
GPpy Documentation

Release

Author

October 03, 2014

1	The detailed Developers Documentation is listed below	3
1.1	GPy package	3
2	Indices and tables	117
	Python Module Index	119

For a quick start, you can have a look at one of the tutorials:

- Basic Gaussian process regression
- Interacting with models
- A kernel overview
- Writing new kernels
- Writing new models

You may also be interested by some examples in the GPpy/examples folder.

The detailed Developers Documentation is listed below

Contents:

1.1 GPy package

1.1.1 Subpackages

GPy.core package

Submodules

GPy.core.domains module

Created on 4 Jun 2013

@author: maxz

(Hyper-)Parameter domains defined for `priors` and `kern`. These domains specify the legitimate realm of the parameters to live in.

REAL : real domain, all values in the real numbers are allowed

POSITIVE: positive domain, only positive real values are allowed

NEGATIVE: same as **POSITIVE**, but only negative values are allowed

BOUNDED: only values within the bounded range are allowed, the bounds are specified withing the object with the bounded range

GPy.core.fitc module

class `GPy.core.fitc.FITC` (*X*, *likelihood*, *kernel*, *Z*, *normalize_X=False*)

Bases: `GPy.core.sparse_gp.SparseGP`

Sparse FITC approximation

Parameters

- **X** (*np.ndarray* (*num_data* *x* *Q*)) – inputs
- **likelihood** (*GPy.likelihood.(Gaussian | EP)*) – a likelihood instance, containing the observed data

- **kernel** (*a GPy.kern.kern instance*) – the kernel (covariance function). See link kernels
- **Z** (*np.ndarray (M x Q) | None*) – inducing inputs (optional, see note)
- **normalize_(X|Y)** (*bool*) – whether to normalize the data before computing (predictions will be in original scales)

dL_dZ ()

dL_dtheta ()

log_likelihood ()

Compute the (lower bound on the) log marginal likelihood

update_likelihood_approximation (***kwargs*)

Approximates a non-Gaussian likelihood using Expectation Propagation

For a Gaussian likelihood, no iteration is required: this function does nothing

GPy.core.gp module

class GPy.core.gp.**GP** (*X, likelihood, kernel, normalize_X=False*)

Bases: GPy.core.gp_base.GPBase

Gaussian Process model for regression and EP

Parameters

- **X** – input observations
- **kernel** – a GPy kernel, defaults to rbf+white
- **likelihood** – a GPy likelihood
- **normalize_X** (*False|True*) – whether to normalize the input data before computing (predictions will be in original scales)

Return type model object

Note: Multiple independent outputs are allowed using columns of Y

getstate ()

log_likelihood ()

The log marginal likelihood of the GP.

For an EP model, can be written as the log likelihood of a regression model for a new variable $Y^* = v_tilde/tau_tilde$, with a covariance matrix $K^* = K + diag(1./tau_tilde)$ plus a normalization term.

predict (*Xnew, which_parts='all', full_cov=False, **likelihood_args*)

Predict the function(s) at the new point(s) Xnew.

Parameters

- **Xnew** (*np.ndarray, Nnew x self.input_dim*) – The points at which to make a prediction
- **which_parts** (*('all', list of bools)*) – specifies which outputs kernel(s) to use in prediction
- **full_cov** (*bool*) – whether to return the full covariance matrix, or just the diagonal

Returns mean: posterior mean, a Numpy array, Nnew x self.input_dim

Returns var: posterior variance, a Numpy array, Nnew x 1 if full_cov=False, Nnew x Nnew otherwise

Returns

lower and upper boundaries of the 95% confidence intervals, Numpy arrays, Nnew x self.input_dim

If full_cov and self.input_dim > 1, the return shape of var is Nnew x Nnew x self.input_dim. If self.input_dim == 1, the return shape is Nnew x Nnew. This is to allow for different normalizations of the output dimensions.

predict_single_output (*Xnew*, *output=0*, *which_parts='all'*, *full_cov=False*, *likelihood_args={}*)

For a specific output, calls predict() at the new point(s) Xnew. This functions calls _add_output_index(), so Xnew should not have an index column specifying the output.

Parameters

- **Xnew** (*np.ndarray*, Nnew x self.input_dim) – The points at which to make a prediction
- **which_parts** ((*'all'*, list of bools)) – specifies which outputs kernel(s) to use in prediction
- **full_cov** (*bool*) – whether to return the full covariance matrix, or just the diagonal

Returns mean: posterior mean, a Numpy array, Nnew x self.input_dim

Returns var: posterior variance, a Numpy array, Nnew x 1 if full_cov=False, Nnew x Nnew otherwise

Returns lower and upper boundaries of the 95% confidence intervals, Numpy arrays, Nnew x self.input_dim

Note: For multiple non-independent outputs models only.

setstate (*state*)

update_likelihood_approximation (***kwargs*)

Approximates a non-gaussian likelihood using Expectation Propagation

For a Gaussian likelihood, no iteration is required: this function does nothing

GPy.core.gp_base module

class GPy.core.gp_base.**GPBase** (*X*, *likelihood*, *kernel*, *normalize_X=False*)

Bases: GPy.core.model.Model

Gaussian process base model for holding shared behaviour between sparse_GP and GP models, and potentially other models in the future.

