chippr

Release 0.1.0

Dec 13, 2019

Contents

This Python package enables estimation of cosmological quantities using photometric redshift probability distributions.

See the following IPython Notebook for an example of using *chippr*:

- Basic Demo

CHAPTER 1

Simulation

chippr enables simulation of surveys of photo-z interim posteriors.

1.1 The discrete Class

class discrete.**discrete**(*bin_ends*, *weights*)

evaluate(*xs*)
Function to evaluate the discrete probability distribution at many points

Parameters **xs** (*ndarray, float*) – values at which to evaluate discrete probability distribution

Returns **ps** – values of discrete probability distribution at xs

Return type ndarray, float

evaluate_one(*x*)
Function to evaluate the discrete probability distribution at one point

Parameters **x** (*float*) – value at which to evaluate discrete probability distribution

Returns **p** – value of discrete probability distribution at x

Return type float

pdf(*xs*)

sample(*n_samps*)
Function to take samples from discrete probability distribution

Parameters **n_samps** (*int*) – number of samples to take

Returns **xs** – array of points sampled from the discrete probability distribution

Return type ndarray, float

sample_one()
Function to sample a single value from discrete probability distribution

Returns **x** – a single point sampled from the discrete probability distribution

Return type float

1.2 The gauss Class

class gauss.**gauss**(*mean*, *var*, *bounds=None*)

evaluate(*xs*)
Function to evaluate univariate Gaussian probability distribution at multiple points

Parameters **xs** (*numpy.ndarray, float*) – input values at which to evaluate probability

Returns **ps** – output probabilities

Return type ndarray, float

evaluate_one(*x*)
Function to evaluate Gaussian probability distribution once

Parameters **x** (*float*) – value at which to evaluate Gaussian probability distribution

Returns **p** – probability associated with x

Return type float

invert_var()
Function to invert variance

norm_var()
Function to create standard deviation from variance

pdf(*xs*)

sample(*n_samps*)
Function to sample univariate Gaussian probability distribution

Parameters **n_samps** (*positive int*) – number of samples to take

Returns **xs** – array of n_samps samples from Gaussian probability distribution

Return type ndarray, float

sample_one()
Function to take one sample from univariate Gaussian probability distribution

Returns **x** – single sample from Gaussian probability distribution

Return type float

1.3 The gmix Class

class gmix.**gmix**(*amps*, *funcs*, *limits=(0.001, 3.501)*)

evaluate(*xs*)
Function to evaluate the Gaussian mixture probability distribution at many points

Parameters **xs** (*ndarray, float*) – values at which to evaluate Gaussian mixture probability distribution

Returns **ps** – values of Gaussian mixture probability distribution at xs

Return type ndarray, float

evaluate_one(*x*)
Function to evaluate Gaussian mixture once

Parameters **x** (*float*) – value at which to evaluate Gaussian mixture

Returns **p** – probability associated with x

Return type float

pdf(*xs*)

sample(*n_samps*)
Function to take samples from Gaussian mixture probability distribution

Parameters **n_samps** (*int*) – number of samples to take

Returns **xs** – array of points sampled from the Gaussian mixture probability distribution

Return type ndarray, float

sample_one()
Function to sample a single value from Gaussian mixture probability distribution

Returns **x** – a single point sampled from the Gaussian mixture probability distribution

Return type float

1.4 The mvn Class

class mvn.**mvn**(*mean*, *var*)

evaluate(*zs*)
Function to evaluate multivariate Gaussian probability distribution at multiple points

Parameters **zs** (*ndarray, float*) – input vectors at which to evaluate probability

Returns **ps** – output probabilities

Return type ndarray, float

evaluate_one(*z*)
Function to evaluate multivariate Gaussian probability distribution once

Parameters **z** (*numpy.ndarray, float*) – value at which to evaluate multivariate Gaussian probability distribution

