# Inference tools Documentation

*Release 0.4*

Chris Bowman

Jan 29, 2019

# CONTENTS

CHAPTER

ONE

# INTRODUCTION

This package aims to provide a set of python-based tools for Bayesian data analysis which are simple to use, allowing them to applied quickly and easily.

This documentation contains full descriptions for all available classes (and their methods), but does not yet contain detailed code examples of how to use these classes.

For example code, instead see the /demos/ folder which contains a demonstration code for every available class, including detailed comments.

Requests for features/improvements can be made via the issue tracker. If you have questions or are interested in getting involved with the development of this package, please contact me at `chris.bowman@york.ac.uk`.

Below is a quick reference for each class in the package:

| Class | Description |
|---|---|
| *GibbsChain* | A Gibbs-sampling class for performing Markov-chain Monte-Carlo sampling. |
| *HamiltonianChain* | A class for performing Hamiltonian Markov-chain Monte-Carlo sampling. |
| *GaussianKDE* | Provides a function which approximates a PDF using Gaussian kernel-density estimation. |
| *UnimodalPdf* | Model-based estimation of PDFs for unimodal, univariate distributions. |
| *GpRegressor* | Gaussian-process regression for unstructured data in one or more dimensions. |
| *GpOptimiser* | Bounded global optimisation using Gaussian-processes. |

CHAPTER

# TWO

# THE MCMC MODULE

This module is designed to provide Markov-Chain Monte-Carlo (MCMC) samplers which can be easily applied to inference problems.

## 2.1 GibbsChain

Gibbs samping is implemented as the GibbsChain class. This implementation is self-tuning, such that detailed knowledge of the PDF being sampled is not required in order for the algorithm to work efficiently.

Demonstrations of how to use the GibbsChain class can be found in `/demos/GibbsChain_demo.py` and `/demos/spectroscopy_demo.py`

**class** `inference.mcmc.`**`GibbsChain`**(**args*, ***kwargs*)

A Gibbs sampling class implemented as a child of the MarkovChain class.

**Parameters**

- **`posterior`** (*`func`*) – a function which returns the log-posterior probability density for a given set of model parameters theta, which should be the only argument so that: ln(P) = posterior(theta)
- **`start`** – vector of model parameters which correspond to the parameter-space coordinates at which the chain will start.
- **`widths`** – vector of standard deviations which serve as initial guesses for the widths of the proposal distribution for each model parameter. If not specified, the starting widths will be approximated as 1% of the values in 'start'.

In Gibbs sampling, each "step" in the chain consists of a series of 1D Metropolis-Hastings steps, one for each parameter, such that each step all parameters have been adjusted.

This allows 1D step acceptance rate data to be collected independently for each parameter, thereby allowing the proposal width of each parameter to be tuned individually.

**`advance`**(*m*)

Advances the chain by taking *m* new steps.

**Parameters** **`m`** (*`int`*) – number of steps the chain will advance.

**`get_interval`**(*interval=None*, *burn=None*, *thin=None*, *samples=None*)

Return the samples from the chain which lie inside a chosen highest-density interval.

**Parameters**

- **`interval`** (*`float`*) – Total probability of the desired interval. For example, if interval = 0.95, then the samples corresponding to the top 95% of posterior probability values are returned.

- **burn** (*int*) – Number of samples to discard from the start of the chain. If not specified, the value of self.burn is used instead.
- **thin** (*int*) – Instead of returning every sample which is not discarded as part of the burn-in, every *m*'th sample is returned for a specified integer *m*. If not specified, the value of self.thin is used instead.
- **samples** (*int*) – The number of samples that should be returned from the requested interval. Note that specifying *samples* overrides the value of *thin*.