Here we define some functions that are use

getstate ()

Get the current state of the class. This is only used to efficiently pickle the model. See also self.setstate

log_predictive_density (*x_test*, *y_test*)

Calculation of the log predictive density

Parameters

- **x_test** (*(Nx1) array*) – test observations (*x_{*}*)
- **y_test** (*(Nx1) array*) – test observations (*y_{*}*)

```
plot (plot_limits=None, which_data_rows='all', which_data_ycols='all', which_parts='all',
      fixed_inputs=[], levels=20, samples=0, fignum=None, ax=None, resolution=None,
      plot_raw=False, linecol='#204a87', fillcol='#729fcf')
```

Plot the posterior of the GP.

- In one dimension, the function is plotted with a shaded region identifying two standard deviations.
- In two dimensions, a contour-plot shows the mean predicted function
- In higher dimensions, use `fixed_inputs` to plot the GP with some of the inputs fixed.

Can plot only part of the data and part of the posterior functions using `which_data_rows`, `which_data_ycols` and `which_parts`

Parameters

- **plot_limits** (*np.array*) – The limits of the plot. If 1D [xmin,xmax], if 2D [[xmin,ymin],[xmax,ymax]]. Defaults to data limits
- **which_data_rows** (*'all' or a list of integers*) – which of the training data to plot (default all)
- **which_data_ycols** – when the data has several columns (independent outputs), only plot these
- **which_parts** (*'all', or list of bools*) – which of the kernel functions to plot (additively)
- **fixed_inputs** (*a list of tuples*) – a list of tuple [(i,v), (i,v)...], specifying that input index i should be set to value v.
- **resolution** (*int*) – the number of intervals to sample the GP on. Defaults to 200 in 1D and 50 (a 50x50 grid) in 2D
- **levels** (*int*) – number of levels to plot in a contour plot.
- **samples** (*int*) – the number of a posteriori samples to plot
- **fignum** (*figure number*) – figure to plot on.
- **ax** (*axes handle*) – axes to plot on.
- **linecol** – color of line to plot.
- **fillcol** – color of fill
- **levels** – for 2D plotting, the number of contour levels to use is ax is None, create a new figure

```
plot_f (*args, **kwargs)
```

Plot the GP's view of the world, where the data is normalized and before applying a likelihood.

This is a convenience function: we simply call `self.plot` with the argument `use_raw_predict` set True. All args and kwargs are passed on to `plot`.

see also: `gp_base.plot`

```
posterior_samples (X, size=10, which_parts='all', noise_model=None)
```

Samples the posterior GP at the points X.

Parameters

- **X** (*np.ndarray, Nnew x self.input_dim.*) – the points at which to take the samples.
- **size** (*int.*) – the number of a posteriori samples to plot.
- **which_parts** (*'all', or list of bools.*) – which of the kernel functions to plot (additively).

- **full_cov** (*bool.*) – whether to return the full covariance matrix, or just the diagonal.
- **noise_model** (*integer.*) – for mixed noise likelihood, the noise model to use in the samples.

Returns Ysim: set of simulations, a Numpy array (N x samples).

posterior_samples_f (*X, size=10, which_parts='all'*)

Samples the posterior GP at the points X.

Parameters

- **X** (*np.ndarray, Nnew x self.input_dim.*) – The points at which to take the samples.
- **size** (*int.*) – the number of a posteriori samples to plot.
- **which_parts** (*'all', or list of bools.*) – which of the kernel functions to plot (additively).
- **full_cov** (*bool.*) – whether to return the full covariance matrix, or just the diagonal.

Returns Ysim: set of simulations, a Numpy array (N x samples).

setstate (*state*)

Set the state of the model. Used for efficient pickling

GPy.core.mapping module

class GPy.core.mapping.**Mapping** (*input_dim, output_dim*)

Bases: GPy.core.parameterized.Parameterized

Base model for shared behavior between models that can act like a mapping.

df_dX (*dL_df, X*)

Evaluate derivatives of mapping outputs with respect to inputs.

Parameters

- **dL_df** (*ndarray (num_data x output_dim)*) – gradient of the objective with respect to the function.
- **X** (*ndarray (num_data x input_dim)*) – the input locations where derivatives are to be evaluated.

Returns matrix containing gradients of the function with respect to the inputs.

df_dtheta (*dL_df, X*)

The gradient of the outputs of the multi-layer perceptron with respect to each of the parameters. :param dL_df: gradient of the objective with respect to the function. :type dL_df: ndarray (num_data x output_dim) :param X: input locations where the function is evaluated. :type X: ndarray (num_data x input_dim) :returns: Matrix containing gradients with respect to parameters of each output for each input data. :rtype: ndarray (num_params length)

f (*X*)

plot (*plot_limits=None, which_data='all', which_parts='all', resolution=None, levels=20, samples=0, fignum=None, ax=None, fixed_inputs=[], linecol='#204a87'*)
Plot the mapping.

