working directory

Returns **folder_name** – The folder path that has just been created

Return type string

See also:

newFile() Creates a new file

saving() Creates the log system

outputting.**listKeyGen**(*data*, *maxListLen=None*, *returnList=False*, *abridge=False*)

Identifies the columns necessary to convert a list into a table

Parameters

- **data** (*numpy.ndarray or list*) – The list to be broken down
- **maxListLen** (*int or float with no decimal places or None, optional*) – The length of the longest expected list. Only useful if returnList is `True`. Default `None`
- **returnList** (*bool, optional*) – Defines if the lists will be broken into 1D lists or values. Default `False`, lists will be broken into values
- **abridge** (*bool, optional*) – Defines if the final dataset will be a summary or the whole lot. If it is a summary, lists of more than 10 elements are removed. Default `False`, not abridged

Returns

- **returnList** (*None or list of tuples of ints or ints*) – The list of co-ordinates for the elements to be extracted from the data. If None the list is used as-is.
- **maxListLen** (*int or float with no decimal places or None, optional*) – If returnList is `True` this should be the length of the longest list. If returnList is `False` this should return its original value

Examples

```
>>> listKeyGen([[1, 2, 3, 4, 5, 6], [4, 5, 6, 7, 8, 9]], maxListLen=None,␣
↪returnList=False, abridge=False)
(array([[0, 0], [1, 0], [0, 1], [1, 1], [0, 2], [1, 2], [0, 3], [1, 3], [0, 4],
↪ [1, 4], [0, 5], [1, 5]]), 1)
>>> listKeyGen([[1, 2, 3, 4, 5, 6], [4, 5, 6, 7, 8, 9]], maxListLen=None,␣
↪returnList=False, abridge=True)
(None, None)
>>> listKeyGen([[1, 2, 3, 4, 5, 6], [4, 5, 6, 7, 8, 9]], maxListLen=None,␣
↪returnList=True, abridge=True)
(array([[0],
       [1]]), 6L)
```

outputting.**listSelection**(*data*, *loc*)

Allows numpy array-like referencing of lists

Parameters

- **data** (*list*) – The data to be referenced
- **loc** (*tuple of integers*) – The location to be referenced

Returns **selection** – The referenced subset

Return type list

Examples

```
>>> listSelection([1, 2, 3], (0,))
1
>>> listSelection([[1, 2, 3], [4, 5, 6]], (0,))
[1, 2, 3]
>>> listSelection([[1, 2, 3], [4, 5, 6]], (0, 2))
3
```

outputting.**newFlatDict**(*store*, *selectKeys=None*, *labelPrefix=u''*)

Takes a list of dictionaries and returns a dictionary of 1D lists.

If a dictionary did not have that key or list element, then 'None' is put in its place

Parameters

- **store** (*list of dicts*) – The dictionaries would be expected to have many of the same keys. Any dictionary keys containing lists in the input have been split into multiple numbered keys
- **selectKeys** (*list of strings, optional*) – The keys whose data will be included in the return dictionary. Default `None`, which results in all keys being returned
- **labelPrefix** (*string*) – An identifier to be added to the beginning of each key string.

Returns **newStore** – The new dictionary with the keys from the keySet and the values as 1D lists with 'None' if the keys, value pair was not found in the store.

Return type dict

Examples

```
>>> store = [{'list': [1, 2, 3, 4, 5, 6]}]
>>> newFlatDict(store)
OrderedDict([('list_[0]', [1]), ('list_[1]', [2]), ('list_[2]', [3]), ('list_
↪[3]', [4]), ('list_[4]', [5]), ('list_[5]', [6])])
>>> store = [{'string': 'string'}]
>>> newFlatDict(store)
OrderedDict([(u'string', ["u'string'"])])
>>> store = [{'dict': {1: {3: "a"}, 2: "b"}}]
>>> newFlatDict(store)
OrderedDict([(u'dict_1_3', ["u'a'"]), (u'dict_2', ["u'b'"])])
```

outputting.**newListDict**(*store*, *labelPrefix=u''*, *maxListLen=0*)

Takes a dictionary of numbers, strings, lists and arrays and returns a dictionary of 1D arrays.

If there is a single value, then a list is created with that value repeated

Parameters

- **store** (*dict*) – A dictionary of numbers, strings, lists, dictionaries and arrays
- **labelPrefix** (*string*) – An identifier to be added to the beginning of each key string. Default empty string

Returns **newStore** – The new dictionary with the keys from the keySet and the values as 1D lists.

