TensionTools Documentation

Release 0.1.1

Marco Raveri

Jul 02, 2021

Full example

tensiometer is a small python package that extends GetDist capabilities to test the level of agreement/disagreement between different posterior distributions.

The best place to start is by looking at the worked examples below.

Other examples are provided for specific tasks.

CHAPTER 1

tensiometer.mcmc_tension

This module contains the functions and utilities to compute non-Gaussian Monte Carlo tension estimators.

The submodule *param_diff* contains the functions and utilities to compute the distribution of parameter differences from the parameter posterior of two experiments.

The submodule *kde* contains the functions to compute the statistical significance of a difference in parameters with KDE methods.

This submodule *flow* contains the functions and utilities to compute the statistical significance of a difference in parameters with normalizing flow methods.

For more details on the method implemented see arxiv 1806.04649 and arxiv 1912.04880.

`tensiometer.mcmc_tension.param_diff.`**`parameter_diff_chain`**(*chain_1*, *chain_2*, *boost=1*)

Compute the chain of the parameter differences between the two input chains. The parameters of the difference chain are related to the parameters of the input chains, θ_1 and θ_2 by:

$$\Delta\theta \equiv \theta_1 - \theta_2$$

This function only returns the differences for the parameters that are common to both chains. This function preserves the chain separation (if any) so that the convergence of the difference chain can be tested. This function does not assume Gaussianity of the chain. This functions does assume that the parameter determinations from the two chains (i.e. the underlying data sets) are uncorrelated. Do not use this function for chains that are correlated.

Parameters

- **`chain_1`** – `MCSamples` first input chain with n_1 samples
- **`chain_2`** – `MCSamples` second input chain with n_2 samples
- **`boost`** – (optional) boost the number of samples in the difference chain. By default the length of the difference chain will be the length of the longest chain. Given two chains the full difference chain can contain $n_1 \times n_2$ samples but this is usually prohibitive for realistic chains. The boost parameters wil increase the number of samples to be $\mathrm{boost} \times \max(n_1, n_2)$. Default boost parameter is one. If boost is None the full difference chain is going to be computed (and will likely require a lot of memory and time).

Returns MCSamples the instance with the parameter difference chain.

tensiometer.mcmc_tension.param_diff.**parameter_diff_weighted_samples**(*samples_1*, *samples_2*, *boost=1*, *indexes_1=None*, *indexes_2=None*)

Compute the parameter differences of two input weighted samples. The parameters of the difference samples are related to the parameters of the input samples, θ_1 and θ_2 by:

$$\Delta\theta \equiv \theta_1 - \theta_2$$

This function does not assume Gaussianity of the chain. This functions does assume that the parameter determinations from the two chains (i.e. the underlying data sets) are uncorrelated. Do not use this function for chains that are correlated.

Parameters

- **samples_1** – WeightedSamples first input weighted samples with n_1 samples.
- **samples_2** – WeightedSamples second input weighted samples with n_2 samples.
- **boost** – (optional) boost the number of samples in the difference. By default the length of the difference samples will be the length of the longest one. Given two samples the full difference samples can contain $n_1 \times n_2$ samples but this is usually prohibitive for realistic chains. The boost parameters wil increase the number of samples to be $\mathrm{boost} \times \max(n_1, n_2)$. Default boost parameter is one. If boost is None the full difference chain is going to be computed (and will likely require a lot of memory and time).
- **indexes_1** – (optional) array with the indexes of the parameters to use for the first samples. By default this tries to use all parameters.
- **indexes_2** – (optional) array with the indexes of the parameters to use for the second samples. By default this tries to use all parameters.

Returns WeightedSamples the instance with the parameter difference samples.

tensiometer.mcmc_tension.kde.**AMISE_bandwidth**(*num_params*, *num_samples*)

Compute Silverman's rule of thumb bandwidth covariance scaling AMISE. This is the default scaling that is used to compute the KDE estimate of parameter shifts.

Parameters

- **num_params** – the number of parameters in the chain.
- **num_samples** – the number of samples in the chain.

Returns AMISE bandwidth matrix.

Reference Chacón, J. E., Duong, T. (2018). Multivariate Kernel Smoothing and Its Applications. United States: CRC Press.

tensiometer.mcmc_tension.kde.**MAX_bandwidth**(*num_params*, *num_samples*)

Compute the maximum bandwidth matrix. This bandwidth is generally oversmoothing.

Parameters

- **num_params** – the number of parameters in the chain.
- **num_samples** – the number of samples in the chain.

Returns MAX bandwidth matrix.

Reference Chacón, J. E., Duong, T. (2018). Multivariate Kernel Smoothing and Its Applications. United States: CRC Press.

tensiometer.mcmc_tension.kde.**MISE_bandwidth**(*num_params*, *num_samples*, *feedback=0*, ***kwargs*)

Computes the MISE bandwidth matrix by numerically minimizing the MISE over the space of positive definite symmetric matrices.

Parameters

- **num_params** – the number of parameters in the chain.
- **num_samples** – the number of samples in the chain.
- **feedback** – feedback level. If > 2 prints a lot of information.
- **kwargs** – optional arguments to be passed to the optimizer algorithm.

Returns MISE bandwidth matrix.

Reference Chacón, J. E., Duong, T. (2018). Multivariate Kernel Smoothing and Its Applications. United States: CRC Press.

tensiometer.mcmc_tension.kde.**MISE_bandwidth_1d**(*num_params*, *num_samples*, ***kwargs*)

Computes the MISE bandwidth matrix. All coordinates are considered the same so the MISE estimate just rescales the identity matrix.

Parameters

- **num_params** – the number of parameters in the chain.
- **num_samples** – the number of samples in the chain.
- **kwargs** – optional arguments to be passed to the optimizer algorithm.

Returns MISE 1d bandwidth matrix.

Reference Chacón, J. E., Duong, T. (2018). Multivariate Kernel Smoothing and Its Applications. United States: CRC Press.

tensiometer.mcmc_tension.kde.**OptimizeBandwidth_1D**(*diff_chain*, *param_names=None*, *num_bins=1000*)

Compute an estimate of an optimal bandwidth for covariance scaling as in GetDist. This is performed on whitened samples (with identity covariance), in 1D, and then scaled up with a dimensionality correction.

Parameters

- **diff_chain** – MCSamples input parameter difference chain
- **param_names** – (optional) parameter names of the parameters to be used in the calculation. By default all running parameters.
- **num_bins** – number of bins used for the 1D estimate

Returns scaling vector for the whitened parameters

tensiometer.mcmc_tension.kde.**Scotts_bandwidth**(*num_params*, *num_samples*)

Compute Scott's rule of thumb bandwidth covariance scaling. This should be a fast approximation of the 1d MISE estimate.

Parameters

- **num_params** – the number of parameters in the chain.
- **num_samples** – the number of samples in the chain.

Returns Scott's scaling matrix.