Plots the mapping associated with the model.

- In one dimension, the function is plotted.
- In two dimensions, a contour-plot shows the function
- In higher dimensions, we've not implemented this yet !TODO!

Can plot only part of the data and part of the posterior functions using `which_data` and `which_functions`

Parameters

- **plot_limits** (*np.array*) – The limits of the plot. If 1D [xmin,xmax], if 2D [[xmin,ymin],[xmax,ymax]]. Defaults to data limits
- **which_data** (*'all' or a slice object to slice self.X, self.Y*) – which if the training data to plot (default all)
- **which_parts** (*'all', or list of bools*) – which of the kernel functions to plot (additively)
- **resolution** (*int*) – the number of intervals to sample the GP on. Defaults to 200 in 1D and 50 (a 50x50 grid) in 2D
- **levels** (*int*) – number of levels to plot in a contour plot.
- **samples** (*int*) – the number of a posteriori samples to plot
- **fignum** (*figure number*) – figure to plot on.
- **ax** (*axes handle*) – axes to plot on.
- **fixed_inputs** (*a list of tuples*) – a list of tuple [(i,v), (i,v)...], specifying that input index i should be set to value v.
- **linecol** – color of line to plot.
- **levels** – for 2D plotting, the number of contour levels to use is ax is None, create a new figure

```
class GPy.core.mapping.Mapping_check_df_dX(mapping=None, dL_df=None, X=None)
```

Bases: `GPy.core.mapping.Mapping_check_model`

This class allows gradient checks for the gradient of a mapping with respect to X.

```
class GPy.core.mapping.Mapping_check_df_dtheta(mapping=None, dL_df=None, X=None)
```

Bases: `GPy.core.mapping.Mapping_check_model`

This class allows gradient checks for the gradient of a mapping with respect to parameters.

```
class GPy.core.mapping.Mapping_check_model(mapping=None, dL_df=None, X=None)
```

Bases: `GPy.core.model.Model`

This is a dummy model class used as a base class for checking that the gradients of a given mapping are implemented correctly. It enables `checkgradient()` to be called independently on each mapping.

```
log_likelihood()
```

GPy.core.model module

```
class GPy.core.model.Model
```

Bases: `GPy.core.parameterized.Parameterized`

```
Laplace_covariance()
```

return the covariance matrix of a Laplace approximation at the current (stationary) point.

```
Laplace_evidence()
```

Returns an estimate of the model evidence based on the Laplace approximation. Uses a numerical estimate of the Hessian if none is available analytically.

```
checkgrad(target_param=None, verbose=False, step=1e-06, tolerance=0.001)
```

Check the gradient of the model by comparing to a numerical estimate. If the verbose flag is passed, individual components are tested (and printed)

Parameters

- **verbose** (*bool*) – If True, print a “full” checking of each parameter
- **step** (*float (default 1e-6)*) – The size of the step around which to linearise the objective
- **tolerance** (*float (default 1e-3)*) – the tolerance allowed (see note)

Note:- The gradient is considered correct if the ratio of the analytical and numerical gradients is within <tolerance> of unity.

ensure_default_constraints ()

Ensure that any variables which should clearly be positive have been constrained somehow. The method performs a regular expression search on parameter names looking for the terms ‘variance’, ‘lengthscale’, ‘precision’ and ‘kappa’. If any of these terms are present in the name the parameter is constrained positive.

get_gradient (name, return_names=False)

Get model gradient(s) by name. The name is applied as a regular expression and all parameters that match that regular expression are returned.

Parameters

- **name** (*regular expression*) – the name of parameters required (as a regular expression).
- **return_names** (*bool*) – whether or not to return the names matched (default False)

getstate ()

Get the current state of the class. Inherited from Parameterized, so add those parameters to the state

Returns list of states from the model.

input_sensitivity ()

return an array describing the sensitivity of the model to each input

NB. Right now, we’re basing this on the lengthscales (or variances) of the kernel. TODO: proper sensitivity analysis where we integrate across the model inputs and evaluate the effect on the variance of the model output.

log_likelihood ()**log_prior ()**

evaluate the prior

objective_and_gradients (x)

Compute the objective function of the model and the gradient of the model at the point given by x.

Parameters **x** – the point at which gradients are to be computed.

objective_function (x)

The objective function passed to the optimizer. It combines the likelihood and the priors.

Failures are handled robustly. The algorithm will try several times to return the objective, and will raise the original exception if it the objective cannot be computed.

Parameters

- **x** – the parameters of the model.
- **type** – np.array

objective_function_gradients (x)

Gets the gradients from the likelihood and the priors.