**Returns** List containing sample points stored as tuples, and a corresponding list of log-probability values

**get_parameter** (*n*, *burn=None*, *thin=None*)
Return sample values for a chosen parameter.

**Parameters**

- **n** (*int*) – Index of the parameter for which samples are to be returned.
- **burn** (*int*) – Number of samples to discard from the start of the chain. If not specified, the value of self.burn is used instead.
- **thin** (*int*) – Instead of returning every sample which is not discarded as part of the burn-in, every *m*'th sample is returned for a specified integer *m*. If not specified, the value of self.thin is used instead.

**Returns** List of samples for parameter *n*'th parameter.

**get_sample** (*burn=None*, *thin=None*)
Return the sample generated by the chain as a list of tuples

**Parameters**

- **burn** (*int*) – Number of samples to discard from the start of the chain. If not specified, the value of self.burn is used instead.
- **thin** (*int*) – Instead of returning every sample which is not discarded as part of the burn-in, every *m*'th sample is returned for a specified integer *m*. If not specified, the value of self.thin is used instead.

**Returns** List containing sample points stored as tuples.

**marginalise** (*n*, *thin=None*, *burn=None*, *unimodal=False*)
Estimate the 1D marginal distribution of a chosen parameter.

**Parameters**

- **n** (*int*) – Index of the parameter for which the marginal distribution is to be estimated.
- **burn** (*int*) – Number of samples to discard from the start of the chain. If not specified, the value of self.burn is used instead.
- **thin** (*int*) – Rather than using every sample which is not discarded as part of the burn-in, every *m*'th sample is used for a specified integer *m*. If not specified, the value of self.thin is used instead, which has a default value of 1.
- **unimodal** (*bool*) – Selects the type of density estimation to be used. The default value is False, which causes a GaussianKDE object to be returned. If however the marginal distribution being estimated is known to be unimodal, setting *unimodal = True* will result in the UnimodalPdf class being used to estimate the density.

Returns one of two 'density estimator' objects which can be called as functions to return the estimated PDF at any point.

**matrix_plot** (*params=None*, *thin=None*, *burn=None*, *labels=None*, *show=True*, *reference=None*, *filename=None*)
Construct a 'matrix plot' of the parameters (or a subset) which displays all 1D and 2D marginal distributions.

**Parameters**

- **params** – A list of integers specifying the indices of parameters which are to be plotted.
- **burn** (*int*) – Number of samples to discard from the start of the chain. If not specified, the value of self.burn is used instead.
- **thin** (*int*) – Rather than using every sample which is not discarded as part of the burn-in, every *m*'th sample is used for a specified integer *m*. If not specified, the value of self.thin is used instead, which has a default value of 1.
- **labels** – A list or tuple of strings to be used axis labels for each parameter being plotted.
- **show** (*bool*) – If set to True, the plot is displayed.
- **reference** – A list of reference values for each parameter which will be over-plotted.
- **filename** (*str*) – File path to which the matrix plot will be saved. If unspecified the plot will be displayed but not saved.

**mode**()

Return the sample with the current highest posterior probability.

**Returns** Tuple containing parameter values.

**plot_diagnostics**(*show=True*, *filename=None*)

Plot diagnostic traces that give information on how the chain is progressing.

Currently this method plots:

- The posterior log-probability as a function of step number, which is useful for checking if the chain has reached a maximum. Any early parts of the chain where the probability is rising rapidly should be removed as burn-in.
- The history of changes to the proposal widths for each parameter. Ideally, the proposal widths should converge, and the point in the chain where this occurs is often a good choice for the end of the burn-in. For highly-correlated pdfs, the proposal widths may never fully converge, but in these cases small fluctuations in the width values are acceptable.

**Parameters**

- **show** (*bool*) – If set to True, the plot is displayed.
- **filename** (*str*) – File path to which the diagnostics plot will be saved. If left unspecified the plot won't be saved.

**run_for**(*minutes=0*, *hours=0*, *days=0*)

Advances the chain for a chosen amount of computation time

**Parameters**

- **minutes** (*int*) – number of minutes for which to run the chain.
- **hours** (*int*) – number of hours for which to run the chain.
- **days** (*int*) – number of days for which to run the chain.