Reference Chacón, J. E., Duong, T. (2018). Multivariate Kernel Smoothing and Its Applications. United States: CRC Press.

tensiometer.mcmc_tension.kde.**UCV_SP_bandwidth**(*white_samples*, *weights*, *feedback=0*, *near=1*, *near_max=20*)

Computes the optimal unbiased cross validation bandwidth scaling for the BALL sampling point KDE estimator.

Parameters

- **white_samples** – pre-whitened samples (identity covariance).
- **weights** – input sample weights.
- **feedback** – (optional) how verbose is the algorithm. Default is zero.
- **near** – (optional) number of nearest neighbour to use. Default is 1.
- **near_max** – (optional) number of nearest neighbour to use for the UCV calculation. Default is 20.

tensiometer.mcmc_tension.kde.**UCV_bandwidth**(*weights*, *white_samples*, *alpha0=None*, *feedback=0*, *mode='full'*, ***kwargs*)

Computes the optimal unbiased cross validation bandwidth for the input samples by numerical minimization.

Parameters

- **weights** – input sample weights.
- **white_samples** – pre-whitened samples (identity covariance)
- **alpha0** – (optional) initial guess for the bandwidth. If none is given then the AMISE band is used as the starting point for minimization.
- **feedback** – (optional) how verbose is the algorithm. Default is zero.
- **mode** – (optional) selects the space for minimization. Default is over the full space of SPD matrices. Other options are *diag* to perform minimization over diagonal matrices and *1d* to perform minimization over matrices that are proportional to the identity.
- **kwargs** – other arguments passed to `scipy.optimize.minimize()`

Returns UCV bandwidth matrix.

Reference Chacón, J. E., Duong, T. (2018). Multivariate Kernel Smoothing and Its Applications. United States: CRC Press.

tensiometer.mcmc_tension.kde.**kde_parameter_shift**(*diff_chain*, *param_names=None*, *scale=None*, *method='neighbor_elimination'*, *feedback=1*, ***kwargs*)

Compute the KDE estimate of the probability of a parameter shift given an input parameter difference chain. This function uses a Kernel Density Estimate (KDE) algorithm discussed in (Raveri, Zacharegkas and Hu 19). If the difference chain contains $n_{\rm samples}$ this algorithm scales as $O(n^2_{\rm samples})$ and might require long run times. For this reason the algorithm is parallelized with the joblib library. If the problem is 1d or 2d use the fft algorithm in *kde_parameter_shift_1D_fft()* and *kde_parameter_shift_2D_fft()*.

Parameters

- **diff_chain** – `MCSamples` input parameter difference chain
- **param_names** – (optional) parameter names of the parameters to be used in the calculation. By default all running parameters.

- **scale** – (optional) scale for the KDE smoothing. The scale is always referred to white samples with unit covariance. If none is provided the algorithm uses MISE estimate. Options are:
  1. a scalar for fixed scaling over all dimensions;
  2. a matrix from anisotropic smoothing;
  3. *MISE*, *AMISE*, *MAX* for the corresponding smoothing scale;
  4. *BALL* or *ELL* for variable adaptive smoothing with nearest neighbour;
- **method** – (optional) a string containing the indication for the method to use in the KDE calculation. This can be very intensive so different techniques are provided.
  1. method = *brute_force* is a parallelized brute force method. This method scales as $O(n^2_{\text{samples}})$ and can be afforded only for small tensions. When suspecting a difference that is larger than 95% other methods are better.
  2. method = *neighbor_elimination* is a KD Tree based elimination method. For large tensions this scales as $O(n_{\text{samples}} \log(n_{\text{samples}}))$ and in worse case scenarions, with small tensions, this can scale as $O(n^2_{\text{samples}})$ but with significant overheads with respect to the brute force method. When expecting a statistically significant difference in parameters this is the recomended algorithm.

  Suggestion is to go with brute force for small problems, neighbor elimination for big problems with signifcant tensions. Default is *neighbor_elimination*.
- **feedback** – (optional) print to screen the time taken for the calculation.
- **kwargs** – extra options to pass to the KDE algorithm. The *neighbor_elimination* algorithm accepts the following optional arguments:
  1. stable_cycle: (default 2) number of elimination cycles that show no improvement in the result.
  2. chunk_size: (default 40) chunk size for elimination cycles. For best perfornamces this parameter should be tuned to result in the greatest elimination rates.
  3. smallest_improvement: (default 1.e-4) minimum percentage improvement rate before switching to brute force.
  4. near: (default 1) n-nearest neighbour to use for variable bandwidth KDE estimators.
  5. near_alpha: (default 1.0) scaling for nearest neighbour distance.

Returns probability value and error estimate from binomial.

Reference Raveri, Zacharegkas and Hu 19

`tensiometer.mcmc_tension.kde.`**`kde_parameter_shift_1D_fft`**(*diff_chain*, *param_names=None*, *scale=None*, *nbins=1024*, *feedback=1*, *boundary_correction_order=1*, *mult_bias_correction_order=1*, ***kwarks*)

Compute the MCMC estimate of the probability of a parameter shift given an input parameter difference chain in 1 dimension and by using FFT. This function uses GetDist 1D fft and optimal bandwidth estimates to perform the MCMC parameter shift integral discussed in (Raveri, Zacharegkas and Hu 19).

Parameters

- **diff_chain** – `MCSamples` input parameter difference chain

- **param_names** – (optional) parameter names of the parameters to be used in the calculation. By default all running parameters.
- **scale** – (optional) scale for the KDE smoothing. If none is provided the algorithm uses GetDist optimized bandwidth.
- **nbins** – (optional) number of 1D bins for the fft. Powers of 2 work best. Default is 1024.
- **mult_bias_correction_order** – (optional) multiplicative bias correction passed to GetDist. See `get2DDensity()`.
- **boundary_correction_order** – (optional) boundary correction passed to GetDist. See `get2DDensity()`.
- **feedback** – (optional) print to screen the time taken for the calculation.

Returns probability value and error estimate.

Reference Raveri, Zacharegkas and Hu 19

tensiometer.mcmc_tension.kde.**kde_parameter_shift_2D_fft**(*diff_chain*, *param_names=None*, *scale=None*, *nbins=1024*, *feedback=1*, *boundary_correction_order=1*, *mult_bias_correction_order=1*, ***kwarks*)

Compute the MCMC estimate of the probability of a parameter shift given an input parameter difference chain in 2 dimensions and by using FFT. This function uses GetDist 2D fft and optimal bandwidth estimates to perform the MCMC parameter shift integral discussed in (Raveri, Zacharegkas and Hu 19).

Parameters

- **diff_chain** – `MCSamples` input parameter difference chain
- **param_names** – (optional) parameter names of the parameters to be used in the calculation. By default all running parameters.
- **scale** – (optional) scale for the KDE smoothing. If none is provided the algorithm uses GetDist optimized bandwidth.
- **nbins** – (optional) number of 2D bins for the fft. Powers of 2 work best. Default is 1024.
- **mult_bias_correction_order** – (optional) multiplicative bias correction passed to GetDist. See `get2DDensity()`.
- **boundary_correction_order** – (optional) boundary correction passed to GetDist. See `get2DDensity()`.
- **feedback** – (optional) print to screen the time taken for the calculation.