Failures are handled robustly. The algorithm will try several times to return the gradients, and will raise the original exception if it the objective cannot be computed.

Parameters

- **x** – the parameters of the model.
- **type** – np.array

optimize (*optimizer=None, start=None, **kwargs*)

Optimize the model using self.log_likelihood and self.log_likelihood_gradient, as well as self.priors. kwargs are passed to the optimizer. They can be:

Parameters

- **max_f_eval** (*int*) – maximum number of function evaluations
- **optimizer** (*string TODO: valid strings?*) – which optimizer to use (defaults to self.preferred_optimizer)

Messages whether to display during optimisation

optimize_SGD (*momentum=0.1, learning_rate=0.01, iterations=20, **kwargs*)

optimize_restarts (*num_restarts=10, robust=False, verbose=True, parallel=False, num_processes=None, **kwargs*)

Perform random restarts of the model, and set the model to the best seen solution.

If the robust flag is set, exceptions raised during optimizations will be handled silently. If `_all_runs` fail, the model is reset to the existing parameter values.

Notes**Parameters**

- **num_restarts** (*int*) – number of restarts to use (default 10)
- **robust** (*bool*) – whether to handle exceptions silently or not (default False)
- **parallel** (*bool*) – whether to run each restart as a separate process. It relies on the multiprocessing module.
- **num_processes** – number of workers in the multiprocessing pool

**kwargs are passed to the optimizer. They can be:

Parameters

- **max_f_eval** (*int*) – maximum number of function evaluations
- **max_iters** (*int*) – maximum number of iterations
- **messages** (*bool*) – whether to display during optimisation

Note: If num_processes is None, the number of workers in the multiprocessing pool is automatically set to the number of processors on the current machine.

pseudo_EM (*stop_crit=0.1, **kwargs*)

EM - like algorithm for Expectation Propagation and Laplace approximation

Parameters **stop_crit** (*float*) – convergence criterion

randomize ()

Randomize the model. Make this draw from the prior if one exists, else draw from $N(0,1)$

set_prior (*regex, what*)

Sets priors on the model parameters.

Notes

Asserts that the prior is suitable for the constraint. If the wrong constraint is in place, an error is raised. If no constraint is in place, one is added (warning printed).

For tied parameters, the prior will only be “counted” once, thus a prior object is only inserted on the first tied index

Parameters

- **regex** – regular expression of parameters on which priors need to be set.
- **what** (*GPy.core.Prior type*) – prior to set on parameter.

setstate (*state*)

set state from previous call to getstate call Parameterized with the rest of the state

Parameters *state* (*list as returned from getstate.*) – the state of the model.

GPy.core.parameterized module

class GPy.core.parameterized.**Parameterized**

Bases: object

all_constrained_indices ()

constrain (*regex*, *transform*, *warning=True*)

constrain_bounded (*regex*, *lower*, *upper*, *warning=True*)

Set bounded constraints.

constrain_fixed (*regex*, *value=None*, *warning=True*)

Parameters

- **regex** (*ndarray(dtype=int) or regular expression object or string*) – which parameters need to be fixed.
- **value** (*float*) – the vlaue to fix the parameters to. If the value is not specified, the parameter is fixed to the current value

Notes

Fixing a parameter which is tied to another, or constrained in some way will result in an error.

To fix multiple parameters to the same value, simply pass a regular expression which matches both parameter names, or pass both of the indexes.

constrain_negative (*regex*, *warning=True*)

Set negative constraints.

constrain_positive (*regex*, *warning=True*)

Set positive constraints.

copy ()

Returns a (deep) copy of the current model

getstate ()

Get the current state of the class, here just all the indices, rest can get recomputed For inheriting from Parameterized:

Allways append the state of the inherited object and call down to the inherited object in setstate!!

grep_model (*regex*)

grep_param_names (*regex*, *transformed=False*, *search=False*)

Parameters `regexp` (*re* | *str* | *int*) – regular expression to select parameter names

Return type the indices of `self._get_param_names` which match the regular expression.

Note:- Other objects are passed through - i.e. integers which weren't meant for grepping

`num_params_transformed()`

`pickle` (*filename*, *protocol=-1*)

`setstate` (*state*)

`tie_params` (*regexp*)

Tie (all!) parameters matching the regular expression *regexp*.

`unconstrain` (*regexp*)

Unconstrain matching parameters. Does not untie parameters

`untie_everything()`

Unties all parameters by setting `tied_indices` to an empty list.

GPy.core.priors module

`class GPy.core.priors.Gamma(a, b)`

Bases: `GPy.core.priors.Prior`

Implementation of the Gamma probability function, coupled with random variables.

Parameters

- **a** – shape parameter
- **b** – rate parameter (warning: it's the *inverse* of the scale)

Note: Bishop 2006 notation is used throughout the code

`domain = 'positive'`

`static from_EV(E, V)`

Creates an instance of a Gamma Prior by specifying the Expected value(s) and Variance(s) of the distribution.