Return type dict

Examples

```
>>> store = {'list': [1, 2, 3, 4, 5, 6]}
>>> newListDict(store)
OrderedDict([('list', [1, 2, 3, 4, 5, 6])])
>>> store = {'string': 'string'}
>>> newListDict(store)
OrderedDict([('string', ['string'])])
>>> store = {'dict': {1: {3: "a"}, 2: "b"}}
>>> newListDict(store)
OrderedDict([(u'dict_1_3', ['a']), (u'dict_2', ['b'])])
```

outputting.**pad**(*values*, *maxListLen*)
Pads a list with None

Parameters

- **values** (*list*) – The list to be extended
- **maxListLen** (*int*) – The number of elements the list needs to have

outputting.**pickleLog**(*results*, *file_name_gen*, *label=u''*)
Stores the data in the appropriate pickle file in a Pickle subfolder of the outputting folder

Parameters

- **results** (*dict*) – The data to be stored
- **file_name_gen** (*function*) – Creates a new file with the name <handle> and the extension <extension>. It takes two string parameters: (`handle`, `extension`) and returns one `fileName` string
- **label** (*string, optional*) – A label for the results file

outputting.**pickle_write**(*data*, *handle*, *file_name_gen*)
Writes the data to a pickle file

Parameters

- **data** (*object*) – Data to be written to the file
- **handle** (*string*) – The name of the file
- **file_name_gen** (*function*) – Creates a new file with the name <handle> and the extension <extension>. It takes two string parameters: (`handle`, `extension`) and returns one `fileName` string

6.10 utils module

6.10.1 utils Module

Author Dominic Hunt

6.10.1.1 Functions

argProcess(**kwargs)	
callableDetails(item)	Takes a callable item and extracts the details.
callableDetailsString(item)	Takes a callable item and returns a string detailing the function.
date()	Provides a string of today's date
discountAverage(data, discount)	An accumulating mean
errorResp()	Takes an error that has been caught and returns the details as a string
find_class(class_name, class_folder, ... [, ...])	Finds and imports a class from a given folder.
find_function(function_name, function_folder)	Finds and imports a function from a given folder.
flatten(data)	Yields the elements in order from any N dimensional iterable
getClassArgs(inspected_class[, arg_ignore])	Finds the arguments that could be passed into the specified class
getClassAttributes(inspected_class[, ignore])	Finds the public attributes of the specified class
getFuncArgs(inspected_function)	Finds the arguments that could be passed into the specified function
kendalw(data[, ranked])	Calculates Kendall's W for a n*m array with n items and m 'judges'.
kendalwt(data[, ranked])	Calculates Kendall's W for a n*m array with n items and m 'judges'.
kendalwts(data[, ranked])	Calculates Kendall's W for a n*m array with n items and m 'judges'.
kldivergence(m0, m1, c0, c1)	Calculates the Kullback–Leibler divergence between two distributions using the means and covariances
listMerge(*args)	For merging lists with objects that are not solely numbers
listMergeGen(*args)	Fast merging of lists of numbers
listMergeNP(*args)	Fast merging of lists of numbers
list_all_equal(data)	Checks if all of the elements of a list are the same.
mergeDatasetRepr(data[, dataLabel])	Take a list of dictionaries and turn it into a dictionary of lists of strings
mergeDatasets(data[, extend])	Take a list of dictionaries and turn it into a dictionary of lists of objects
mergeDicts(*args)	Merges any number of dictionaries with different keys into a new dict
mergeTwoDicts(x, y)	Given two dicts, merge them into a new dict as a shallow copy
movingaverage(data, windowSize[, edgeCorrection])	Average over an array
runningAverage(data)	An accumulating mean
runningMean(oldMean, newValue, numValues)	A running mean
runningSTD(oldSTD, oldMean, newMean, newValue)	**param oldSTD** The old running average standard deviation
unique(seq[, idfun])	Finds the unique items in a list and returns them in order found.
varyingParams(intObjects, params)	Takes a list of models or tasks and returns a dictionary with only the parameters which vary and their values

argProcess

utils.**argProcess**(***kwargs*)

callableDetails

utils.**callableDetails**(*item*)

Takes a callable item and extracts the details.

Currently only extracts things stored in `item.Name` and `item.Params`

Parameters **item** (*callable item*) –

Returns **details** – Contains the properties of the

Return type tuple pair with string and dictionary of strings

Examples

```
>>> from utils import callableDetails
>>> def foo(): print("foo")
>>> foo.Name = "boo"
>>> callableDetails(foo)
('boo', None)
>>> foo.Params = {1: 2, 2: 3}
>>> callableDetails(foo)
('boo', {'1': '2', '2': '3'})
```

callableDetailsString

utils.**callableDetailsString**(*item*)

Takes a callable item and returns a string detailing the function.

Currently only extracts things stored in `item.Name` and `item.Params`

Parameters **item** (*callable item*) –

Returns **description** – Contains the properties and name of the callable

Return type string

Examples

```
>>> from utils import callableDetailsString
>>> def foo(): print("foo")
>>> foo.Name = "boo"
>>> callableDetailsString(foo)
'boo'
>>> foo.Params = {1: 2, 2: 3}
>>> callableDetailsString(foo)
'boo with 1 : 2, 2 : 3'
```

date

utils.**date**()

Provides a string of today's date

Returns **date** – The string is of the form [year]-[month]-[day]

Return type string

discountAverage

utils.**discountAverage**(*data*, *discount*)
An accumulating mean

Parameters

- **data** (*list or 1-D array of floats*) – The set of values to be averaged
- **discount** (*float*) – The value by which each previous value is discounted

Returns **results** – The values from the moving average

Return type ndArray of length data

Examples

```
>>> discountAverage([1, 2, 3, 4], 1)
array([1. , 1.5, 2. , 2.5])
>>> discountAverage([1, 2, 3, 4], 0.25)
array([1.        , 1.8       , 2.71428571, 3.68235294])
```

errorResp

utils.**errorResp**()
Takes an error that has been caught and returns the details as a string

Returns **description** – Contains the description of the error

Return type string

find_class

utils.**find_class**(*class_name*, *class_folder*, *inherited_class*, *excluded_files=None*)
Finds and imports a class from a given folder. Does not look in subfolders

Parameters

- **class_name** (*string*) – The name of the class to be used
- **class_folder** (*basestring*) – The path where the class is likely to be found
- **inherited_class** (*class*) – The class that the searched for class inherits from
- **excluded_files** (*list, optional*) – A list of modules to be excluded from the search. Can be described using portions of file names.