**set_boundaries**(*parameter*, *boundaries*, *remove=False*)

Constrain the value of a particular parameter to specified boundaries.

**Parameters**

- **parameter** (*int*) – Index of the parameter for which boundaries are to be set.

- **`boundaries`** – Tuple of boundaries in the format (lower_limit, upper_limit)

**`set_non_negative`**(*parameter*, *flag=True*)

Constrain a particular parameter to have non-negative values.

**Parameters** **`parameter`** (*`int`*) – Index of the parameter which is to be set as non-negative.

## 2.2 HamiltonianChain

Hamiltonian Monte-Carlo (HMC) is an algorithm where proposed steps are generated by integrating Hamilton's equations, treating the negative posterior log-probability as a scalar potential. In order to do this, the algorithm requires the gradient of the log-posterior. Assuming this gradient can be calculated efficiently, HMC deals well with strongly correlated variables and scales favourably to higher-dimensionality problems.

This implementation automatically selects an appropriate time-step for the Hamiltonian dynamics simulation, but does not currently automatically select an appropriate number of time-steps to take. We would like to add this functionality in the future, for example by implementing the NUTS algorithm.

Demonstrations of how to use the HamiltonianChain class can be found in `/demos/HamiltonianChain_demo.py`

**`class`** `inference.mcmc.`**`HamiltonianChain`**(*posterior=None*, *grad=None*, *start=None*, *epsilon=0.1*, *temperature=1*, *bounds=None*, *inv_mass=None*)

Hamiltonian Monte-Carlo implemented as a child of the MarkovChain class.

**Parameters**

- **`posterior`** (*`func`*) – A function which returns the log-posterior probability density for a given set of model parameters theta, which should be the only argument so that: ln(P) = posterior(theta)
- **`grad`** (*`func`*) – A function which returns the gradient of the log-posterior probability density for a given set of model parameters theta. If this function is not given, the gradient will instead be estimated by finite difference.
- **`start`** – Vector of model parameters which correspond to the parameter-space coordinates at which the chain will start.
- **`epsilon`** (*`float`*) – Initial guess for the time-step of the Hamiltonian dynamics simulation.
- **`temperature`** (*`int`*) – The temperature of the markov chain. This parameter is used for parallel tempering and should be otherwise left unspecified.
- **`bounds`** – A list or tuple containing two numpy arrays which specify the upper and lower bounds for the parameters in the form (lower_bounds, upper_bounds).
- **`inv_mass`** – A vector specifying the inverse-mass value to be used for each parameter. The inverse-mass effectively re-scales the parameters to make the problem more isotropic, which helps ensure good performance. The inverse-mass value for a given parameter should be set to roughly the range over which that parameter is expected to vary.

**`advance`**(*m*)

Advances the chain by taking *m* new steps.

**Parameters** **`m`** (*`int`*) – number of steps the chain will advance.

**`get_parameter`**(*n*, *burn=None*, *thin=None*)

Return sample values for a chosen parameter.

**Parameters**

- **n** (*int*) – Index of the parameter for which samples are to be returned.
- **burn** (*int*) – Number of samples to discard from the start of the chain. If not specified, the value of self.burn is used instead.
- **thin** (*int*) – Instead of returning every sample which is not discarded as part of the burn-in, every *m*'th sample is returned for a specified integer *m*. If not specified, the value of self.thin is used instead.

**Returns** List of samples for parameter *n*'th parameter.

**marginalise**(*n*, *thin=None*, *burn=None*, *unimodal=False*)

Estimate the 1D marginal distribution of a chosen parameter.

**Parameters**

- **n** (*int*) – Index of the parameter for which the marginal distribution is to be estimated.
- **burn** (*int*) – Number of samples to discard from the start of the chain. If not specified, the value of self.burn is used instead.
- **thin** (*int*) – Rather than using every sample which is not discarded as part of the burn-in, every *m*'th sample is used for a specified integer *m*. If not specified, the value of self.thin is used instead, which has a default value of 1.
- **unimodal** (*bool*) – Selects the type of density estimation to be used. The default value is False, which causes a GaussianKDE object to be returned. If however the marginal distribution being estimated is known to be unimodal, setting *unimodal = True* will result in the UnimodalPdf class being used to estimate the density.