Returns **p** – probability associated with z

Return type float

invert_var()
Function to invert covariance matrix

Returns **inv** – inverse variance

Return type numpy.ndarray, float

norm_var()
Function to normalize covariance matrix

Returns **det** – determinant of variance

Return type float

pdf(*points*)

sample(*n_samps*)
Function to sample from multivariate Gaussian probability distribution

Parameters **n_samps** (*positive int*) – number of samples to take

Returns **zs** – array of n_samps samples from multivariate Gaussian probability distribution

Return type ndarray, float

sample_one()
Function to take one sample from multivariate Gaussian probability distribution

Returns **z** – single sample from multivariate Gaussian probability distribution

Return type numpy.ndarray, float

1.5 The catalog Class

class catalog.**catalog**(*params={}*, *vb=True*, *loc='.'*, *prepend=''*)

coarsify(*fine*)
Function to bin function evaluated on fine grid

Parameters **fine** (*numpy.ndarray, float*) – matrix of probability values of function on fine grid for N galaxies

Returns **coarse** – vector of binned values of function

Return type numpy.ndarray, float

create(*truth*, *int_pr*, *N=4*, *vb=True*)
Function creating a catalog of interim posterior probability distributions, will split this up into helper functions

Parameters

- **truth** (*chippr.gmix object or chippr.gauss object or chippr.discrete*) –
- **object** – true redshift distribution object
- **int_pr** (*chippr.gmix object or chippr.gauss object or chippr.discrete*) –
- **object** – interim prior distribution object
- **vb** (*boolean, optional*) – True to print progress messages to stdout, False to suppress

Returns **self.cat** – dictionary comprising catalog information

Return type dict

evaluate_lfs (*pspace*, *vb=True*)
Evaluates likelihoods based on observed sample values

Parameters

- **pspace** (*chippr.gauss or chippr.gmix or chippr.gamma or chippr.multi object*) – the probability function to evaluate
- **vb** (*boolean*) – print progress to stdout?

Returns **lfs** – array of likelihood values for each item as a function of fine binning

Return type numpy.ndarray, float

make_probs (*vb=True*)
Makes the continuous 2D probability distribution over z_spec, z_phot

Parameters **vb** (*boolean*) – print progress to stdout?

Notes

TO DO: only one outlier population at a time for now, will enable more TO DO: also doesn't yet include perpendicular features from passing between filter curves, should add that

proc_bins (*vb=True*)
Function to process binning

Parameters **vb** (*boolean, optional*) – True to print progress messages to stdout, False to suppress

read (*loc='data'*, *style='.txt'*)
Function to read in catalog file

Parameters **loc** (*string, optional*) – location of catalog file

sample (*N*, *vb=False*)
Samples (z_spec, z_phot) pairs

Parameters

- **N** (*int*) – number of samples to take
- **vb** (*boolean*) – print progress to stdout?

Returns **samps** – (z_spec, z_phot) pairs

Return type numpy.ndarray, float

write (*loc='data'*, *style='.txt'*)
Function to write newly-created catalog to file

Parameters

- **loc** (*string, optional*) – file name into which to save catalog
- **style** (*string, optional*) – file format in which to save the catalog

CHAPTER 2

Inference

chippr currently enables estimation of the redshift density function.

2.1 The log_z_dens Class

class log_z_dens.**log_z_dens**(*catalog*, *hyperprior*, *truth=None*, *loc='.'*, *prepend=''*, *vb=True*)

calculate_mexp(*vb=True*)
Calculates the marginalized expected value estimator of the redshift density function

Parameters **vb** (*boolean, optional*) – True to print progress messages to stdout, False to suppress

Returns **log_exp_nz** – array of logged redshift density function bin values

Return type ndarray, float

calculate_mmap(*vb=True*)
Calculates the marginalized maximum a posteriori estimator of the redshift density function

Parameters **vb** (*boolean, optional*) – True to print progress messages to stdout, False to suppress