Returns **sought_class** – The uninstansiated class sought

Return type inherited_class

find_function

utils.**find_function**(*function_name*, *function_folder*, *excluded_files=None*)
Finds and imports a function from a given folder. Does not look in subfolders

Parameters

- **function_name** (*string*) – The name of the function to be used
- **function_folder** (*basestring*) – The path where the function is likely to be found

- **excluded_files** (*list, optional*) – A list of modules to be excluded from the search. Can be described using portions of file names.

Returns **sought_class** – The uninstansiated class sought

Return type inherited_class

flatten

utils.**flatten**(*data*)

Yields the elements in order from any N dimensional iterable

Parameters **data** (*iterable*) –

Yields **ID** ((*string,list*)) – A pair containing the value at each location and the co-ordinates used to access them.

Examples

```
>>> a = [[1, 2, 3],[4, 5, 6]]
>>> for i, loc in flatten(a): print(i,loc)
1 [0, 0]
2 [0, 1]
3 [0, 2]
4 [1, 0]
5 [1, 1]
6 [1, 2]
```

getClassArgs

utils.**getClassArgs**(*inspected_class, arg_ignore=[u'self']*)

Finds the arguments that could be passed into the specified class

getClassAttributes

utils.**getClassAttributes**(*inspected_class, ignore=[u'self']*)

Finds the public attributes of the specified class

getFuncArgs

utils.**getFuncArgs**(*inspected_function*)

Finds the arguments that could be passed into the specified function

Parameters **inspected_function** –

Returns

kendalw

utils.**kendalw**(*data*, *ranked=False*)

Calculates Kendall's W for a n*m array with n items and m 'judges'.

Parameters

- **data** (*list or np.ndarray*) – The data in the form of an n*m array with n items and m 'judges'

- **ranked** (*bool, optional*) – If the data has already been ranked or not. Default `False`

Returns **w** – The Kendall's W

Return type float

Notes

Based on Legendre, P. (2010). Coefficient of Concordance. In Encyclopedia of Research Design (pp. 164–169). 2455 Teller Road, Thousand Oaks California 91320 United States: SAGE Publications, Inc. http://doi.org/10.4135/9781412961288.n55

Examples

```
>>> data = np.array([[2., 0., 5., 1.], [3., 3., 3., 4.], [1., 5., 3., 5.], [1.,
↪ 1., 4., 2.], [2., 4., 5., 1.], [1., 0., 0., 2.]])
```

```
>>> kendalw(data)
0.22857142857142856
```

```
>>> data = np.array([[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3], [4, 4, 4, 4],␣
↪[5, 5, 5, 5], [6, 6, 6, 6]])
```

```
>>> kendalw(data)
1.0
```

kendalwt

utils.**kendalwt**(*data*, *ranked=False*)

Calculates Kendall's W for a n*m array with n items and m 'judges'. Corrects for ties.

Parameters

- **data** (*list or np.ndarray*) – The data in the form of an n*m array with n items and m 'judges'
- **ranked** (*bool, optional*) – If the data has already been ranked or not. Default `False`

Returns **w** – The Kendall's W

Return type float

Notes

Based on Legendre, P. (2010). Coefficient of Concordance. In Encyclopedia of Research Design (pp. 164–169). 2455 Teller Road, Thousand Oaks California 91320 United States: SAGE Publications, Inc. http://doi.org/10.4135/9781412961288.n55

Examples

```
>>> data = np.array([[2., 0., 5., 1.], [3., 3., 3., 4.], [1., 5., 3., 5.], [1.,
↪ 1., 4., 2.], [2., 4., 5., 1.], [1., 0., 0., 2.]])
>>> kendalwt(data)
0.24615384615384617
```

```
>>> data = np.array([[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3], [4, 4, 4, 4],␣
↪[5, 5, 5, 5], [6, 6, 6, 6]])
>>> kendalwt(data)
1.0
```

kendalwts

utils.**kendalwts**(*data*, *ranked=False*)

Calculates Kendall's W for a n*m array with n items and m 'judges'. Corrects for ties.