Returns probability value and error estimate.

Reference Raveri, Zacharegkas and Hu 19

class tensiometer.mcmc_tension.flow.**DiffFlowCallback**(*diff_chain*, *param_names=None*, *Z2Y_bijector='MAF'*, *pregauss_bijector=None*, *learning_rate=0.001*, *feedback=1*, *validation_split=0.1*, *early_stop_nsigma=0.0*, *early_stop_patience=10*, ***kwargs*)

A class to compute the normalizing flow estimate of the probability of a parameter shift given an input parameter difference chain.

A normalizing flow is trained to approximate the difference distribution and then used to numerically evaluate the probablity of a parameter shift (see REF). To do so, it defines a bijective mapping that is optimized to gaussianize the difference chain samples. This mapping is performed in two steps, using the gaussian approximation as pre-whitening. The notations used in the code are:

- *X* designates samples in the original parameter difference space;
- *Y* designates samples in the gaussian approximation space, *Y* is obtained by shifting and scaling *X* by its mean and covariance (like a PCA);
- *Z* designates samples in the gaussianized space, connected to *Y* with a normalizing flow denoted *Z2Y_bijector*.

The user may provide the *Z2Y_bijector* as a `Bijector` object from Tensorflow Probability or make use of the utility class *`SimpleMAF`* to instantiate a Masked Autoregressive Flow (with *Z2Y_bijector='MAF'*).

This class derives from `Callback` from Keras, which allows for visualization during training. The normalizing flows (X->Y->Z) are implemented as `Bijector` objects and encapsulated in a Keras `Model`.

Here is an example:

```
# Initialize the flow and model
diff_flow_callback = DiffFlowCallback(diff_chain, Z2Y_bijector='MAF')
# Train the model
diff_flow_callback.train()
# Compute the shift probability and confidence interval
p, p_low, p_high = diff_flow_callback.estimate_shift_significance()
```

Parameters

- **diff_chain** (`MCSamples`) – input parameter difference chain.
- **param_names** (*`list, optional`*) – parameter names of the parameters to be used in the calculation. By default all running parameters.
- **Z2Y_bijector** (*`optional`*) – either a `Bijector` object representing the mapping from *Z* to *Y*, or 'MAF' to instantiate a *`SimpleMAF`*, defaults to 'MAF'.
- **pregauss_bijector** (*`optional`*) – not implemented yet, defaults to None.
- **learning_rate** (*`float, optional`*) – initial learning rate, defaults to 1e-3.
- **feedback** (*`int, optional`*) – feedback level, defaults to 1.
- **validation_split** (*`float, optional`*) – fraction of samples to use for the validation sample, defaults to 0.1
- **early_stop_nsigma** (*`float, optional`*) – absolute error on the tension at the zero-shift point to be used as an approximate convergence criterion for early stopping, defaults to 0.
- **early_stop_patience** (*`int, optional`*) – minimum number of epochs to use when *early_stop_nsigma* is non-zero, defaults to 10.

Raises **`NotImplementedError`** – if *pregauss_bijector* is not None.

Reference George Papamakarios, Theo Pavlakou, Iain Murray (2017). Masked Autoregressive Flow for Density Estimation. arXiv:1705.07057

estimate_shift(*tol=0.05*, *max_iter=1000*, *step=100000*)
Compute the normalizing flow estimate of the probability of a parameter shift given the input parameter difference chain. This is done with a Monte Carlo estimate by comparing the probability density at the zero-shift point to that at samples drawn from the normalizing flow approximation of the distribution.

Parameters

- **tol** (*float, optional*) – absolute tolerance on the shift significance, defaults to 0.05.
- **max_iter** (*int, optional*) – maximum number of sampling steps, defaults to 1000.
- **step** (*int, optional*) – number of samples per step, defaults to 100000.

Returns probability value and error estimate.

on_epoch_end(*epoch*, *logs={}*)
This method is used by Keras to show progress during training if *feedback* is True.

train(*epochs=100*, *batch_size=None*, *steps_per_epoch=None*, *callbacks=[]*, *verbose=1*, ***kwargs*)
Train the normalizing flow model. Internallay, this runs the fit method of the Keras `Model`, to which ***kwargs are passed*.

Parameters

- **epochs** (*int, optional*) – number of training epochs, defaults to 100.
- **batch_size** (*int, optional*) – number of samples per batch, defaults to None. If None, the training sample is divided into *steps_per_epoch* batches.
- **steps_per_epoch** (*int, optional*) – number of steps per epoch, defaults to None. If None and *batch_size* is also None, then *steps_per_epoch* is set to 100.
- **callbacks** (*list, optional*) – a list of additional Keras callbacks, such as `ReduceLROnPlateau`, defaults to [].
- **verbose** (*int, optional*) – verbosity level, defaults to 1.

Returns A `History` object. Its *history* attribute is a dictionary of training and validation loss values and metrics values at successive epochs: *"shift0_chi2"* is the squared norm of the zero-shift point in the gaussianized space, with the probability-to-exceed and corresponding tension in *"shift0_pval"* and *"shift0_nsigma"*; *"chi2Z_ks"* and *"chi2Z_ks_p"* contain the D_n statistic and probability-to-exceed of the Kolmogorov-Smironov test that squared norms of the transformed samples Z are χ^2 distributed (with a number of degrees of freedom equal to the number of parameters).

class tensiometer.mcmc_tension.flow.**SimpleMAF**(*num_params*, *n_maf=None*, *hidden_units=None*, *permutations=True*, *activation=<function asinh>*, *kernel_initializer='glorot_uniform'*, *feedback=0*, ***kwargs*)
A class to implement a simple Masked AutoRegressive Flow (MAF) using the implementation `tfp.bijectors.AutoregressiveNetwork` from from Tensorflow Probability. Additionally, this class provides utilities to load/save models, including random permutations.

Parameters

- **num_params** (*int*) – number of parameters, ie the dimension of the space of which the bijector is defined.
- **n_maf** (*int, optional*) – number of MAFs to stack. Defaults to None, in which case it is set to *2*num_params*.

- **hidden_units** (*list, optional*) – a list of the number of nodes per hidden layers. Defaults to None, in which case it is set to *[num_params*2]*2*.
- **permutations** (*bool, optional*) – whether to use shuffle dimensions between stacked MAFs, defaults to True.
- **activation** (*optional*) – activation function to use in all layers, defaults to `tf.math.asinh()`.
- **kernel_initializer** (*str, optional*) – kernel initializer, defaults to 'glorot_uniform'.
- **feedback** (*int, optional*) – print the model architecture, defaults to 0.

Reference George Papamakarios, Theo Pavlakou, Iain Murray (2017). Masked Autoregressive Flow for Density Estimation. arXiv:1705.07057

classmethod load(*num_params*, *path*, ***kwargs*)

Load a saved *SimpleMAF* object. The number of parameters and all other keyword arguments (except for *permutations*) must be included as the MAF is first created with random weights and then these weights are restored.

Parameters

- **num_params** (*int*) – number of parameters, ie the dimension of the space of which the bijector is defined.
- **path** (*str*) – path of the directory from which to load.