Returns **log_map_nz** – array of logged redshift density function bin values

Return type ndarray, float

calculate_mmle(*start*, *vb=True*, *no_data=0*, *no_prior=0*)
Calculates the marginalized maximum likelihood estimator of the redshift density function

Parameters

- **start** (*numpy.ndarray, float*) – array of log redshift density function bin values at which to begin optimization

- **vb** (*boolean, optional*) – True to print progress messages to stdout, False to suppress
- **no_data** (*boolean, optional*) – True to exclude data contribution to hyperposterior
- **no_prior** (*boolean, optional*) – True to exclude prior contribution to hyperposterior

Returns **log_mle_nz** – array of logged redshift density function bin values maximizing hyperposterior

Return type numpy.ndarray, float

calculate_samples(*ivals*, *n_accepted=3*, *n_burned=2*, *vb=True*, *n_procs=1*, *no_data=0*, *no_prior=0*, *gr_threshold=1.2*)

Calculates samples estimating the redshift density function

Parameters

- **ivals** (*numpy.ndarray, float*) – initial values of log n(z) for each walker
- **n_accepted** (*int, optional*) – log10 number of samples to accept per walker
- **n_burned** (*int, optional*) – log10 number of samples between tests of burn-in condition
- **n_procs** (*int, optional*) – number of processors to use, defaults to single-thread
- **vb** (*boolean, optional*) – True to print progress messages to stdout, False to suppress
- **no_data** (*boolean, optional*) – True to exclude data contribution to hyperposterior
- **no_prior** (*boolean, optional*) – True to exclude prior contribution to hyperposterior

Returns **log_samples_nz** – array of sampled log redshift density function bin values

Return type ndarray, float

calculate_stacked(*vb=True*)

Calculates the stacked estimator of the redshift density function

Parameters **vb** (*boolean, optional*) – True to print progress messages to stdout, False to suppress

Returns **log_stk_nz** – array of logged redshift density function bin values

Return type ndarray, float

compare(*vb=True*)

Calculates all available goodness of fit measures

Parameters **vb** (*boolean, optional*) – True to print progress messages to stdout, False to suppress

Returns **out_info** – dictionary of all available statistics

Return type dict

evaluate_log_hyper_likelihood(*log_nz*)

Function to evaluate log hyperlikelihood

Parameters **log_nz** (*numpy.ndarray, float*) – vector of logged redshift density bin values at which to evaluate the hyperlikelihood

Returns **log_hyper_likelihood** – log likelihood probability associated with parameters in log_nz

Return type float

evaluate_log_hyper_posterior(*log_nz*)
Function to evaluate log hyperposterior

Parameters **log_nz** (*numpy.ndarray, float*) – vector of logged redshift density bin values at which to evaluate the full posterior

Returns **log_hyper_posterior** – log hyperposterior probability associated with parameters in log_nz

Return type float

evaluate_log_hyper_prior(*log_nz*)
Function to evaluate log hyperprior

Parameters **log_nz** (*numpy.ndarray, float*) – vector of logged redshift density bin values at which to evaluate the hyperprior

Returns **log_hyper_prior** – log prior probability associated with parameters in log_nz

Return type float

optimize(*start*, *no_data*, *no_prior*, *vb=True*)
Maximizes the hyperposterior of the redshift density

Parameters

- **start** (*numpy.ndarray, float*) – array of log redshift density function bin values at which to begin optimization
- **no_data** (*boolean*) – True to exclude data contribution to hyperposterior
- **no_prior** (*boolean*) – True to exclude prior contribution to hyperposterior
- **vb** (*boolean, optional*) – True to print progress messages to stdout, False to suppress

Returns **res.x** – array of logged redshift density function bin values maximizing hyperposterior

Return type numpy.ndarray, float

plot_estimators(*log=True*, *mini=True*)
Plots all available estimators of the redshift density function.

read(*read_loc*, *style='pickle'*, *vb=True*)
Function to load inferred quantities from files.