Parameters

- **data** (*list or np.ndarray*) – The data in the form of an n*m array with n items and m 'judges'
- **ranked** (*bool, optional*) – If the data has already been ranked or not. Default False

Returns **w** – The Kendall's W

Return type float

Notes

Based on Legendre, P. (2010). Coefficient of Concordance. In Encyclopedia of Research Design (pp. 164–169). 2455 Teller Road, Thousand Oaks California 91320 United States: SAGE Publications, Inc. http://doi.org/10.4135/9781412961288.n55

Examples

```
>>> data = np.array([[2., 0., 5., 1.], [3., 3., 3., 4.], [1., 5., 3., 5.], [1.,
↪ 1., 4., 2.], [2., 4., 5., 1.], [1., 0., 0., 2.]])
>>> kendalwts(data)
0.24615384615384617
```

```
>>> data = np.array([[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3], [4, 4, 4, 4],␣
↪[5, 5, 5, 5], [6, 6, 6, 6]])
>>> kendalwts(data)
1.0
```

kldivergence

utils.**kldivergence**(*m0*, *m1*, *c0*, *c1*)

Calculates the Kullback–Leibler divergence between two distributions using the means and covariances

Parameters

- **m0** (*array of N floats*) – The means of distribution 0
- **m1** (*array of N floats*) – The means of distribution 1
- **c0** (*NxN array of floats*) – The covariance matrix for distribution 0
- **c1** (*NxN array of floats*) – The covariance matrix for distribution 1

Returns **kl** – The Kullback–Leibler divergence

Return type float

listMerge

utils.**listMerge**(**args*)

For merging lists with objects that are not solely numbers

Parameters **args** (*list of lists*) – A list of 1D lists of objects

Returns **combinations** – An np.array with len(args) columns and a row for each combination

Return type np.array

Examples

```
>>> listMerge([1, 2, 3], [5, 6, 7]).T
array([[1, 2, 3, 1, 2, 3, 1, 2, 3],
       [5, 5, 5, 6, 6, 6, 7, 7, 7]])
```

listMergeGen

utils.**listMergeGen**(**args*)

Fast merging of lists of numbers

Parameters **args** (*list of lists of numbers*) – A list of 1D lists of numbers

Yields **combination** (*numpy.array of 1 x len(args)*) – Array of all combinations

Examples

```
>>> for i in listMergeGen(0.7): print(repr(i))
array([0.7])
>>> for i in listMergeGen([0.7, 0.1]): print(repr(i))
array([0.7])
array([0.1])
>>> for i in listMergeGen([0.7, 0.1], [0.6]): print(repr(i))
array([0.7,  0.6])
array([0.1,  0.6])
>>> for i in listMergeGen([0.7, 0.1], []): print(repr(i))
```

```
>>> for i in listMergeGen([0.7, 0.1], 0.6): print(repr(i))
array([0.7,  0.6])
array([0.1,  0.6])
```

listMergeNP

utils.**listMergeNP**(**args*)

Fast merging of lists of numbers

Parameters **args** (*list of lists of numbers*) – A list of 1D lists of numbers

Returns **combinations** – An np.array with len(args) columns and a row for each combination

Return type np.array

Examples

```
>>> listMergeNP([1, 2, 3], [5, 6, 7]).T
array([[1, 2, 3, 1, 2, 3, 1, 2, 3], [5, 5, 5, 6, 6, 6, 7, 7, 7]])
```

list_all_equal

utils.**list_all_equal**(*data*)

Checks if all of the elements of a list are the same.

Parameters **data** (*list of 1D*) – The list of elements to compare

Returns **equivalence** – True if the elements are all the same

Return type bool

Notes

Based on https://stackoverflow.com/questions/3844801

mergeDatasetRepr

utils.**mergeDatasetRepr**(*data*, *dataLabel=u''*)

Take a list of dictionaries and turn it into a dictionary of lists of strings

Parameters

- **data** (*list of dicts containing strings, lists or numbers*) –
- **dataLabel** (*string, optional*) – This string will be appended to the front of each key in the new dataset Default blank

Returns **newStore** – For each key a list will be formed of the string representations of each of the former key values.

Return type dictionary of lists of strings

mergeDatasets

utils.**mergeDatasets**(*data*, *extend=False*)

Take a list of dictionaries and turn it into a dictionary of lists of objects

Parameters

- **data** (*list of dicts containing strings, lists or numbers*) –
- **extend** (*bool, optional*) – If lists should be extended rather than appended. Default False

Returns **newStore** – For each key a list will be formed of the former key values. If a data set did not contain a key a value of None will be entered for it.

Return type dictionary of lists of objects

Examples

```
>>> data = [{'a':[1, 2, 3], 'b':[7, 8, 9]}, {'b':[4, 5, 6], 'c':'string', 'd
↪':5}]
>>> mergeDatasets(data)
{'a': [[1, 2, 3], None], 'c': [None, 'string'], 'b': [[7, 8, 9], [4, 5, 6]], 'd
↪': [None, 5]}
>>> mergeDatasets(data, extend=True)
{'a': [1, 2, 3, None], 'c': [None, 'string'], 'b': [7, 8, 9, 4, 5, 6], 'd':␣
↪[None, 5]}
 >>> data = [{'b': np.array([[7, 8, 9], [1, 2, 3]])}, {'b': np.array([[4, 5,␣
↪6], [2, 3, 4]])}]
```

(continues on next page)

(continued from previous page)

```
>>> mergeDatasets(data, extend = True)
{'b': [array([7, 8, 9]), array([1, 2, 3]), array([4, 5, 6]), array([2, 3,␣
↪4])]}
>>> mergeDatasets(data)
{'b': [array([[7, 8, 9], [1, 2, 3]]), array([[4, 5, 6], [2, 3, 4]])]}
```

mergeDicts

utils.**mergeDicts**(**args*)