Parameters

- **E** – expected value
- **V** – variance

`lnpdf(x)`

`lnpdf_grad(x)`

`rvs(n)`

`summary()`

`class GPy.core.priors.Gaussian(mu, sigma)`

Bases: `GPy.core.priors.Prior`

Implementation of the univariate Gaussian probability function, coupled with random variables.

Parameters

- **mu** – mean

- **sigma** – standard deviation

Note: Bishop 2006 notation is used throughout the code

domain = 'real'

lnpdf (*x*)

lnpdf_grad (*x*)

rvs (*n*)

class GPy.core.priors.**LogGaussian** (*mu*, *sigma*)

Bases: GPy.core.priors.Prior

Implementation of the univariate *log*-Gaussian probability function, coupled with random variables.

Parameters

- **mu** – mean
- **sigma** – standard deviation

Note: Bishop 2006 notation is used throughout the code

domain = 'positive'

lnpdf (*x*)

lnpdf_grad (*x*)

rvs (*n*)

class GPy.core.priors.**MultivariateGaussian** (*mu*, *var*)

Implementation of the multivariate Gaussian probability function, coupled with random variables.

Parameters

- **mu** – mean (N-dimensional array)
- **var** – covariance matrix (NxN)

Note: Bishop 2006 notation is used throughout the code

domain = 'real'

lnpdf (*x*)

lnpdf_grad (*x*)

pdf (*x*)

plot ()

rvs (*n*)

summary ()

class GPy.core.priors.**Prior**

domain = None

pdf (*x*)

plot ()

`GPy.core.priors.gamma_from_EV(E, V)`

class `GPy.core.priors.inverse_gamma(a, b)`

Bases: `GPy.core.priors.Prior`

Implementation of the inverse-Gamma probability function, coupled with random variables.

Parameters

- **a** – shape parameter
- **b** – rate parameter (warning: it's the *inverse* of the scale)

Note: Bishop 2006 notation is used throughout the code

domain = 'positive'

lnpdf (*x*)

lnpdf_grad (*x*)

rvs (*n*)

GPy.core.sparse_gp module

class `GPy.core.sparse_gp.SparseGP(X, likelihood, kernel, Z, X_variance=None, normalize_X=False)`

Bases: `GPy.core.gp_base.GPBase`

Variational sparse GP model

Parameters

- **X** (*np.ndarray (num_data x input_dim)*) – inputs
- **likelihood** (*GPy.likelihood.(Gaussian | EP | Laplace)*) – a likelihood instance, containing the observed data
- **kernel** (*a GPy.kern.kern instance*) – the kernel (covariance function). See link kernels
- **X_variance** (*np.ndarray (num_data x input_dim) | None*) – The uncertainty in the measurements of X (Gaussian variance)
- **Z** (*np.ndarray (num_inducing x input_dim) | None*) – inducing inputs (optional, see note)
- **num_inducing** (*int*) – Number of inducing points (optional, default 10. Ignored if Z is not None)
- **normalize_X(Y)** (*bool*) – whether to normalize the data before computing (predictions will be in original scales)

dL_dZ ()

The derivative of the bound wrt the inducing inputs Z

dL_dtheta ()

Compute and return the derivative of the log marginal likelihood wrt the parameters of the kernel

getstate ()

Get the current state of the class, here just all the indices, rest can get recomputed

log_likelihood ()

Compute the (lower bound on the) log marginal likelihood

plot (*plot_limits=None*, *which_data_rows='all'*, *which_data_ycols='all'*, *which_parts='all'*, *fixed_inputs=[]*, *plot_raw=False*, *levels=20*, *samples=0*, *fignum=None*, *ax=None*, *resolution=None*)

Plot the posterior of the sparse GP.

- In one dimension, the function is plotted with a shaded region identifying two standard deviations.
- In two dimensions, a contour-plot shows the mean predicted function
- In higher dimensions, use *fixed_inputs* to plot the GP with some of the inputs fixed.

Can plot only part of the data and part of the posterior functions using *which_data_rows*, *which_data_ycols* and *which_parts*

Parameters

- **plot_limits** (*np.array*) – The limits of the plot. If 1D [*xmin*,*xmax*], if 2D [[*xmin*,*ymin*],[*xmax*,*ymax*]]. Defaults to data limits
- **which_data_rows** (*'all'* or a list of integers) – which of the training data to plot (default all)
- **which_data_ycols** – when the data has several columns (independent outputs), only plot these
- **which_parts** (*'all'*, or list of bools) – which of the kernel functions to plot (additively)
- **fixed_inputs** (a list of tuples) – a list of tuple [(*i*,*v*), (*i*,*v*)...], specifying that input index *i* should be set to value *v*.
- **resolution** (*int*) – the number of intervals to sample the GP on. Defaults to 200 in 1D and 50 (a 50x50 grid) in 2D
- **levels** (*int*) – number of levels to plot in a contour plot.
- **samples** (*int*) – the number of a posteriori samples to plot
- **fignum** (*figure number*) – figure to plot on.
- **ax** (*axes handle*) – axes to plot on.
- **linecol** – color of line to plot.
- **fillcol** – color of fill
- **levels** – for 2D plotting, the number of contour levels to use is *ax* is None, create a new figure

plot_f (*samples=0*, *plot_limits=None*, *which_data_rows='all'*, *which_data_ycols='all'*, *which_parts='all'*, *resolution=None*, *full_cov=False*, *fignum=None*, *ax=None*)