Returns one of two 'density estimator' objects which can be called as functions to return the estimated PDF at any point.

**matrix_plot**(*params=None*, *thin=None*, *burn=None*, *labels=None*, *show=True*, *reference=None*, *filename=None*)

Construct a 'matrix plot' of the parameters (or a subset) which displays all 1D and 2D marginal distributions.

**Parameters**

- **params** – A list of integers specifying the indices of parameters which are to be plotted.
- **burn** (*int*) – Number of samples to discard from the start of the chain. If not specified, the value of self.burn is used instead.
- **thin** (*int*) – Rather than using every sample which is not discarded as part of the burn-in, every *m*'th sample is used for a specified integer *m*. If not specified, the value of self.thin is used instead, which has a default value of 1.
- **labels** – A list or tuple of strings to be used axis labels for each parameter being plotted.
- **show** (*bool*) – If set to True, the plot is displayed.
- **reference** – A list of reference values for each parameter which will be over-plotted.
- **filename** (*str*) – File path to which the matrix plot will be saved. If unspecified the plot will be displayed but not saved.

**mode**()

Return the sample with the current highest posterior probability.

**Returns** Tuple containing parameter values.

**plot_diagnostics**(*show=True*, *filename=None*)

Plot diagnostic traces that give information on how the chain is progressing.

Currently this method plots:

- The posterior log-probability as a function of step number, which is useful for checking if the chain has reached a maximum. Any early parts of the chain where the probability is rising rapidly should be removed as burn-in.
- The history of changes to the proposal widths for each parameter. Ideally, the proposal widths should converge, and the point in the chain where this occurs is often a good choice for the end of the burn-in. For highly-correlated pdfs, the proposal widths may never fully converge, but in these cases small fluctuations in the width values are acceptable.

**Parameters**

- **`show`** (*`bool`*) – If set to True, the plot is displayed.
- **`filename`** (*`str`*) – File path to which the diagnostics plot will be saved. If left unspecified the plot won't be saved.

**`run_for`**(*minutes=0*, *hours=0*, *days=0*)

Advances the chain for a chosen amount of computation time

**Parameters**

- **`minutes`** (*`int`*) – number of minutes for which to run the chain.
- **`hours`** (*`int`*) – number of hours for which to run the chain.
- **`days`** (*`int`*) – number of days for which to run the chain.

CHAPTER

THREE

# THE PDF_TOOLS MODULE

This module provides tools for reconstructing the probability density function from which a given set of samples was drawn.

## 3.1 GaussianKDE

The GaussianKDE class provides an estimate of a univariate PDF for a given set of sample data using Gaussian kernel density estimation. An estimate of the mode is calulated automatically, and like UnimodalPdf, the credible interval for a given percentile is available through a method call.

A demonstration of how to use the UnimodalPdf class can be found in `/demos/GaussianKDE_demo.py`

**class** `inference.pdf_tools.`**GaussianKDE**(*sample*, *bandwidth=None*, *cross_validation=False*, *max_cv_samples=5000*)

Construct an estimate of a univariate probability distribution.

**Parameters**

- **sample** (*array-like*) – 1D array of samples from which to estimate the probability distribution
- **bandwidth** (*float*) – width of the Gaussian kernels used for the estimate. If not specified, an appropriate width is estimated based on sample data.
- **cross_validation** (*bool*) – Indicate whether or not cross-validation should be used to estimate the bandwidth in place of the simple 'rule of thumb' estimate which is normally used.
- **max_cv_samples** (*int*) – The maximum number of samples to be used when estimating the bandwidth via cross-validation. The computational cost scales roughly quadratically with the number of samples used, and can become prohibitive for samples of size in the tens of thousands and up. Instead, if the sample size is greater than *max_cv_samples*, the cross-validation is performed on a sub-sample of this size.