Returns a *SimpleMAF*.

save(*path*)

Save a *SimpleMAF* object.

Parameters **path** (*str*) – path of the directory where to save.

tensiometer.mcmc_tension.flow.**flow_parameter_shift**(*diff_chain*, *param_names=None*, *epochs=100*, *batch_size=None*, *steps_per_epoch=None*, *callbacks=[]*, *verbose=1*, *tol=0.05*, *max_iter=1000*, *step=100000*, ***kwargs*)

Wrapper function to compute a normalizing flow estimate of the probability of a parameter shift given the input parameter difference chain with a standard MAF. It creates a *DiffFlowCallback* object with a *SimpleMAF* model (to which kwargs are passed), trains the model and returns the estimated shift probability.

Parameters

- **diff_chain** (MCSamples) – input parameter difference chain.
- **param_names** (*list, optional*) – parameter names of the parameters to be used in the calculation. By default all running parameters.
- **epochs** (*int, optional*) – number of training epochs, defaults to 100.
- **batch_size** (*int, optional*) – number of samples per batch, defaults to None. If None, the training sample is divided into *steps_per_epoch* batches.
- **steps_per_epoch** (*int, optional*) – number of steps per epoch, defaults to None. If None and *batch_size* is also None, then *steps_per_epoch* is set to 100.
- **callbacks** (*list, optional*) – a list of additional Keras callbacks, such as ReduceLROnPlateau, defaults to [].
- **verbose** (*int, optional*) – verbosity level, defaults to 1.

- **tol** (*float, optional*) – absolute tolerance on the shift significance, defaults to 0.05.
- **max_iter** (*int, optional*) – maximum number of sampling steps, defaults to 1000.
- **step** (*int, optional*) – number of samples per step, defaults to 100000.

Returns probability value and error estimate.

CHAPTER 2

tensiometer.gaussian_tension

This file contains the functions and utilities to compute agreement and disagreement between two different chains using a Gaussian approximation for the posterior.

For more details on the method implemented see arxiv 1806.04649 and arxiv 1912.04880.

tensiometer.gaussian_tension.**KL_PCA**(*chain_1*, *chain_12*, *param_names=None*, *conditional_params=[]*, *param_map=None*, *normparam=None*, *num_modes=None*, *localize=True*, *dimensional_reduce=True*, *dimensional_threshold=0.1*, *verbose=True*, ***kwargs*)

Perform the KL analysis of two chains. Directions that chain_2 improves over chain_1.

Parameters

- **chain_1** – MCSamples the first input chain.
- **chain_12** – MCSamples the second input chain.
- **param_names** – list of names of the parameters to use
- **param_names** – (optional) parameter names to restrict the calculation. If none is given the default assumes that all running parameters
- **conditional_params** – (optional) list of parameters to treat as fixed, i.e. for KL_PCA conditional on fixed values of these parameters
- **param_map** – (optional) a transformation to apply to parameter values; A list or string containing either N (no transformation) or L (for log transform) or M (for minus log transform of negative parameters) for each parameter. By default uses log if no parameter values cross zero. The transformed parameters are added to the joint chain.
- **normparam** – (optional) name of parameter to normalize result (i.e. this parameter will have unit power) By default scales to the parameter that has the largest impactr on the KL mode variance.
- **num_modes** – (optional) only return the num_modes best modes.

- **localize** – (optional) localize the first covariance with the second, useful when chain_1 spans a much larger region with respect to chain_12.
- **dimensional_reduce** – (optional) perform dimensional reduction of the KL modes considered keeping only parameters with a large impact on KL mode variances. Default is True.
- **dimensional_threshold** – (optional) threshold for dimensional reducetion. Default is 10% so that parameters with a contribution less than 10% of KL mode variance are neglected from a specific KL mode.
- **verbose** – (optional) chatty output. Default True.

tensiometer.gaussian_tension.**Q_DM**(*chain_1*, *chain_2*, *prior_chain=None*, *param_names=None*, *cutoff=0.05*, *prior_factor=1.0*)

Compute the value and degrees of freedom of the quadratic form giving the probability of a difference between the means of the two input chains, in the Gaussian approximation.

This is defined as in (Raveri and Hu 18) to be:

$$Q_{\rm DM} \equiv (\theta_1 - \theta_2)(\mathcal{C}_1 + \mathcal{C}_2 - \mathcal{C}_1\mathcal{C}_{\Pi}^{-1}\mathcal{C}_2 - \mathcal{C}_2\mathcal{C}_{\Pi}^{-1}\mathcal{C}_1)^{-1}(\theta_1 - \theta_2)^T$$

where θ_i is the parameter mean of the i-th posterior, $\mathcal{C}$ the posterior covariance and $\mathcal{C}_{\Pi}$ the prior covariance. $Q_{\rm DM}$ is χ^2 distributed with number of degrees of freedom equal to the rank of the shift covariance.

Parameters

- **chain_1** – MCSamples the first input chain.
- **chain_2** – MCSamples the second input chain.
- **prior_chain** – (optional) the prior only chain. If the prior is not well approximated by a ranged prior and is informative it is better to explicitly use a prior only chain. If this is not given the algorithm will assume ranged priors with the ranges computed from the input chains.
- **param_names** – (optional) parameter names of the parameters to be used in the calculation. By default all running parameters.
- **cutoff** – (optional) the algorithms needs to detect prior constrained directions (that do not contribute to the test) from data constrained directions. This is achieved through a Karhunen–Loeve decomposition to avoid issues with physical dimensions of parameters and cutoff sets the minimum improvement with respect to the prior that is used. Default is five percent.
- **prior_factor** – (optional) factor to scale the prior covariance. In case of strongly non-Gaussian posteriors it might be useful to artificially tighten the prior to have less noise in telling apart parameter space directions that are constrained by data and prior. Default is no scaling, prior_factor=1.

Returns $Q_{\rm DM}$ value and number of degrees of freedom. Since $Q_{\rm DM}$ is χ^2 distributed the probability to exceed the test can be computed using the cdf method of scipy.stats.chi2 or *tensiometer.utilities.from_chi2_to_sigma()*.

tensiometer.gaussian_tension.**Q_DMAP**(*chain_1*, *chain_2*, *chain_12*, *prior_chain=None*, *param_names=None*, *prior_factor=1.0*, *feedback=True*)

Compute the value and degrees of freedom of the quadratic form giving the goodness of fit loss measure, in Gaussian approximation.

This is defined as in (Raveri and Hu 18) to be:

$$Q_{\rm DMAP} \equiv Q^{12}_{\rm MAP} - Q^{1}_{\rm MAP} - Q^{2}_{\rm MAP}$$

where $Q^{12}_{\rm MAP}$ is the joint likelihood at maximum posterior (MAP) and $Q^{i}_{\rm MAP}$ is the likelihood at MAP for the two single data sets. In Gaussian approximation this is distributed as:

$$Q_{\rm DMAP} \sim \chi^2(N^1_{\rm eff} + N^2_{\rm eff} - N^{12}_{\rm eff})$$

where $N_{\rm eff}$ is the number of effective parameters, as computed by the function *tensiometer.gaussian_tension.get_Neff()* for the joint and the two single data sets.