Merges any number of dictionaries with different keys into a new dict

Precedence goes to key value pairs in latter dicts

Parameters **args** (*list of dictionaries*) –

Returns **mergedDict**

Return type dictionary

mergeTwoDicts

utils.**mergeTwoDicts**(*x*, *y*)

Given two dicts, merge them into a new dict as a shallow copy

Assumes different keys in both dictionaries

Parameters

- **x** (*dictionary*) –
- **y** (*dictionary*) –

Returns **mergedDict**

Return type dictionary

movingaverage

utils.**movingaverage**(*data*, *windowSize*, *edgeCorrection=False*)

Average over an array

Parameters

- **data** (*list of floats*) – The data to average
- **windowSize** (*int*) – The size of the window
- **edgeCorrection** (*bool*) – If `True` the edges are repaired so that there is no unusual dropoff

Returns **convolution**

Return type array

Examples

```
>>> movingaverage([1, 1, 1, 1, 1], 3)
array([0.66666667, 1.        , 1.        , 1.        , 0.66666667])
>>> movingaverage([1, 1, 1, 1, 1, 1, 1, 1], 4)
array([0.5 , 0.75, 1.  , 1.  , 1.  , 1.  , 1.  , 0.75])
```

(continues on next page)

(continued from previous page)

```
>>> movingaverage([1, 1, 1, 1, 1], 3, edgeCorrection=True)
array([1., 1., 1., 1., 1.])
```

runningAverage

utils.**runningAverage**(*data*)

An accumulating mean

Parameters **data** (*list or 1-D array of floats*) – The set of values to be averaged

Returns **results** – The values from the moving average

Return type ndArray of length data

Examples

```
>>> runningAverage([1,2,3,4])
array([1. ,  1.5,  2. ,  2.5])
```

runningMean

utils.**runningMean**(*oldMean*, *newValue*, *numValues*)

A running mean

Parameters

- **oldMean** (*float*) – The old running average mean
- **newValue** (*float*) – The new value to be added to the mean
- **numValues** (*int*) – The number of values in the new running mean once this value is included

Returns **newMean** – The new running average mean

Return type float

Notes

Based on Donald Knuth's Art of Computer Programming, Vol 2, page 232, 3rd edition and taken from https://www.johndcook.com/blog/standard_deviation/

Examples

```
>>> runningMean(1, 2, 2)
1.5
>>> runningMean(1.5, 3, 3)
2.0
```

runningSTD

utils.**runningSTD**(*oldSTD*, *oldMean*, *newMean*, *newValue*)

Parameters

- **oldSTD** (*float*) – The old running average standard deviation

- **oldMean** (*float*) – The old running average mean
- **newMean** (*float*) – The new running average mean
- **newValue** (*float*) – The new value to be added to the mean

Returns **newSTD** – The new running average standard deviation

Return type float

Notes

Based on Donald Knuth's Art of Computer Programming, Vol 2, page 232, 3rd edition (which is based on B. P. Welford (2012) Note on a Method for Calculating Corrected Sums of Squares and Products, Technometrics, 4:3, 419-420, DOI: 10.1080/00401706.1962.10490022 This version is taken from https://www.johndcook.com/blog/standard_deviation/

Examples

```
>>> runningSTD(0, 1, 1.5, 2)
0.5
```

```
>>> runningSTD(0.5, 1.5, 2.0, 3)
2.0
```

unique

utils.**unique**(*seq*, *idfun=None*)

Finds the unique items in a list and returns them in order found.

Inspired by discussion on `http://www.peterbe.com/plog/uniqifiers-benchmark` Notably f10 Andrew Dalke and f8 by Dave Kirby

Parameters

- **seq** (*an iterable object*) – The sequence from which the unique list will be compiled
- **idfun** (*function, optional*) – A hashing function for transforming the items into the form that is to be compared. Default is the `None`

Returns **result** – The list of unique items

Return type list

Examples

```
>>> a=list('ABeeE')
>>> unique(a)
['A', 'B', 'e', 'E']
```

```
>>> unique(a, lambda x: x.lower())
['A', 'B', 'e']
```

Note: Unless order is needed it is best to use list(set(seq))

varyingParams

utils.**varyingParams**(*intObjects*, *params*)
Takes a list of models or tasks and returns a dictionary with only the parameters which vary and their values

6.10.1.2 Classes

ClassNameError
FunctionNameError

ClassNameError

exception utils.**ClassNameError**

FunctionNameError

exception utils.**FunctionNameError**

6.10.1.3 Class Inheritance Diagram

Author Dominic Hunt

exception utils.**ClassNameError**
Bases: exceptions.Exception

exception utils.**FunctionNameError**
Bases: exceptions.Exception

utils.**argProcess**(***kwargs*)

utils.**callableDetails**(*item*)
Takes a callable item and extracts the details.

Currently only extracts things stored in item.Name and item.Params

Parameters **item** (*callable item*) –

Returns **details** – Contains the properties of the

Return type tuple pair with string and dictionary of strings

Examples

```
>>> from utils import callableDetails
>>> def foo(): print("foo")
>>> foo.Name = "boo"
>>> callableDetails(foo)
('boo', None)
>>> foo.Params = {1: 2, 2: 3}
>>> callableDetails(foo)
('boo', {'1': '2', '2': '3'})
```

utils.**callableDetailsString**(*item*)
Takes a callable item and returns a string detailing the function.