Plot the GP's view of the world, where the data is normalized and the

- In one dimension, the function is plotted with a shaded region identifying two standard deviations.
- In two dimensions, a contour-plot shows the mean predicted function
- Not implemented in higher dimensions

Parameters

- **samples** – the number of a posteriori samples to plot
- **plot_limits** – The limits of the plot. If 1D [*xmin*,*xmax*], if 2D [[*xmin*,*ymin*],[*xmax*,*ymax*]]. Defaults to data limits

- **which_data_rows** (*'all'* or a slice object to slice *self.X*, *self.Y*) – which if the training data to plot (default all)
- **which_parts** (*'all'*, or list of bools) – which of the kernel functions to plot (additively)
- **resolution** (*int*) – the number of intervals to sample the GP on. Defaults to 200 in 1D and 50 (a 50x50 grid) in 2D
- **full_cov** (*bool* :param fignum: figure to plot on.) –
- **ax** (*axes handle*) – axes to plot on.
- **output** (*integer* (first output is 0)) – which output to plot (for multiple output models only)

predict (*Xnew*, *X_variance_new=None*, *which_parts='all'*, *full_cov=False*, ***likelihood_args*)

Predict the function(s) at the new point(s) *Xnew*.

Arguments

Parameters

- **Xnew** (*np.ndarray*, *Nnew x self.input_dim*) – The points at which to make a prediction
- **X_variance_new** (*np.ndarray*, *Nnew x self.input_dim*) – The uncertainty in the prediction points
- **which_parts** (*(('all', list of bools))*) – specifies which outputs kernel(s) to use in prediction
- **full_cov** (*bool*) – whether to return the full covariance matrix, or just the diagonal

Return type posterior mean, a Numpy array, *Nnew x self.input_dim*

Return type posterior variance, a Numpy array, *Nnew x 1* if *full_cov=False*, *Nnew x Nnew* otherwise

Return type

lower and upper boundaries of the 95% confidence intervals, Numpy arrays, *Nnew x self.input_dim*

If *full_cov* and *self.input_dim > 1*, the return shape of var is *Nnew x Nnew x self.input_dim*. If *self.input_dim == 1*, the return shape is *Nnew x Nnew*. This is to allow for different normalizations of the output dimensions.

setstate (*state*)

update_likelihood_approximation (***kwargs*)

Approximates a non-gaussian likelihood using Expectation Propagation

For a Gaussian likelihood, no iteration is required: this function does nothing

GPy.core.svgp module

class `GPy.core.svgp.SVIGP` (*X*, *likelihood*, *kernel*, *Z*, *q_u=None*, *batchsize=10*, *X_variance=None*)

Bases: `GPy.core.gp_base.GPBase`

Stochastic Variational inference in a Gaussian Process

Parameters

- **X** (*np.ndarray* (*num_data x num_inputs*)) – inputs
- **Y** (*np.ndarray* of observations (*num_data x output_dim*)) – observed data
- **batchsize** – the size of a minibatch

- **q_u** (*np.ndarray*) – canonical parameters of the distribution squashed into a 1D array
- **kernel** (*a GPy kernel*) – the kernel/covariance function. See link kernels
- **Z** (*np.ndarray (num_inducing x num_inputs)*) – inducing inputs

dL_dtheta ()

get_vb_param ()

Return the canonical parameters of the distribution $q(u)$

getstate ()

load_batch ()

load a batch of data (set self.X_batch and self.likelihood.Y from self.X, self.Y)

log_likelihood ()

As for uncollapsed sparse GP, but account for the proportion of data we're looking at right now.

NB. self.batchsize is the size of the batch, not the size of X_all

optimize (*iterations, print_interval=10, callback=<function <lambda> at 0x7f37af1867d0>, callback_interval=5*)

plot (*ax=None, fignum=None, Z_height=None, **kwargs*)

plot_traces ()

predict (*Xnew, X_variance_new=None, which_parts='all', full_cov=False, sampling=False, num_samples=15000*)

set_vb_param (*vb_param*)

set the distribution $q(u)$ from the canonical parameters

setstate (*state*)

vb_grad_natgrad ()

Compute the gradients of the lower bound wrt the canonical and Expectation parameters of u .