Parameters

- **chain_1** – MCSamples the first input chain.
- **chain_2** – MCSamples the second input chain.
- **chain_12** – MCSamples the joint input chain.
- **prior_chain** – (optional) the prior chain. If the prior is not well approximated by a ranged prior and is informative it is better to explicitly use a prior only chain. If this is not given the algorithm will assume ranged priors with the ranges computed from the input chain.
- **param_names** – (optional) parameter names of the parameters to be used in the calculation. By default all running parameters.
- **prior_factor** – (optional) factor to scale the prior covariance. In case of strongly non-Gaussian posteriors it might be useful to artificially tighten the prior to have less noise in telling apart parameter space directions that are constrained by data and prior. Default is no scaling, prior_factor=1.
- **feedback** – logical flag to set whether the function should print a warning every time the explicit MAP file is not found. By default this is true.

Returns $Q_{\rm DMAP}$ value and number of degrees of freedom. Since $Q_{\rm DMAP}$ is χ^2 distributed the probability to exceed the test can be computed using the cdf method of scipy.stats.chi2 or *tensiometer.utilities.from_chi2_to_sigma()*.

tensiometer.gaussian_tension.**Q_MAP**(*chain*, *num_data*, *prior_chain=None*, *normalization_factor=0.0*, *prior_factor=1.0*, *feedback=True*)

Compute the value and degrees of freedom of the quadratic form giving the goodness of fit measure at maximum posterior (MAP), in Gaussian approximation.

This is defined as in (Raveri and Hu 18) to be:

$$Q_{\rm MAP} \equiv -2\ln\mathcal{L}(\theta_{\rm MAP})$$

where $\mathcal{L}(\theta_{\rm MAP})$ is the data likelihood evaluated at MAP. In Gaussian approximation this is distributed as:

$$Q_{\rm MAP} \sim \chi^2(d - N_{\rm eff})$$

where d is the number of data points and $N_{\rm eff}$ is the number of effective parameters, as computed by the function *tensiometer.gaussian_tension.get_Neff()*.

Parameters

- **chain** – MCSamples the input chain.
- **num_data** – number of data points.
- **prior_chain** – (optional) the prior chain. If the prior is not well approximated by a ranged prior and is informative it is better to explicitly use a prior only chain. If this is not given the algorithm will assume ranged priors with the ranges computed from the input chain.

- **normalization_factor** – (optional) likelihood normalization factor. This should make the likelihood a chi square.
- **prior_factor** – (optional) factor to scale the prior covariance. In case of strongly non-Gaussian posteriors it might be useful to artificially tighten the prior to have less noise in telling apart parameter space directions that are constrained by data and prior. Default is no scaling, prior_factor=1.
- **feedback** – logical flag to set whether the function should print a warning every time the explicit MAP file is not found. By default this is true.

Returns $Q_{\rm MAP}$ value and number of degrees of freedom. Since $Q_{\rm MAP}$ is χ^2 distributed the probability to exceed the test can be computed using the cdf method of `scipy.stats.chi2` or *tensiometer.utilities.from_chi2_to_sigma()*.

tensiometer.gaussian_tension.**Q_UDM**(*chain_1*, *chain_12*, *lower_cutoff=1.05*, *upper_cutoff=100.0*, *param_names=None*)

Compute the value and degrees of freedom of the quadratic form giving the probability of a difference between the means of the two input chains, in update form with the Gaussian approximation.

This is defined as in (Raveri and Hu 18) to be:

$$Q_{\rm UDM} \equiv (\theta_1 - \theta_{12})(\mathcal{C}_1 - \mathcal{C}_{12})^{-1}(\theta_1 - \theta_{12})^T$$

where θ_1 is the parameter mean of the first posterior, θ_{12} is the parameter mean of the joint posterior, $\mathcal{C}$ the posterior covariance and $\mathcal{C}_\Pi$ the prior covariance. $Q_{\rm UDM}$ is χ^2 distributed with number of degrees of freedom equal to the rank of the shift covariance.

In case of uninformative priors the statistical significance of $Q_{\rm UDM}$ is the same as the one reported by $Q_{\rm DM}$ but offers likely mitigation against non-Gaussianities of the posterior distribution. In the case where both chains are Gaussian $Q_{\rm UDM}$ is symmetric if the first input chain is swapped $1 \leftrightarrow 2$. If the input distributions are not Gaussian it is better to use the most constraining chain as the base for the parameter update.

Parameters

- **chain_1** – `MCSamples` the first input chain.
- **chain_12** – `MCSamples` the joint input chain.
- **lower_cutoff** – (optional) the algorithms needs to detect prior constrained directions (that do not contribute to the test) from data constrained directions. This is achieved through a Karhunen–Loeve decomposition to avoid issues with physical dimensions of parameters and cutoff sets the minimum improvement with respect to the prior that is used. Default is five percent.
- **upper_cutoff** – (optional) upper cutoff for the selection of KL modes.
- **param_names** – (optional) parameter names of the parameters to be used in the calculation. By default all running parameters.

Returns $Q_{\rm UDM}$ value and number of degrees of freedom. Since $Q_{\rm UDM}$ is χ^2 distributed the probability to exceed the test can be computed using the cdf method of `scipy.stats.chi2` or *tensiometer.utilities.from_chi2_to_sigma()*.

tensiometer.gaussian_tension.**Q_UDM_KL_components**(*chain_1*, *chain_12*, *param_names=None*)

Function that computes the Karhunen–Loeve (KL) decomposition of the covariance of a chain with the covariance of that chain joint with another one. This function is used for the parameter shift algorithm in update form.

Parameters

- **chain_1** – `MCSamples` the first input chain.

- **chain_12** – MCSamples the joint input chain.
- **param_names** – (optional) parameter names of the parameters to be used in the calculation. By default all running parameters.

Returns the KL eigenvalues, the KL eigenvectors and the parameter names that are used, sorted in decreasing order.

tensiometer.gaussian_tension.**Q_UDM_covariance_components**(*chain_1*, *chain_12*, *param_names=None*, *which='1'*)

Compute the decomposition of the covariance matrix in terms of KL modes.

Parameters

- **chain_1** – MCSamples the first input chain.
- **chain_12** – MCSamples the joint input chain.
- **param_names** – (optional) parameter names of the parameters to be used in the calculation. By default all running parameters.
- **which** – (optional) which decomposition to report. Possibilities are '1' for the chain 1 covariance matrix, '2' for the chain 2 covariance matrix and '12' for the joint covariance matrix.

Returns parameter names used in the calculation, values of improvement, fractional covariance matrix and covariance matrix (inverse covariance).

tensiometer.gaussian_tension.**Q_UDM_fisher_components**(*chain_1*, *chain_12*, *param_names=None*, *which='1'*)

Compute the decomposition of the Fisher matrix in terms of KL modes.