The GaussianKDE class uses Gaussian kernel-density estimation to approximate a probability distribution given a sample drawn from that distribution.

**__call__**(*x_vals*)

Evaluate the PDF estimate at a set of given axis positions.

**Parameters** **x_vals** – axis location(s) at which to evaluate the estimate.

**Returns** values of the PDF estimate at the specified locations.

**interval**(*frac=0.95*)

Calculate the highest-density interval(s) which contain a given fraction of total probability.

**Parameters** **frac** (*float*) – Fraction of total probability contained by the desired interval(s).

**Returns** A list of tuples which specify the intervals.

**`mode = None`**
The mode of the pdf, calculated automatically when an instance of GaussianKDE is created.

**`plot_summary`**(*filename=None*, *show=True*)
Plot the estimated PDF along with summary statistics.

**Parameters**

- **`filename`** (*`str`*) – Filename to which the plot will be saved. If unspecified, the plot will not be saved.
- **`show`** (*`bool`*) – Boolean value indicating whether the plot should be displayed in a window. (Default is True)

## 3.2 UnimodalPdf

UnimodalPdf provides smooth estimates of univariate, unimodal PDFs by fitting an extremely flexible parametric model to a given set of sample data. The class also provides a method which returns the credible interval for a chosen percentile.

A demonstration of how to use the UnimodalPdf class can be found in `/demos/UnimodalPdf_demo.py`

**`class`** `inference.pdf_tools.`**`UnimodalPdf`**(*sample*)
Construct an estimate of a univariate, unimodal probability distribution based on a given set of samples.

**Parameters** **`sample`** (*`array-like`*) – 1D array of samples from which to estimate the probability distribution

The UnimodalPdf class is designed to robustly estimate univariate, unimodal probability distributions given a sample drawn from that distribution.

This is a parametric method based on an extensively modified student-t distribution, which is extremely flexible.

**`__call__`**(*x*)
Evaluate the PDF estimate at a set of given axis positions.

**Parameters** **`x_vals`** – axis location(s) at which to evaluate the estimate.

**Returns** values of the PDF estimate at the specified locations.

**`mode = None`**
The mode of the pdf, calculated automatically when an instance of UnimodalPdf is created.

**`plot_summary`**(*filename=None*, *show=True*)
Plot the estimated PDF along with summary statistics.

**Parameters**

- **`filename`** (*`str`*) – Filename to which the plot will be saved. If unspecified, the plot will not be saved.
- **`show`** (*`bool`*) – Boolean value indicating whether the plot should be displayed in a window. (Default is True)

CHAPTER

FOUR

# THE GP_TOOLS MODULE

This module provides implementations of some useful applications of ‘Gaussian processes’. This involves modelling data through multivariate normal distributions where the covariance of any two points is defined by the ‘distance’ between them in some arbitrary space.

## 4.1 GpRegressor

Gaussian-process regression has been implemented as the GpRegressor class, and can fit data which is arbitrarily spaced (i.e. non-gridded) in any number of dimensions. A key advantage this technique holds over other regression methods is its ability to properly account for errors on the data and provide a corresponding error on the regression estimate.

A demonstration of how to use the GpRegressor class can be found in `/demos/GpRegressor_demo.py`

**class** `inference.gp_tools.`**GpRegressor**(*x*, *y*, *y_err=None*, *scale_lengths=None*, *hyperpars=None*)

A class for performing Gaussian-process regression in one or more dimensions.