Note that the natural gradient in either is given by the gradient in the other (See Hensman et al 2012 Fast Variational inference in the conjugate exponential Family)

GPy.core.transformations module

class GPy.core.transformations.**exponent**

Bases: GPy.core.transformations.transformation

domain = 'positive'

f (*x*)

finv (*x*)

gradfactor (*f*)

initialize (*f*)

class GPy.core.transformations.**logexp**

Bases: GPy.core.transformations.transformation

domain = 'positive'

f (*x*)

finv (*f*)

```
    gradfactor (f)
    initialize (f)
class GPy.core.transformations.logexp_clipped (lower=1e-06)
    Bases: GPy.core.transformations.logexp
    domain = 'positive'
    f (x)
    finv (f)
    gradfactor (f)
    initialize (f)
    log_max_bound = 230.25850929940458
    log_min_bound = -23.025850929940457
    max_bound = 1e+100
    min_bound = 1e-10
class GPy.core.transformations.logistic (lower, upper)
    Bases: GPy.core.transformations.transformation
    domain = 'bounded'
    f (x)
    finv (f)
    gradfactor (f)
    initialize (f)
class GPy.core.transformations.negative_exponent
    Bases: GPy.core.transformations.exponent
    domain = 'negative'
    f (x)
    finv (f)
    gradfactor (f)
    initialize (f)
class GPy.core.transformations.negative_logexp
    Bases: GPy.core.transformations.transformation
    domain = 'negative'
    f (x)
    finv (f)
    gradfactor (f)
    initialize (f)
class GPy.core.transformations.square
    Bases: GPy.core.transformations.transformation
    domain = 'positive'
    f (x)
```

```

finv(x)

gradfactor(f)

initialize(f)

class GPy.core.transformations.transformation
    Bases: object

    domain = None

    f(x)

    finv(x)

    gradfactor(f)
        df_dx evaluated at self.f(x)=f

    initialize(f)
        produce a sensible initial value for f(x)

```

Module contents

GPy.examples package

Submodules

GPy.examples.classification module

Gaussian Processes classification

```

GPy.examples.classification.crescent_data(model_type='Full',          num_inducing=10,
                                           seed=10000, kernel=None, optimize=True,
                                           plot=True)

```

Run a Gaussian process classification on the crescent data. The demonstration calls the basic GP classification model and uses EP to approximate the likelihood.

Parameters

- **model_type** – type of model to fit ['Full', 'FITC', 'DTC'].
- **inducing** (*int*) – number of inducing variables (only used for 'FITC' or 'DTC').
- **seed** (*int*) – seed value for data generation.
- **kernel** (*a GPy kernel*) – kernel to use in the model

```

GPy.examples.classification.oil(num_inducing=50, max_iters=100, kernel=None, opti-
                                mize=True, plot=True)

```

Run a Gaussian process classification on the three phase oil data. The demonstration calls the basic GP classification model and uses EP to approximate the likelihood.

```

GPy.examples.classification.sparse_toy_linear_1d_classification(num_inducing=10,
                                                                seed=10000,
                                                                opti-
                                                                mize=True,
                                                                plot=True)

```

Sparse 1D classification example

Parameters **seed** (*int*) – seed value for data generation (default is 4).

`GPy.examples.classification.toy_heaviside` (*seed=10000, optimize=True, plot=True*)

Simple 1D classification example using a heavy side gp transformation

Parameters *seed* (*int*) – seed value for data generation (default is 4).

`GPy.examples.classification.toy_linear_1d_classification` (*seed=10000, optimize=True, plot=True*)

Simple 1D classification example using EP approximation

Parameters *seed* (*int*) – seed value for data generation (default is 4).

`GPy.examples.classification.toy_linear_1d_classification_laplace` (*seed=10000, optimize=True, plot=True*)

Simple 1D classification example using Laplace approximation

Parameters *seed* (*int*) – seed value for data generation (default is 4).

GPy.examples.dimensionality_reduction module

`GPy.examples.dimensionality_reduction.bcgplvm_linear_stick` (*kernel=None, optimize=True, verbose=True, plot=True*)

`GPy.examples.dimensionality_reduction.bcgplvm_stick` (*kernel=None, optimize=True, verbose=True, plot=True*)

`GPy.examples.dimensionality_reduction.bgplvm_oil` (*optimize=True, verbose=1, plot=True, N=200, Q=7, num_inducing=40, max_iters=1000, **k*)

`GPy.examples.dimensionality_reduction.bgplvm_simulation` (*optimize=True, verbose=1, plot=True, plot_sim=False, max_iters=20000.0*)

`GPy.examples.dimensionality_reduction.bgplvm_test_model` (*seed=None, optimize=False, verbose=1, plot=False*)

model for testing purposes. Samples from a GP with rbf kernel and learns the samples with a new kernel. Normally not for optimization, just model cheking

`GPy.examples.dimensionality_reduction.brendan_faces` (*optimize=True, verbose=True, plot=True*)

`GPy.examples.dimensionality_reduction.cmu_mocap` (*subject='35', motion=['01'], in_place=True, optimize=True, verbose=True, plot=True*)

`GPy.examples.dimensionality_reduction.gplvm_oil_100` (*optimize=True, verbose=1, plot=True*)