Parameters

- **chain_1** – MCSamples the first input chain.
- **chain_12** – MCSamples the joint input chain.
- **param_names** – (optional) parameter names of the parameters to be used in the calculation. By default all running parameters.
- **which** – (optional) which decomposition to report. Possibilities are '1' for the chain 1 Fisher matrix, '2' for the chain 2 Fisher matrix and '12' for the joint Fisher matrix.

Returns parameter names used in the calculation, values of improvement and fractional Fisher matrix.

tensiometer.gaussian_tension.**Q_UDM_get_cutoff**(*chain_1*, *chain_2*, *chain_12*, *prior_chain=None*, *param_names=None*, *prior_factor=1.0*)

Function to estimate the cutoff for the spectrum of parameter differences in update form to match Delta Neff.

Parameters

- **chain_1** – MCSamples the first input chain.
- **chain_2** – MCSamples the second chain that joined with the first one (modulo the prior) should give the joint chain.
- **chain_12** – MCSamples the joint input chain.
- **prior_chain** – MCSamples (optional) If the prior is not well approximated by a ranged prior and is informative it is better to explicitly use a prior only chain. If this is not given the algorithm will assume ranged priors with the ranges computed from the input chain.

- **param_names** – (optional) parameter names of the parameters to be used in the calculation. By default all running parameters.
- **prior_factor** – (optional) factor to scale the prior covariance. In case of strongly non-Gaussian posteriors it might be useful to artificially tighten the prior to have less noise in telling apart parameter space directions that are constrained by data and prior. Default is no scaling, prior_factor=1.

Returns the optimal KL cutoff, KL eigenvalues, KL eigenvectors and the parameter names that are used.

tensiometer.gaussian_tension.**gaussian_approximation**(*chain*, *param_names=None*)

Function that computes the Gaussian approximation of a given chain.

Parameters

- **chain** – MCSamples the input chain.
- **param_names** – (optional) parameter names to restrict the Gaussian approximation. If none is given the default assumes that all parameters should be used.

Returns GaussianND object with the Gaussian approximation of the chain.

tensiometer.gaussian_tension.**get_MAP_loglike**(*chain*, *feedback=True*)

Utility function to obtain the data part of the maximum posterior for a given chain. The best possibility is that a separate file with the posterior explicit MAP is given. If this is not the case then the function will try to get the likelihood at MAP from the samples. This possibility is far more noisy in general.

Parameters

- **chain** – MCSamples the input chain.
- **feedback** – logical flag to set whether the function should print a warning every time the explicit MAP file is not found. By default this is true.

Returns the data log likelihood at maximum posterior.

tensiometer.gaussian_tension.**get_Neff**(*chain*, *prior_chain=None*, *param_names=None*, *prior_factor=1.0*, *localize=False*, ***kwargs*)

Function to compute the number of effective parameters constrained by a chain over the prior. The number of effective parameters is defined as in Eq. (29) of (Raveri and Hu 18) as:

$$N_{\rm eff} \equiv N - {\rm tr}[\mathcal{C}_\Pi^{-1}\mathcal{C}_p]$$

where N is the total number of nominal parameters of the chain, $\mathcal{C}_\Pi$ is the covariance of the prior and $\mathcal{C}_p$ is the posterior covariance.

Parameters

- **chain** – MCSamples the input chain.
- **prior_chain** – (optional) the prior chain. If the prior is not well approximated by a ranged prior and is informative it is better to explicitly use a prior only chain. If this is not given the algorithm will assume ranged priors with the ranges computed from the input chain.
- **param_names** – (optional) parameter names to restrict the calculation of $N_{\rm eff}$. If none is given the default assumes that all running parameters should be used.
- **prior_factor** – (optional) factor to scale the prior covariance. In case of strongly non-Gaussian posteriors it might be useful to artificially tighten the prior to have less noise in telling apart parameter space directions that are constrained by data and prior. Default is no scaling, prior_factor=1.

Returns the number of effective parameters.

tensiometer.gaussian_tension.**get_localized_covariance**(*chain_1*, *chain_2*, *param_names*, *localize_params=None*, *scale=10.0*)

Get localized covariance of chain_1 localized with chain_2

tensiometer.gaussian_tension.**get_prior_covariance**(*chain*, *param_names=None*)

Utility to estimate the prior covariance from the ranges of a chain. The flat range prior covariance (link) is given by:

$$C_{ij} = \delta_{ij} \frac{(max(p_i) - min(p_i))^2}{12}$$

Parameters

- **chain** – MCSamples the input chain.
- **param_names** – optional choice of parameter names to restrict the calculation.

Returns the estimated covariance of the prior.

CHAPTER 3

tensiometer.chains_convergence

This file contains some functions to study convergence of the chains and to compare the two posteriors.

tensiometer.chains_convergence.**GR_test**(*chains*, *param_names=None*)

Function performing the Gelman Rubin (GR) test (described in Gelman and Rubin 92 and Brooks and Gelman 98) on a list of MCSamples or on a single MCSamples with different sub-chains. This test compares the variation of the mean across a pool of chains with the expected variation of the mean under the pdf that is being sampled. If we define the covariance of the mean as:

$$C_{ij} \equiv \mathrm{Cov}_c(\mathrm{Mean}_s(\theta))_{ij}$$

and the mean covariance as:

$$M_{ij} \equiv \mathrm{Mean}_c[\mathrm{Cov}_s(\theta)_{ij}]$$

then we seek to maximize:

$$R - 1 = \mathrm{max}_\theta \frac{C_{ij}\theta^i\theta^j}{M_{ij}\theta^i\theta^j}$$

where the subscript c means that the statistics is computed across chains while the subscrit s indicates that it is computed across samples. In this case the maximization is solved by finding the maximum eigenvalue of CM^{-1}.

Parameters

- **chains** – single or list of MCSamples
- **param_names** – names of the parameters involved in the test. By default uses all non-derived parameters.

Returns value of the GR test and corresponding parameter combination

tensiometer.chains_convergence.**GR_test_from_samples**(*samples*, *weights*)

Lower level function to perform the Gelman Rubin (GR) test. This works on a list of samples from different chains and corresponding weights. Refer to *tensiometer.chains_convergence.GR_test()* for more details of what this function is doing.

Parameters

- **samples** – list of samples from different chains
- **weights** – weights of the samples for each chain

Returns value of the GR test and corresponding parameter combination

tensiometer.chains_convergence.**GRn_test**(*chains*, *n*, *theta0=None*, *param_names=None*, *feedback=0*, *optimizer='ParticleSwarm'*, ***kwargs*)

Multi dimensional higher order moments convergence test. Compares the variation of a given moment among the population of chains with the expected variation of that quantity from the samples pdf.

We first build the k order tensor of parameter differences around a point $\tilde{\theta}$:

$$Q^{(k)} \equiv Q_{i_1,\ldots,i_k} \equiv (\theta_{i_1} - \tilde{\theta}_{i_1}) \cdots (\theta_{i_k} - \tilde{\theta}_{i_k})$$

then we build the tensor encoding its covariance across chains

$$V_M = \mathrm{Var}_c(E_s[Q^{(k)}])$$

which is a $2k$ and then build the second tensor encoding the mean in chain moment:

$$M_V = \mathrm{Mean}_c(E_s[(Q^{(k)} - E_s[Q^{(k)}]) \otimes (Q^{(k)} - E_s[Q^{(k)}])])$$

where we have suppressed all indexes to not crowd the notation.