Currently only extracts things stored in item.Name and item.Params

Parameters **item** (*callable item*) –

Returns **description** – Contains the properties and name of the callable

Return type string

Examples

```
>>> from utils import callableDetailsString
>>> def foo(): print("foo")
>>> foo.Name = "boo"
>>> callableDetailsString(foo)
'boo'
>>> foo.Params = {1: 2, 2: 3}
>>> callableDetailsString(foo)
'boo with 1 : 2, 2 : 3'
```

utils.**date**()
Provides a string of today's date

Returns **date** – The string is of the form [year]-[month]-[day]

Return type string

utils.**discountAverage**(*data*, *discount*)
An accumulating mean

Parameters

- **data** (*list or 1-D array of floats*) – The set of values to be averaged
- **discount** (*float*) – The value by which each previous value is discounted

Returns **results** – The values from the moving average

Return type ndArray of length data

Examples

```
>>> discountAverage([1, 2, 3, 4], 1)
array([1. , 1.5, 2. , 2.5])
>>> discountAverage([1, 2, 3, 4], 0.25)
array([1.        , 1.8       , 2.71428571, 3.68235294])
```

utils.**errorResp**()
Takes an error that has been caught and returns the details as a string

Returns **description** – Contains the description of the error

Return type string

utils.**find_class**(*class_name*, *class_folder*, *inherited_class*, *excluded_files=None*)
Finds and imports a class from a given folder. Does not look in subfolders

Parameters

- **class_name** (*string*) – The name of the class to be used
- **class_folder** (*basestring*) – The path where the class is likely to be found
- **inherited_class** (*class*) – The class that the searched for class inherits from
- **excluded_files** (*list, optional*) – A list of modules to be excluded from the search. Can be described using portions of file names.

Returns **sought_class** – The uninstansiated class sought

Return type inherited_class

utils.**find_function**(*function_name*, *function_folder*, *excluded_files=None*)
Finds and imports a function from a given folder. Does not look in subfolders

Parameters

- **function_name** (*string*) – The name of the function to be used
- **function_folder** (*basestring*) – The path where the function is likely to be found
- **excluded_files** (*list, optional*) – A list of modules to be excluded from the search. Can be described using portions of file names.

Returns **sought_class** – The uninstansiated class sought

Return type inherited_class

utils.**flatten**(*data*)
Yields the elements in order from any N dimensional iterable

Parameters **data** (*iterable*) –

Yields **ID** (*(string,list)*) – A pair containing the value at each location and the co-ordinates used to access them.

Examples

```
>>> a = [[1, 2, 3],[4, 5, 6]]
>>> for i, loc in flatten(a): print(i,loc)
1 [0, 0]
2 [0, 1]
3 [0, 2]
4 [1, 0]
5 [1, 1]
6 [1, 2]
```

utils.**getClassArgs**(*inspected_class, arg_ignore=[u'self']*)
Finds the arguments that could be passed into the specified class

utils.**getClassAttributes**(*inspected_class, ignore=[u'self']*)
Finds the public attributes of the specified class

utils.**getFuncArgs**(*inspected_function*)
Finds the arguments that could be passed into the specified function

Parameters **inspected_function** –

Returns

utils.**kendalw**(*data*, *ranked=False*)
Calculates Kendall's W for a n*m array with n items and m 'judges'.

Parameters

- **data** (*list or np.ndarray*) – The data in the form of an n*m array with n items and m 'judges'
- **ranked** (*bool, optional*) – If the data has already been ranked or not. Default `False`

Returns **w** – The Kendall's W

Return type float

Notes

Based on Legendre, P. (2010). Coefficient of Concordance. In Encyclopedia of Research Design (pp. 164–169). 2455 Teller Road, Thousand Oaks California 91320 United States: SAGE Publications, Inc. http://doi.org/10.4135/9781412961288.n55

Examples

```
>>> data = np.array([[2., 0., 5., 1.], [3., 3., 3., 4.], [1., 5., 3., 5.], [1.,
↪ 1., 4., 2.], [2., 4., 5., 1.], [1., 0., 0., 2.]])
```

```
>>> kendalw(data)
0.22857142857142856
```

```
>>> data = np.array([[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3], [4, 4, 4, 4],␣
↪[5, 5, 5, 5], [6, 6, 6, 6]])
```

```
>>> kendalw(data)
1.0
```

utils.**kendalwt**(*data*, *ranked=False*)

Calculates Kendall's W for a n*m array with n items and m 'judges'. Corrects for ties.

Parameters

- **data** (*list or np.ndarray*) – The data in the form of an n*m array with n items and m 'judges'
- **ranked** (*bool, optional*) – If the data has already been ranked or not. Default `False`

Returns **w** – The Kendall's W

Return type float

Notes

Based on Legendre, P. (2010). Coefficient of Concordance. In Encyclopedia of Research Design (pp. 164–169). 2455 Teller Road, Thousand Oaks California 91320 United States: SAGE Publications, Inc. http://doi.org/10.4135/9781412961288.n55

Examples

```
>>> data = np.array([[2., 0., 5., 1.], [3., 3., 3., 4.], [1., 5., 3., 5.], [1.,
↪ 1., 4., 2.], [2., 4., 5., 1.], [1., 0., 0., 2.]])
>>> kendalwt(data)
0.24615384615384617
```

```
>>> data = np.array([[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3], [4, 4, 4, 4],␣
↪[5, 5, 5, 5], [6, 6, 6, 6]])
>>> kendalwt(data)
1.0
```

utils.**kendalwts**(*data*, *ranked=False*)

Calculates Kendall's W for a n*m array with n items and m 'judges'. Corrects for ties.

