BayesCMD Documentation

Release

Joshua Russell-Buckland

Oct 06, 2017

CONTENTS:

BayesCMD is a package intended to expand the capabilities of the Brain/Circulation Modelling (BCMD) framework. It introduces the ability to obtain posterior distributions for model parameters by using Approximate Bayesian Computation (ABC).

CHAPTER

ONE

BCMDMODEL

1.1 Running the BCMD Model

class bayescmd.bcmdModel.**ModelBCMD**(*model_name*, *inputs=None*, *params=None*, *times=None*, *outputs=None*, *burn_in=999*, *create_input=True*, *input_file=None*, *suppress=False*, *workdir=None*, *deleteWorkdir=False*, *timeout=30*, *basedir='../bayescmd'*, *debug=False*, *testing=False*)

BCMD model class. this can be used to create inputs, run simulations etc.

create_default_input()

Method to create input file and write to string buffer for acces direct from memory.

create_initialised_input()

Method to create input file and write to string buffer for access direct from memory.

output_parse()

Function to parse the output files into a dictionary.

write_default_input()

Function to write a default input to file.

write_initialised_input()

Function to write a default input to file.

1.2 Input Creation

Input files are required by the BCMD model.

1.2.1 input_creation

Create input files for use with a BCMD model.

Input files are needed in order to set model parameters and provide driving inputs.

class bayescmd.bcmdModel.**InputCreator**(*times*, *inputs*, *outputs=None*, *params=None*, *filename=None*)

Create an input file by passing relevant information to the class.

This input file is then used to create an input file that can either be written to file or kept in buffer and passed directly to the model.

Parameters

- **times** (list of float or int) – List of times at which measurement data has been collected and needs to be simulated.
- **inputs** (*dict*) – Dictionary of model inputs and their values. Has form {'names' : list of str, 'values' : list of list of float} where *names* should be a list of each model input name, matching up to the model inputs and *values* would be a list of lsits, where each sublist is the input values for that time point. With this in mind, the length of *inputs['values']'* should equal length of *times*.
- **filename** (str, optional) – Name of the input file to be written to if writing to file is required. Default is None.
- **params** (dict of str: float, optional) – Dictionary of {'parameter': param_value}
- **outputs** (list of str, optional) – List of model outputs to return.

times
list of float or int – List of times at which measurement data has been collected and needs to be simulated.

inputs
dict – Dictionary of model inputs and their values. Has form {'names' : list of str, 'values' : list of list of float} where *names* should be a list of each model input name, matching up to the model inputs and *values* would be a list of lsits, where each sublist is the input values for that time point. With this in mind, the length of *inputs['values']'* should equal length of *times*.

f_out
StringIO() – String buffer object to which the input file will be written.

filename
str – Name of the input file to be written to if writing to file is required. Default is None.

params
dict of str: float. – Dictionary of {'parameter': param_value}

outputs
list of str – List of model outputs to return.

default_creation()
Create a default input file from given arguments.

Assumes parameters remain unchanged from default values.

Returns Returns the input file as a String.IO() buffer object.

Return type String.IO()

initialised_creation(*burn_in*)
Create an input file from given arguments.

Creates an input file thatcan have non-default parameter values and outputs, as well as a burn in period. Assumes parameters remain constant for the full duration of the simulation.

Parameters **burn_in** (float or int) – Length of burn in period at start of the simulation.

Returns Returns the input file as a String.IO() buffer object.

Return type String.IO()

input_file_write()
Write input file from buffer to file.

CHAPTER

TWO

ABC

The *abc* subpackage is used to handle the Approximate Bayesian Computation (ABC) specific components of BayesCMD. This includes running the model multiple times in a batch process, calculating distances between datasets and generating priors for parameters.

2.1 Distances

Use to generate distance measures between simulated and real time series.

bayescmd.abc.distances.**DISTANCES**
dict – Dictionary contianing the distance aliases, mapping to the functions.

exception bayescmd.abc.distances.**Error**
Base class for exceptions in this module.

exception bayescmd.abc.distances.**ZeroArrayError**
Exception raised for errors in the zero array.

bayescmd.abc.distances.**check_for_key**(*dictionary*, *target*)
Check that a dictionary contains a key, and if so, return its data.