Parameters

- **read_loc** (*string*) – filepath where inferred redshift density function is stored
- **style** (*string, optional*) – keyword for file format, currently only 'pickle' supported
- **vb** (*boolean, optional*) – True to print progress messages to stdout, False to suppress

Returns **self.info** – returns the log_z_dens information dictionary object

Return type dict

sample(*ivals*, *n_samps*, *vb=True*)
Samples the redshift density hyperposterior

Parameters

- **ivals** (*numpy.ndarray, float*) – initial values of the walkers
- **n_samps** (*int*) – number of samples to accept before stopping
- **vb** (*boolean, optional*) – True to print progress messages to stdout, False to suppress

Returns **mcmc_outputs** – dictionary containing array of sampled redshift density function bin values as well as posterior probabilities, acceptance fractions, and autocorrelation times

Return type dict

write(*write_loc*, *style='pickle'*, *vb=True*)

Function to write results of inference to files.

Parameters

- **write_loc** (*string*) – filepath where results of inference should be saved.
- **style** (*string, optional*) – keyword for file format, currently only 'pickle' supported
- **vb** (*boolean, optional*) – True to print progress messages to stdout, False to suppress

CHAPTER 3

Utilities

chippr includes a number of modules containing helper functions.

3.1 Default Settings

defaults.**check_basic_setup**(*params*)
Sets parameter values pertaining to basic constants of simulation

Parameters **params** (*dict*) – dictionary containing key/value pairs for simulation

Returns **params** – dictionary containing key/value pairs for simulation

Return type dict

defaults.**check_bias_params**(*params*)
Sets parameter values pertaining to presence of a systematic bias

Parameters **params** (*dict*) – dictionary containing key/value pairs for simulation

Returns **params** – dictionary containing key/value pairs for simulation

Return type dict

defaults.**check_catastrophic_outliers**(*params*)
Sets parameter values pertaining to presence of a catastrophic outlier population

Parameters **params** (*dict*) – dictionary containing key/value pairs for simulation

Returns **params** – dictionary containing key/value pairs for simulation

Return type dict

Notes

defaults.**check_inf_params**(*params={}*)
Checks inference parameter dictionary for various keywords and sets to default values if not present

Parameters **params** (*dict, optional*) – dictionary containing initial key/value pairs for inference

Returns **params** – dictionary containing final key/value pairs for inference

Return type dict

defaults.**check_sampler_params**(*params*)
Sets parameter values pertaining to basic constants of inference

Parameters **params** (*dict*) – dictionary containing key/value pairs for inference

Returns **params** – dictionary containing key/value pairs for inference

Return type dict

defaults.**check_sim_params**(*params={}*)
Checks simulation parameter dictionary for various keywords and sets to default values if not present

Parameters **params** (*dict, optional*) – dictionary containing initial key/value pairs for simulation of catalog

Returns **params** – dictionary containing final key/value pairs for simulation of catalog

Return type dict

defaults.**check_variable_sigmas**(*params*)
Sets parameter values pertaining to widths of Gaussian PDF components

Parameters **params** (*dict*) – dictionary containing key/value pairs for simulation

Returns **params** – dictionary containing key/value pairs for simulation

Return type dict

Notes

rms_scatter –> variable_sigmas

3.2 General Utilities

utils.**ingest**(*in_info*)
Function reading in parameter file to define functions necessary for generation of posterior probability distributions

Parameters **in_info** (*string or dict*) – string containing path to plaintext input file or dict containing likelihood input parameters

Returns **in_dict** – dict containing keys and values necessary for posterior probability distributions

Return type dict

utils.**safe_log**(*arr*, *threshold=4.450147717014403e-308*)
Takes the natural logarithm of an array that might contain zeros.