Parameters

- **data** (*list or np.ndarray*) – The data in the form of an n*m array with n items and m 'judges'
- **ranked** (*bool, optional*) – If the data has already been ranked or not. Default False

Returns **w** – The Kendall's W

Return type float

Notes

Based on Legendre, P. (2010). Coefficient of Concordance. In Encyclopedia of Research Design (pp. 164–169). 2455 Teller Road, Thousand Oaks California 91320 United States: SAGE Publications, Inc. http://doi.org/10.4135/9781412961288.n55

Examples

```
>>> data = np.array([[2., 0., 5., 1.], [3., 3., 3., 4.], [1., 5., 3., 5.], [1.,
↪ 1., 4., 2.], [2., 4., 5., 1.], [1., 0., 0., 2.]])
>>> kendalwts(data)
0.24615384615384617
```

```
>>> data = np.array([[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3], [4, 4, 4, 4],␣
↪[5, 5, 5, 5], [6, 6, 6, 6]])
>>> kendalwts(data)
1.0
```

utils.**kldivergence**(*m0*, *m1*, *c0*, *c1*)

Calculates the Kullback–Leibler divergence between two distributions using the means and covariances

Parameters

- **m0** (*array of N floats*) – The means of distribution 0
- **m1** (*array of N floats*) – The means of distribution 1
- **c0** (*NxN array of floats*) – The covariance matrix for distribution 0
- **c1** (*NxN array of floats*) – The covariance matrix for distribution 1

Returns **kl** – The Kullback–Leibler divergence

Return type float

utils.**listMerge**(**args*)

For merging lists with objects that are not solely numbers

Parameters **args** (*list of lists*) – A list of 1D lists of objects

Returns **combinations** – An np.array with len(args) columns and a row for each combination

Return type np.array

Examples

```
>>> listMerge([1, 2, 3], [5, 6, 7]).T
array([[1, 2, 3, 1, 2, 3, 1, 2, 3],
       [5, 5, 5, 6, 6, 6, 7, 7, 7]])
```

utils.**listMergeGen**(**args*)

Fast merging of lists of numbers

Parameters **args** (*list of lists of numbers*) – A list of 1D lists of numbers

Yields **combination** (*numpy.array of 1 x len(args)*) – Array of all combinations

Examples

```
>>> for i in listMergeGen(0.7): print(repr(i))
array([0.7])
>>> for i in listMergeGen([0.7, 0.1]): print(repr(i))
array([0.7])
array([0.1])
>>> for i in listMergeGen([0.7, 0.1], [0.6]): print(repr(i))
array([0.7,  0.6])
array([0.1,  0.6])
>>> for i in listMergeGen([0.7, 0.1], []): print(repr(i))
```

```
>>> for i in listMergeGen([0.7, 0.1], 0.6): print(repr(i))
array([0.7,  0.6])
array([0.1,  0.6])
```

utils.**listMergeNP**(**args*)
Fast merging of lists of numbers

Parameters **args** (*list of lists of numbers*) – A list of 1D lists of numbers

Returns **combinations** – An np.array with len(args) columns and a row for each combination

Return type np.array

Examples

```
>>> listMergeNP([1, 2, 3], [5, 6, 7]).T
array([[1, 2, 3, 1, 2, 3, 1, 2, 3], [5, 5, 5, 6, 6, 6, 7, 7, 7]])
```

utils.**list_all_equal**(*data*)
Checks if all of the elements of a list are the same.

Parameters **data** (*list of 1D*) – The list of elements to compare

Returns **equivalence** – True if the elements are all the same

Return type bool

Notes

Based on https://stackoverflow.com/questions/3844801

utils.**mergeDatasetRepr**(*data*, *dataLabel=u''*)
Take a list of dictionaries and turn it into a dictionary of lists of strings

Parameters

- **data** (*list of dicts containing strings, lists or numbers*) –
- **dataLabel** (*string, optional*) – This string will be appended to the front of each key in the new dataset Default blank

Returns **newStore** – For each key a list will be formed of the string representations of each of the former key values.

Return type dictionary of lists of strings

utils.**mergeDatasets**(*data*, *extend=False*)
Take a list of dictionaries and turn it into a dictionary of lists of objects

Parameters

- **data** (*list of dicts containing strings, lists or numbers*) –
- **extend** (*bool, optional*) – If lists should be extended rather than appended. Default False

Returns **newStore** – For each key a list will be formed of the former key values. If a data set did not contain a key a value of None will be entered for it.