**Parameters**

- **x** – The spatial coordinates of the y-data values. For the 1-dimensional case, this should be a list or array of floats. For greater than 1 dimension, a list of coordinate arrays or tuples should be given.
- **y** – The y-data values as a list or array of floats.
- **y_err** – The error on the y-data values supplied as a list or array of floats. This technique explicitly assumes that errors are Gaussian, so the supplied error values represent normal distribution standard deviations. If this argument is not specified the errors are taken to be small but non-zero.
- **scale_lengths** – The default behaviour of GpRegressor is to determine an appropriate scale-length for each dimension separately, such that for a problem with N dimensions, there are N+1 total hyperparameters. Alternatively, this can be reduced to only 2 hyperparameters regardless of the number of dimensions by specifying the scale_lengths argument. In this case, the hyperparameters become and amplitude and a scalar multiplier for the provided scale-lengths. The specified lengths must be given as an iterable of length equal to the number of dimensions.
- **hyperpars** – The amplitude and scale-length parameters for the normal prior distribution. If a single global scale length should be used, the hyperparameters should be specified as a two element list, i.e. [amplitude, length]. Alternatively, a separate length-scale for each dimension can be specified by passing an amplitude followed by iterable of lengths, i.e. [amplitude, (L1, L2, ...)].

**__call__**(*q*, *theta=None*)

Calculate the mean and standard deviation of the regression estimate at a series of specified spatial points.

**Parameters** **q** – A list containing the spatial locations where the mean and standard deviation of the estimate is to be calculated. In the 1D case this would be a list of floats, or a list of coordinate tuples in the multi-dimensional case.

**Returns** Two 1D arrays, the first containing the means and the second containing the sigma values.

**build_posterior**(*q*)

Generates the full mean vector and covariance matrix for the GP fit at a set of specified points 'q'.

**Parameters** **q** – A list containing the spatial locations which will be used to construct the Gaussian process. In the 1D case this would be a list of floats, or a list of coordinate tuples in the multi-dimensional case.

**Returns** The mean vector as a 1D array, followed by covariance matrix as a 2D array.

## 4.2 GpOptimiser

Bounded global optimisation using gaussian-process regression has been implemented through the GpOptimiser class. This algorithm, often referred to as "Bayesian optimisation" specifically suited to problems where a single evaluation of the function which is to be maximised is very expensive, such that the total number of evaluations must be minimised.

A demonstration of how to use the GpOptimiser class can be found in `/demos/GpOptimiser_demo.py`

**class** `inference.gp_tools.`**GpOptimiser**(*x*, *y*, *y_err=None*, *bounds=None*)

A class for performing Gaussian-process optimisation in one or more dimensions.

GpOptimiser extends the functionality of GpRegressor to perform Gaussian-process optimisation, often also referred to as 'Bayesian optimisation'. This technique is suited to problems for which a single evaluation of the function being explored is expensive, such that the total number of function evaluations must be made as small as possible.

In order to construct the gaussian-process regression estimate which is used to search for the global maximum, on initialisation GpOptimiser must be provided with at least two evaluations of the function which is to be maximised.

**Parameters**

- **x** – The spatial coordinates of the y-data values. For the 1-dimensional case, this should be a list or array of floats. For greater than 1 dimension, a list of coordinate arrays or tuples should be given.
- **y** – The y-data values as a list or array of floats.
- **y_err** – The error on the y-data values supplied as a list or array of floats. This technique explicitly assumes that errors are Gaussian, so the supplied error values represent normal distribution standard deviations. If this argument is not specified the errors are taken to be small but non-zero.
- **bounds** – A iterable containing tuples which specify for the upper and lower bounds for the optimisation in each dimension in the format (lower_bound, upper_bound).

**add_evaluation**(*new_x*, *new_y*, *new_y_err=None*)

Add the latest evaluation to the data set and re-build the Gaussian process so a new proposed evaluation can be made.

**Parameters**

- **new_x** – location of the new evaluation
- **new_y** – function value of the new evaluation
- **new_y_err** – Error of the new evaluation.

**`search_for_maximum()`**

Request a proposed location for the next evaluation. This proposal is selected in order to maximise the "expected improvement" criteria which searches for the global maximum value of the function.

**Returns** location of the next proposed evaluation.

# INDEX