Then we maximize over parameters:

$$R_n - 1 \equiv \max_\theta \frac{V_M \theta^{2k}}{M_V \theta^{2k}}$$

where θ^{2k} is the tensor product of θ for $2k$ times.

Differently from the 2D case this problem has no solution in terms of eigenvalues of tensors so far and the solution is obtained by numerical minimization with the pymanopt library.

Parameters

- **chains** – single or list of MCSamples
- **n** – order of the moment
- **theta0** – center of the moments. By default equal to the mean
- **param_names** – names of the parameters involved in the test. By default uses all non-derived parameters.
- **feedback** – level of feedback. 0=no feedback, >0 increasingly chatty
- **optimizer** – choice of optimization algorithm for pymanopt. Default is ParticleSwarm, other possibility is TrustRegions.
- **kwargs** – keyword arguments for the optimizer.

Returns value of the GR moment test and corresponding parameter combination

tensiometer.chains_convergence.**GRn_test_1D**(*chains*, *n*, *param_name*, *theta0=None*)

One dimensional higher moments test. Compares the variation of a given moment among the population of chains with the expected variation of that quantity from the samples pdf.

This test is defined by:

$$R_n(\theta_0) - 1 = \frac{\mathrm{Var}_c(\mathrm{Mean}_s(\theta - \theta_0)^n)}{\mathrm{Mean}_c(\mathrm{Var}_s(\theta - \theta_0)^n)}$$

where the subscript c means that the statistics is computed across chains while the subscrit s indicates that it is computed across samples.

Parameters

- **chains** – single or list of MCSamples
- **n** – order of the moment
- **param_name** – names of the parameter involved in the test.
- **theta0** – center of the moments. By default equal to the mean.

Returns value of the GR moment test and corresponding parameter combination (an array with one since this works in 1D)

tensiometer.chains_convergence.**GRn_test_1D_samples**(*samples*, *weights*, *n*, *theta0=None*)

Lower level function to compute the one dimensional higher moments test. This works on a list of samples from different chains and corresponding weights. Refer to *tensiometer.chains_convergence.GRn_test_1D()* for more details of what this function is doing.

Parameters

- **samples** – list of samples from different chains
- **weights** – weights of the samples for each chain
- **n** – order of the moment
- **theta0** – center of the moments. By default equal to the mean.

Returns value of the GR moment test and corresponding parameter combination (an array with one since this works in 1D)

tensiometer.chains_convergence.**GRn_test_from_samples**(*samples*, *weights*, *n*, *theta0=None*, *feedback=0*, *optimizer='ParticleSwarm'*, ***kwargs*)

Lower level function to compute the multi dimensional higher moments test. This works on a list of samples from different chains and corresponding weights. Refer to *tensiometer.chains_convergence.GRn_test()* for more details of what this function is doing.

Parameters

- **samples** – list of samples from different chains
- **weights** – weights of the samples for each chain
- **n** – order of the moment
- **theta0** – center of the moments. By default equal to the mean
- **feedback** – level of feedback. 0=no feedback, >0 increasingly chatty
- **optimizer** – choice of optimization algorithm for pymanopt. Default is ParticleSwarm, other possibility is TrustRegions.
- **kwargs** – keyword arguments for the optimizer.

Returns value of the GR moment test and corresponding parameter combination

CHAPTER 4

tensiometer.cosmosis_interface

File with tools to interface Cosmosis chains with GetDist.

tensiometer.cosmosis_interface.**MCSamplesFromCosmosis**(*chain_root*, *chain_min_root=None*, *param_name_dict=None*, *param_label_dict=None*, *name_tag=None*, *settings=None*)

Function to import Cosmosis chains in GetDist.

Parameters

- **chain_root** – the name and path to the chain or the path to the folder that contains it.
- **chain_min_root** – (optional) name of the file containing the explicit best fit.
- **param_name_dict** – (optional) a dictionary with the mapping between cosmosis names and reasonable parameter names.
- **param_label_dict** – (optional) dictionary with the mapping between parameter names and parameter labels, since Cosmosis does not save the labels in the chain.
- **name_tag** – (optional) a string with the name tag for the chain.
- **settings** – (optional) dictionary of analysis settings to override getdist defaults

Returns The MCSamples instance

tensiometer.cosmosis_interface.**get_cosmosis_info**(*file*)

Parse a file to get all the information about a Cosmosis run.

Parameters **file** – path and name of the file to parse.

Returns a list of strings with the cosmosis parameters for the run.

tensiometer.cosmosis_interface.**get_maximum_likelihood**(*dummy*, *max_posterior*, *chain_min_root*, *param_name_dict*, *param_label_dict*)

Import the maximum likelihood file for a Cosmosis run, if present.

Parameters

- **dummy** – dummy argument for interfacing, not used in practice
- **chain_min_root** – name of the minimum file or the folder that contains it.
- **param_name_dict** – a dictionary with the mapping between cosmosis names and reasonable names.
- **param_label_dict** – dictionary with the mapping between the parameter names and the labels.

Returns BestFit the best fit object.

tensiometer.cosmosis_interface.**get_name_tag**(*info*)

Get the name tag for a chain given the a list of strings containing the cosmosis run parameter informations.

Parameters **info** – a list of strings with the cosmosis parameters for the run.

Returns a string with the name tag if any, otherwise returns none.

tensiometer.cosmosis_interface.**get_param_labels**(*info*, *param_names*, *param_label_dict*)

Get the labels for the parameter names of a Cosmosis run.

Parameters

- **info** – a list of strings with the cosmosis parameters for the run.
- **param_names** – a list of strings with the parameter names.
- **param_label_dict** – a dictionary with the mapping between names and labels.

Returns a list of strings with the parameter labels.

tensiometer.cosmosis_interface.**get_param_names**(*info*)

Get the parameter names for a Cosmosis run.

Parameters **info** – a list of strings with the cosmosis parameters for the run.

Returns a list of strings with the parameter names.

tensiometer.cosmosis_interface.**get_ranges**(*info*, *param_names*)

Get the ranges for the parameters from the info file.

Parameters

- **info** – a list of strings with the cosmosis parameters for the run.
- **param_names** – a list with the parameter names.

Returns a dictionary with the parameter ranges.

tensiometer.cosmosis_interface.**get_sampler_type**(*info*)

Get the sampler type for a chain given the a list of strings containing the cosmosis run parameter informations. To process the sampler type the function defines internally a dictionary with the mapping from sampler name to sampler type.

Parameters **info** – a list of strings with the cosmosis parameters for the run.

Returns a string with the sampler type if any, otherwise returns none.

tensiometer.cosmosis_interface.**polish_samples**(*chain*)

Remove fixed parameters and samples with some parameter that is Nan from the input chain.

Parameters **chain** – MCSamples the input chain.