Return type dictionary of lists of objects

Examples

```
>>> data = [{'a':[1, 2, 3], 'b':[7, 8, 9]}, {'b':[4, 5, 6], 'c':'string', 'd
↪':5}]
>>> mergeDatasets(data)
{'a': [[1, 2, 3], None], 'c': [None, 'string'], 'b': [[7, 8, 9], [4, 5, 6]], 'd
↪': [None, 5]}
>>> mergeDatasets(data, extend=True)
{'a': [1, 2, 3, None], 'c': [None, 'string'], 'b': [7, 8, 9, 4, 5, 6], 'd':␣
↪[None, 5]}
 >>> data = [{'b': np.array([[7, 8, 9], [1, 2, 3]])}, {'b': np.array([[4, 5,␣
↪6], [2, 3, 4]])}]
 >>> mergeDatasets(data, extend = True)
 {'b': [array([7, 8, 9]), array([1, 2, 3]), array([4, 5, 6]), array([2, 3,␣
↪4])]}
 >>> mergeDatasets(data)
 {'b': [array([[7, 8, 9], [1, 2, 3]]), array([[4, 5, 6], [2, 3, 4]])]}
```

utils.**mergeDicts**(**args*)

Merges any number of dictionaries with different keys into a new dict

Precedence goes to key value pairs in latter dicts

Parameters **args** (*list of dictionaries*) –

Returns **mergedDict**

Return type dictionary

utils.**mergeTwoDicts**(*x*, *y*)

Given two dicts, merge them into a new dict as a shallow copy

Assumes different keys in both dictionaries

Parameters

- **x** (*dictionary*) –
- **y** (*dictionary*) –

Returns **mergedDict**

Return type dictionary

utils.**movingaverage**(*data*, *windowSize*, *edgeCorrection=False*)

Average over an array

Parameters

- **data** (*list of floats*) – The data to average
- **windowSize** (*int*) – The size of the window
- **edgeCorrection** (*bool*) – If `True` the edges are repaired so that there is no unusual dropoff

Returns **convolution**

Return type array

Examples

```
>>> movingaverage([1, 1, 1, 1, 1], 3)
array([0.66666667, 1.        , 1.        , 1.        , 0.66666667])
>>> movingaverage([1, 1, 1, 1, 1, 1, 1, 1], 4)
array([0.5 , 0.75, 1.  , 1.  , 1.  , 1.  , 1.  , 0.75])
>>> movingaverage([1, 1, 1, 1, 1], 3, edgeCorrection=True)
array([1., 1., 1., 1., 1.])
```

utils.**runningAverage**(*data*)

An accumulating mean

Parameters **data** (*list or 1-D array of floats*) – The set of values to be averaged

Returns **results** – The values from the moving average

Return type ndArray of length data

Examples

```
>>> runningAverage([1,2,3,4])
array([1. ,  1.5,  2. ,  2.5])
```

utils.**runningMean**(*oldMean*, *newValue*, *numValues*)

A running mean

Parameters

- **oldMean** (*float*) – The old running average mean
- **newValue** (*float*) – The new value to be added to the mean
- **numValues** (*int*) – The number of values in the new running mean once this value is included

Returns **newMean** – The new running average mean

Return type float

Notes

Based on Donald Knuth's Art of Computer Programming, Vol 2, page 232, 3rd edition and taken from https://www.johndcook.com/blog/standard_deviation/

Examples

```
>>> runningMean(1, 2, 2)
1.5
>>> runningMean(1.5, 3, 3)
2.0
```

utils.**runningSTD**(*oldSTD*, *oldMean*, *newMean*, *newValue*)

Parameters

- **oldSTD** (*float*) – The old running average standard deviation
- **oldMean** (*float*) – The old running average mean
- **newMean** (*float*) – The new running average mean

- **newValue** (*float*) – The new value to be added to the mean

Returns **newSTD** – The new running average standard deviation

Return type float

Notes

Based on Donald Knuth's Art of Computer Programming, Vol 2, page 232, 3rd edition (which is based on B. P. Welford (2012) Note on a Method for Calculating Corrected Sums of Squares and Products, Technometrics, 4:3, 419-420, DOI: 10.1080/00401706.1962.10490022 This version is taken from https://www.johndcook.com/blog/standard_deviation/

Examples

```
>>> runningSTD(0, 1, 1.5, 2)
0.5
```

```
>>> runningSTD(0.5, 1.5, 2.0, 3)
2.0
```

utils.**unique**(*seq*, *idfun=None*)

Finds the unique items in a list and returns them in order found.

Inspired by discussion on `http://www.peterbe.com/plog/uniqifiers-benchmark` Notably f10 Andrew Dalke and f8 by Dave Kirby

Parameters

- **seq** (*an iterable object*) – The sequence from which the unique list will be compiled
- **idfun** (*function, optional*) – A hashing function for transforming the items into the form that is to be compared. Default is the `None`

Returns **result** – The list of unique items

Return type list

Examples

```
>>> a=list('ABeeE')
>>> unique(a)
['A', 'B', 'e', 'E']
```

```
>>> unique(a, lambda x: x.lower())
['A', 'B', 'e']
```

Note: Unless order is needed it is best to use list(set(seq))

utils.**varyingParams**(*intObjects*, *params*)

Takes a list of models or tasks and returns a dictionary with only the parameters which vary and their values

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index

A

B

C

D

E

F

G

I

K

L

M

N

O

R

S

T

U

V

W

X