Parameters

- **arr** (*ndarray, float*) – array of values to be logged
- **threshold** (*float, optional*) – small, positive value to replace zeros and negative numbers

Returns **logged** – logged values, with small value replacing un-loggable values

Return type ndarray

3.3 Simulation Utilities

sim_utils.**choice**(*weights*)
Function sampling discrete distribution

Parameters **weights** (*numpy.ndarray*) – relative probabilities for each category

Returns **index** – chosen category

Return type int

3.4 Statistics

stat_utils.**acors**(*xtimeswalkersbins*, *mode='bins'*)
Calculates autocorrelation time for MCMC chains

Parameters

- **xtimeswalkersbins** (*numpy.ndarray, float*) – emcee chain values of dimensions (n_iterations, n_walkers, n_parameters)
- **mode** (*string, optional*) – 'bins' for one autocorrelation time per parameter, 'walkers' for one autocorrelation time per walker

Returns **taus** – autocorrelation times by bin or by walker depending on mode

Return type numpy.ndarray, float

stat_utils.**calculate_kld**(*pe*, *qe*, *vb=True*)
Calculates the Kullback-Leibler Divergence between two PDFs.

Parameters

- **pe** (*numpy.ndarray, float*) – probability distribution evaluated on a grid whose distance from q will be calculated.
- **qe** (*numpy.ndarray, float*) – probability distribution evaluated on a grid whose distance to p will be calculated.
- **vb** (*boolean*) – report on progress to stdout?

Returns **Dpq** – the value of the Kullback-Leibler Divergence from q to p

Return type float

stat_utils.**calculate_rms**(*pe*, *qe*, *vb=True*)
Calculates the Root Mean Square Error between two PDFs.

Parameters

- **pe** (*numpy.ndarray, float*) – probability distribution evaluated on a grid whose distance _from_ q will be calculated.
- **qe** (*numpy.ndarray, float*) – probability distribution evaluated on a grid whose distance _to_ p will be calculated.
- **vb** (*boolean*) – report on progress to stdout?

Returns **rms** – the value of the RMS error between q and p

Return type float

stat_utils.**cf**(*xtimes*)
Helper function to calculate autocorrelation time for chain of MCMC samples

Parameters **xtimes** (*numpy.ndarray, float*) – single parameter values for a single walker over all iterations

Returns **cf** – autocorrelation time over all time lags for one parameter of one walker

Return type numpy.ndarray, float

stat_utils.**cfs**(*x*, *mode*)
Helper function for calculating autocorrelation time for MCMC chains

Parameters

- **x** (*numpy.ndarray, float*) – input parameter values of length number of iterations by number of walkers if mode='walkers' or dimension of parameters if mode='bins'
- **mode** (*string*) – 'bins' for one autocorrelation time per parameter, 'walkers' for one autocorrelation time per walker

Returns **cfs** – autocorrelation times for all walkers if mode='walkers' or all parameters if mode='bins'

Return type numpy.ndarray, float

stat_utils.**cft**(*xtimes*, *lag*)
Helper function to calculate autocorrelation time for chain of MCMC samples

Parameters

- **xtimes** (*numpy.ndarray, float*) – single parameter values for a single walker over all iterations
- **lag** (*int*) – maximum lag time in number of iterations

Returns **ans** – autocorrelation time for one time lag for one parameter of one walker

Return type numpy.ndarray, float

stat_utils.**gr_test**(*sample*, *threshold=1.2*)
Performs the Gelman-Rubin test of convergence of an MCMC chain

Parameters

- **sample** (*numpy.ndarray, float*) – chain output
- **threshold** (*float, optional*) – Gelman-Rubin test statistic criterion (usually around 1)

Returns **test_result** – True if burning in, False if post-burn in

Return type boolean

stat_utils.**mean**(*population*)
Calculates the mean of a population

Parameters **population** (*np.array, float*) – population over which to calculate the mean

Returns **mean** – mean value over population

Return type np.array, float

stat_utils.**multi_parameter_gr_stat**(*sample*)
Calculates the Gelman-Rubin test statistic of convergence of an MCMC chain over multiple parameters