Parameters

- **dictionary** (*dict*) – Dictionary to check for *target* key.
- **target** (*str*) – String containing the target variable that is expected to be found in *dictionary*

Returns **data** – List of data found in *dictionary*. This is likely to be the time series data collected experimentally or generated by the model.

Return type list

bayescmd.abc.distances.**euclidean_dist**(*data1*, *data2*)
Get the euclidean distance between two numpy arrays.

Parameters

- **data1** (*np.ndarray*) – First data array.

  The shape should match that of data2 and the number of rows should match the number of model outputs i.e. 2 model outputs will be two rows.

- **data2** (*np.ndarray*) – Second data array.

  The shape should match that of data1 and the number of rows should match the number of model outputs i.e. 2 model outputs will be two rows.

Returns **d** – Euclidean distance measure

Return type float

bayescmd.abc.distances.**get_distance**(*actual_data*, *sim_data*, *targets*, *zero_flag*, *distance='euclidean'*, *normalise=False*)

Obtain distance between two sets of data.

Get a distance as defined by *distance* between two sets of data as well as between each signal in the data.

Parameters

- **actual_data** (*dict*) – Dictionary of actual data, as generated by `bayescmd.abc.data_import.import_actual_data()`
- **sim_data** (*dict*) – Dictionary of simulated data, as created by *bayescmd.bcmdModel.ModelBCMD.output_parse()*
- **targets** (list of `str`) – List of model targets, which should all be strings.
- **zero_flag** (*dict*) – Dictionary of form target(`str`): bool, where bool indicates whether to zero that target.

  Note: zero_flag keys should match targets list.
- **distance** (*str, optional*) – Name of distance measure to use. One of ['euclidean', 'manhattan', 'MAE', 'MSE'], where default is 'euclidean'.
- **normalise** (*bool, optional*) – Boolean flag to indicate whether the signals need normalising, default is False. Current normalisation is done using z-score but that is likely to change with time.

Returns

distances –

Dictionary of form: {'TOTAL': summed distance of all signals, 'target1: distance of 1st target', ... 'targetN': distance of Nth target }

Return type dict

bayescmd.abc.distances.**manhattan_dist**(*data1*, *data2*)

Get the Manhattan distance between two numpy arrays.

Parameters

- **data1** (*np.ndarray*) – First data array.

  The shape should match that of data2 and the number of rows should match the number of model outputs i.e. 2 model outputs will be two rows.
- **data2** (*np.ndarray*) – Second data array.

  The shape should match that of data1 and the number of rows should match the number of model outputs i.e. 2 model outputs will be two rows.

Returns **d** – Manhattan distance measure

Return type float

bayescmd.abc.distances.**mean_absolute_error_dist**(*data1*, *data2*)

Get the normalised manhattan distance between two numpy arrays.

Parameters

- **data1** (*np.ndarray*) – First data array.

  The shape should match that of data2 and the number of rows should match the number of model outputs i.e. 2 model outputs will be two rows.

- **data2** (*np.ndarray*) – Second data array.

  The shape should match that of data1 and the number of rows should match the number of model outputs i.e. 2 model outputs will be two rows.

Returns **d** – Normalised Manhattan distance measure

Return type float

bayescmd.abc.distances.**mean_square_error_dist**(*data1*, *data2*)

Get the Mean Square Error distance between two numpy arrays.

Parameters

- **data1** (*np.ndarray*) – First data array.

  The shape should match that of data2 and the number of rows should match the number of model outputs i.e. 2 model outputs will be two rows.

- **data2** (*np.ndarray*) – Second data array.

  The shape should match that of data1 and the number of rows should match the number of model outputs i.e. 2 model outputs will be two rows.

Returns **d** – Mean Square Error distance measure

Return type float

bayescmd.abc.distances.**zero_array**(*array*, *zero_flag*)

Zero an array of data with its initial values.

Parameters

- **array** (*list*) – List of data
- **zero_flags** (*bool*) – Boolean indicating if data needs zeroing

Returns **zerod** – Zero'd list

Return type list

CHAPTER

THREE

JSONPARSING

..automodule:: bayescmd.jsonParsing.modelJSON

CHAPTER

FOUR

MISCELLANEOUS

Here you will find a number of useful functions that are used throughout the general BayesCMD package.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

INDEX