Returns `MCSamples` the polished chain.

CHAPTER 5

tensiometer.utilities

This file contains some utilities that are used in the tensiometer package.

tensiometer.utilities.**KL_decomposition**(*matrix_a*, *matrix_b*)

Computes the Karhunen–Loeve (KL) decomposition of the matrix A and B.

Notice that B has to be real, symmetric and positive.

The algorithm is taken from this link. The algorithm is NOT optimized for speed but for precision.

Parameters

- **matrix_a** – the first matrix.
- **matrix_b** – the second matrix.

Returns the KL eigenvalues and the KL eigenvectors.

tensiometer.utilities.**PDM_to_vector**(*pdm*)

Transforms a positive definite matrix of dimension $d \times d$ into an unconstrained vector of dimension $d(d+1)/2$. This does not use the Cholesky decomposition since we need guarantee of strictly positive definiteness.

The absolute values of the elements with indexes of the returned vector that satisfy:

```
np.tril_indices(d, 0)[0] == np.tril_indices(d, 0)[1]
```

are the eigenvalues of the matrix. The sign of these elements define the orientation of the eigenvectors.

Note that this is not strictly the inverse of *tensiometer.utilities.vector_to_PDM()* since there are a number of discrete symmetries in the definition of the eigenvectors that we ignore since they are irrelevant for the sake of representing the matrix.

Parameters **pdm** – the input positive definite matrix.

Returns output vector representation.

Reference https://arxiv.org/abs/1906.00587

tensiometer.utilities.**QR_inverse**(*matrix*)

Invert a matrix with the QR decomposition. This is much slower than standard inversion but has better accuracy for matrices with higher condition number.

Parameters **matrix** – the input matrix.

Returns the inverse of the matrix.

tensiometer.utilities.**bernoulli_thin**(*chain*, *temperature=1*, *num_repeats=1*)

Function that thins a chain with a Bernoulli process.

Parameters

- **chain** – MCSamples the input chain.
- **temperature** – temperature of the Bernoulli process. If T=1 then this produces a unit weight chain.
- **num_repeats** – number of repetitions of the Bernoulli process.

Returns a MCSamples chain with the reweighted chain.

tensiometer.utilities.**clopper_pearson_binomial_trial**(*k*, *n*, *alpha=0.32*)

http://en.wikipedia.org/wiki/Binomial_proportion_confidence_interval alpha confidence intervals for a binomial distribution of k expected successes on n trials.

Parameters

- **k** – number of success.
- **n** – total number of trials.
- **alpha** – (optional) confidence level.

Returns lower and upper bound.

tensiometer.utilities.**from_chi2_to_sigma**(*val*, *dofs*, *exact_threshold=6*)

Computes the effective number of standard deviations for a chi squared variable. This matches the probability computed from the chi squared variable to the number of standard deviations that an event with the same probability would have had in a Gaussian distribution as in Eq. (G1) of (Raveri and Hu 18).

$$n_\sigma^{\rm eff}(x, {\rm dofs}) \equiv \sqrt{2}{\rm Erf}^{-1}({\rm CDF}(\chi^2_{\rm dofs}(x)))$$

For very high statistical significant events this function switches from the direct formula to an accurate asyntotic expansion.

Parameters

- **val** – value of the chi2 variable
- **dofs** – number of degrees of freedom of the chi2 variable
- **exact_threshold** – (default 6) threshold of value/dofs to switch to the asyntotic formula.

Returns the effective number of standard deviations.

tensiometer.utilities.**from_confidence_to_sigma**(*P*)

Transforms a probability to effective number of sigmas. This matches the input probability with the number of standard deviations that an event with the same probability would have had in a Gaussian distribution as in Eq. (G1) of (Raveri and Hu 18).

$$n_\sigma^{\rm eff}(P) \equiv \sqrt{2}{\rm Erf}^{-1}(P)$$

Parameters **P** – the input probability.

Returns the effective number of standard deviations.

tensiometer.utilities.**from_sigma_to_confidence**(*nsigma*)
Gives the probability of an event at a given number of standard deviations in a Gaussian distribution.

Parameters **nsigma** – the input number of standard deviations.

Returns the probability to exceed the number of standard deviations.

tensiometer.utilities.**get_separate_mcsamples**(*chain*)
Function that returns separate MCSamples for each sampler chain.

Parameters **chain** – MCSamples the input chain.

Returns list of MCSamples with the separate chains.

tensiometer.utilities.**is_outlier**(*points*, *thresh=3.5*)
Returns a boolean array with True if points are outliers and False otherwise.

Parameters

- **points** – An num-observations by num-dimensions array of observations
- **thresh** – The modified z-score to use as a threshold. Observations with a modified z-score (based on the median absolute deviation) greater than this value will be classified as outliers.

Returns A num-observations-length boolean array.

Reference Boris Iglewicz and David Hoaglin (1993), "Volume 16: How to Detect and Handle Outliers", The ASQC Basic References in Quality Control: Statistical Techniques, Edward F. Mykytka, Ph.D., Editor.

tensiometer.utilities.**make_list**(*elements*)
Checks if elements is a list. If yes returns elements without modifying it. If not creates and return a list with elements inside.

Parameters **elements** – an element or a list of elements.

Returns a list containing elements.

tensiometer.utilities.**min_samples_for_tension**(*nsigma*, *sigma_err*)
Computes the minimum number of uncorrelated samples that are needed to quantify a tension of a given significance with a given error through binomial trials.

This function works by inverting the Clopper Pearson binomial trial and likely delivers an underestimate of the points needed.

Parameters

- **nsigma** – number of effective sigmas of the given tension.
- **sigma_err** – the desired error on the determination of nsigma.

Returns minimum number of samples.

tensiometer.utilities.**random_samples_reshuffle**(*chain*)
Performs a coherent random reshuffle of the samples.

Parameters **chain** – MCSamples the input chain.

Returns a MCSamples chain with the reshuffled chain.

tensiometer.utilities.**vector_to_PDM**(*vec*)
Transforms an unconstrained vector of dimension $d(d+1)/2$ into a positive definite matrix of dimension $d \times d$. In the input vector the eigenvalues are in the positions where

The absolute values of the elements with indexes of the input vector that satisfy:

```
np.tril_indices(d, 0)[0] == np.tril_indices(d, 0)[1]
```

are the eigenvalues of the matrix. The sign of these elements define the orientation of the eigenvectors.

The purpose of this function is to allow optimization over the space of positive definite matrices that is either unconstrained or has constraints on the condition number of the matrix.

Parameters **pdm** – the input vector.

Returns output positive definite matrix.

Reference https://arxiv.org/abs/1906.00587

tensiometer.utilities.**whiten_samples**(*samples*, *weights*)

Rescales samples by the square root of their inverse covariance. The resulting samples have identity covariance. This amounts to a change of coordinates so the physical meaning of different coordinates is changed.

Parameters

- **samples** – the input samples.
- **weights** – the input weights of the samples.

Returns whitened samples with identity covariance.

Full documentation for the modules.

- genindex

Python Module Index

t

Index