Parameters **sample** (*numpy.ndarray, float*) – multi-parameter chain output

Returns **Rs** – vector of the potential scale reduction factors

Return type numpy.ndarray, float

stat_utils.**norm_fit**(*population*)
Calculates the mean and standard deviation of a population

Parameters **population** (*np.array, float*) – population over which to calculate the mean

Returns **norm_stats** – mean and standard deviation over population

Return type tuple, list, float

stat_utils.**single_parameter_gr_stat**(*chain*)
Calculates the Gelman-Rubin test statistic of convergence of an MCMC chain over one parameter

Parameters **chain** (*numpy.ndarray, float*) – single-parameter chain

Returns **R_hat** – potential scale reduction factor

Return type float

3.5 Plotting Utilities

plot_utils.**plot_h**(*sub_plot, bin_ends, to_plot, s='–', c='k', a=1, w=1, d=[(0, (1, 0.0001))], l=None, r=False*)
Helper function to plot horizontal lines of a step function

Parameters

- **sub_plot** (*matplotlib.pyplot subplot object*) – subplot into which step function is drawn
- **bin_ends** (*list or ndarray*) – list or array of endpoints of bins
- **to_plot** (*list or ndarray*) – list or array of values within each bin
- **s** (*string, optional*) – matplotlib.pyplot linestyle
- **c** (*string, optional*) – matplotlib.pyplot color
- **a** (*int or float, [0., 1.], optional*) – matplotlib.pyplot alpha (transparency)
- **w** (*int or float, optional*) – matplotlib.pyplot linewidth
- **d** (*list of tuple, optional*) – matplotlib.pyplot dash style, of form [(start_point, (points_on, points_off, ...))]
- **l** (*string, optional*) – label for function
- **r** (*boolean, optional*) – True for rasterized, False for vectorized

plot_utils.**plot_step**(*sub_plot, bin_ends, to_plot, s='–', c='k', a=1, w=1, d=[(0, (1, 0.0001))], l=None, r=False*)
Plots a step function

Parameters

- **sub_plot** (*matplotlib.pyplot subplot object*) – subplot into which step function is drawn
- **bin_ends** (*list or ndarray*) – list or array of endpoints of bins

- **to_plot** (*list or ndarray*) – list or array of values within each bin
- **s** (*string, optional*) – matplotlib.pyplot linestyle
- **c** (*string, optional*) – matplotlib.pyplot color
- **a** (*int or float, [0., 1.], optional*) – matplotlib.pyplot alpha (transparency)
- **w** (*int or float, optional*) – matplotlib.pyplot linewidth
- **d** (*list of tuple, optional*) – matplotlib.pyplot dash style, of form [(start_point, (points_on, points_off, ...))]
- **l** (*string, optional*) – label for function
- **r** (*boolean, optional*) – True for rasterized, False for vectorized

Notes

Make this not need a subplot

plot_utils.**plot_v**(*sub_plot, bin_ends, to_plot, s='–', c='k', a=1, w=1, d=[(0, (1, 0.0001))], r=False*)
Helper function to plot vertical lines of a step function

Parameters

- **sub_plot** (*matplotlib.pyplot subplot object*) – subplot into which step function is drawn
- **bin_ends** (*list or ndarray*) – list or array of endpoints of bins
- **to_plot** (*list or ndarray*) – list or array of values within each bin
- **s** (*string, optional*) – matplotlib.pyplot linestyle
- **c** (*string, optional*) – matplotlib.pyplot color
- **a** (*int or float, [0., 1.], optional*) – matplotlib.pyplot alpha (transparency)
- **w** (*int or float, optional*) – matplotlib.pyplot linewidth
- **d** (*list of tuple, optional*) – matplotlib.pyplot dash style, of form [(start_point, (points_on, points_off, ...))]
- **r** (*boolean, optional*) – True for rasterized, False for vectorized

plot_utils.**set_up_plot**()
Sets up plots to look decent

Index

N

O

P

R

S

U

W