TensionTools Documentation

Release 0.0.7

Marco Raveri

Aug 13, 2020

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 file contains the functions and utilities to compute agreement and disagreement between two different chains with Monte Carlo methods.

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

`tensiometer.mcmc_tension.`**`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`** –
- **`param_names`** –
- **`num_bins`** – number of bins used for the 1D estimate

Returns scaling vector for the whitened parameters

`tensiometer.mcmc_tension.`**`Scotts_RT`**`(`*num_params*, *num_samples*`)`

Compute Scott's rule of thumb bandwidth covariance scaling. 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 Scott's scaling.

`tensiometer.mcmc_tension.`**`Silvermans_RT`**`(`*num_params*, *num_samples*`)`

Compute Silverman's rule of thumb bandwidth covariance scaling. 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 Silverman's scaling.

`tensiometer.mcmc_tension.`**`exact_parameter_shift`**(*diff_chain*, *param_names=None*, *scale=None*, *method='brute_force'*, *feedback=1*, ***kwargs*)

Compute the MCMC 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 $O(n_{\rm samples}^2)$ and might require long run times. For this reason the algorithm is parallelized with the joblib library. To compute the KDE density estimate several methods are implemented.

In the 2D case this defaults to `exact_parameter_shift_2D_fft()` unless the kwarg use_fft is False.

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 Silverman's rule of thumb scaling.
- **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.
  - method = brute_force is a parallelized brute force method. This method scales as $O(n_{\rm samples}^2)$ and can be afforded only for small tensions. When suspecting a difference that is larger than 95% other methods are better.
  - method = nearest_elimination is a KD Tree based elimination method. For large tensions this scales as $O(n_{\rm samples}\log(n_{\rm samples}))$ and in worse case scenarions, with small tensions, this can scale as $O(n_{\rm samples}^2)$ 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, nearest elimination for big problems with signifcant tensions.
- **feedback** – (optional) print to screen the time taken for the calculation.
- **kwargs** – extra options to pass to the KDE algorithm. The nearest_elimination algorithm accepts the following optional arguments:
  - stable_cycle: (default 2) number of elimination cycles that show no improvement in the result.
  - chunk_size: (default 40) chunk size for elimination cycles. For best perfornamces this parameter should be tuned to result in the greatest elimination rates.
  - smallest_improvement: (default 1.e-4) minimum percentage improvement rate before switching to brute force.
  - use_fft: whether to use fft when possible.

Returns probability value and error estimate.

`tensiometer.mcmc_tension.`**`exact_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. 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.

`tensiometer.mcmc_tension.`**`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.`**`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.

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.`**`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:

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

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

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

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.

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

Compute the decomposition of the Fisher matrix for the first probe 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.

Returns parameter names used in the calculation, values of improvement and the 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*)

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_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 $\tilde{\theta}$:

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

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.parameter_reporting

This file contains some utilities to report parameter results in a nice way.

`tensiometer.parameter_reporting.`**`get_PJHPD_bounds`**(*chain*, *param_names*, *levels*)

Compute some estimate of the global ML confidence interval as described in https://arxiv.org/pdf/2007.01844.pdf

Parameters

- **`chain`** – `MCSamples` the input chain.
- **`param_names`** – optional choice of parameter names to restrict the calculation. Default is all parameters.
- **`levels`** – array with confidence levels to compute, i.e. [0.68, 0.95]

Returns an array with the bounds for each parameter.

`tensiometer.parameter_reporting.`**`get_best_fit_params`**(*chain*, *param_names*)

Utility to compute the parameter best fit. This tries to load the best fit from file and if it fails reports the best fit from samples (which could be noisy).

Parameters

- **`chain`** – `MCSamples` the input chain.
- **`param_names`** – optional choice of parameter names to restrict the calculation. Default is all parameters.

Returns an array with the best fit.

`tensiometer.parameter_reporting.`**`get_mean`**(*chain*, *param_names*)

Utility to compute the parameter mean. Mostly this is an utility to get the mean from parameter names rather than parameter indexes as we would do in GetDist.

Parameters

- **`chain`** – `MCSamples` the input chain.
- **`param_names`** – optional choice of parameter names to restrict the calculation. Default is all parameters.

Returns an array with the mean.

`tensiometer.parameter_reporting.`**`get_mode1d`**(*chain*, *param_names*)

Utility to compute the peak of the 1d posterior distribution for all parameters (parameter 1d mode). This depends and relies on the precomputed KDE smoothing so one can feed different analysis settings to change that.

Parameters

- **`chain`** – `MCSamples` the input chain.
- **`param_names`** – optional choice of parameter names to restrict the calculation. Default is all parameters.

Returns an array with the 1d mode.

`tensiometer.parameter_reporting.`**`parameter_table`**(*chain*, *param_names*, *use_peak=False*, *use_best_fit=True*, *use_PJHPD_bounds=False*, *ncol=1*, ***kwargs*)

Generate latex parameter table with summary results.

Parameters

- **`chain`** – `MCSamples` the input chain.
- **`param_names`** – optional choice of parameter names to restrict the calculation. Default is all parameters.
- **`use_peak`** – whether to use the peak of the 1d distribution instead of the mean. Default is False.
- **`use_best_fit`** – whether to include the best fit either from explicit minimization or sample. Default True.
- **`use_PJHPD_bounds`** – whether to report PJHPD bounds. Default False.
- **`ncol`** – number of columns for the table. Default 1.
- **`analysis_settings`** – optional analysis settings to use.
- **`kwargs`** – arguments for `ResultTable`

Returns a `ResultTable`

CHAPTER 5

tensiometer.cosmosis_interface

File with tools to interface Cosmosis chains with GetDist.

`tensiometer.cosmosis_interface.`**`MCSamplesFromCosmosis`**(*chain_root*, *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.
- **`param_label_dict`** – dictionary with the mapping between parameter names and parameter labels, since Cosmosis does not save the labels in the chain.
- **`name_tag`** – a string with the name tag for the chain.
- **`settings`** – dictionary of analysis settings to override 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`**(*chain_root*, *param_label_dict*)

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

Parameters

- **`chain_root`** – name of the chain file or the folder that contains it.
- **`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 6

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.`**`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.`**`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.`**`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.

Full documentation for the modules.

- genindex

Python Module Index

Index