`GPy.examples.dimensionality_reduction.mrd_simulation` (*optimize=True, verbose=True, plot=True, plot_sim=True, **kw*)

`GPy.examples.dimensionality_reduction.olivetti_faces` (*optimize=True, verbose=True, plot=True*)


```

GPy.examples.dimensionality_reduction.robot_wireless (optimize=True, verbose=True,
                                                       plot=True)
GPy.examples.dimensionality_reduction.sparse_gplvm_oil (optimize=True,          ver-
                                                       bose=0, plot=True, N=100,
                                                       Q=6,   num_inducing=15,
                                                       max_iters=50)
GPy.examples.dimensionality_reduction.stick (kernel=None, optimize=True, verbose=True,
                                              plot=True)
GPy.examples.dimensionality_reduction.stick_bgplvm (model=None, optimize=True, ver-
                                                    bose=True, plot=True)
GPy.examples.dimensionality_reduction.stick_play (range=None,          frame_rate=15,
                                                    optimize=False,    verbose=True,
                                                    plot=True)
GPy.examples.dimensionality_reduction.swiss_roll (optimize=True,          verbose=1,
                                                    plot=True,          N=1000,
                                                    num_inducing=15,    Q=4,
                                                    sigma=0.2)

```

GPy.examples.non_gaussian module

```

GPy.examples.non_gaussian.boston_example (optimize=True, plot=True)
GPy.examples.non_gaussian.student_t_approx (optimize=True, plot=True)
    Example of regressing with a student t likelihood using Laplace

```

GPy.examples.regression module

Gaussian Processes regression examples

```

GPy.examples.regression.coregionalization_sparse (optimize=True, plot=True)
    A simple demonstration of coregionalization on two sinusoidal functions using sparse approximations.
GPy.examples.regression.coregionalization_toy2 (optimize=True, plot=True)
    A simple demonstration of coregionalization on two sinusoidal functions.
GPy.examples.regression.epomeo_gpx (max_iters=200, optimize=True, plot=True)
    Perform Gaussian process regression on the latitude and longitude data from the Mount Epomeo runs. Requires
    gpxpy to be installed on your system to load in the data.
GPy.examples.regression.multiple_optima (gene_number=937,          resolution=80,
                                         model_restarts=10, seed=10000, max_iters=300,
                                         optimize=True, plot=True)
    Show an example of a multimodal error surface for Gaussian process regression. Gene 939 has bimodal be-
    haviour where the noisy mode is higher.
GPy.examples.regression.olympic_100m_men (optimize=True, plot=True)
    Run a standard Gaussian process regression on the Rogers and Girolami olympics data.
GPy.examples.regression.olympic_marathon_men (optimize=True, plot=True)
    Run a standard Gaussian process regression on the Olympic marathon data.
GPy.examples.regression.robot_wireless (max_iters=100,   kernel=None,   optimize=True,
                                         plot=True)
    Predict the location of a robot given wirelss signal strength readings.

```

```
GPpy.examples.regression.silhouette(max_iters=100, optimize=True, plot=True)
```

Predict the pose of a figure given a silhouette. This is a task from Agarwal and Triggs 2004 ICML paper.

```
GPpy.examples.regression.sparse_GP_regression_1D(num_samples=400, num_inducing=5,
                                                  max_iters=100, optimize=True,
                                                  plot=True)
```

Run a 1D example of a sparse GP regression.

```
GPpy.examples.regression.sparse_GP_regression_2D(num_samples=400,
                                                  num_inducing=50, max_iters=100,
                                                  optimize=True, plot=True)
```

Run a 2D example of a sparse GP regression.

```
GPpy.examples.regression.toy_ARD(max_iters=1000, kernel_type='linear', num_samples=300,
                                  D=4, optimize=True, plot=True)
```

```
GPpy.examples.regression.toy_ARD_sparse(max_iters=1000, kernel_type='linear',
                                          num_samples=300, D=4, optimize=True,
                                          plot=True)
```

```
GPpy.examples.regression.toy_poisson_rbf_1d_laplace(optimize=True, plot=True)
```

Run a simple demonstration of a standard Gaussian process fitting it to data sampled from an RBF covariance.

```
GPpy.examples.regression.toy_rbf_1d(optimize=True, plot=True)
```

Run a simple demonstration of a standard Gaussian process fitting it to data sampled from an RBF covariance.

```
GPpy.examples.regression.toy_rbf_1d_50(optimize=True, plot=True)
```

Run a simple demonstration of a standard Gaussian process fitting it to data sampled from an RBF covariance.

```
GPpy.examples.regression.uncertain_inputs_sparse_regression(max_iters=200,
                                                             optimize=True,
                                                             plot=True)
```

Run a 1D example of a sparse GP regression with uncertain inputs.

GPpy.examples.stochastic module

```
GPpy.examples.stochastic.toy_1d(optimize=True, plot=True)
```

GPpy.examples.tutorials module

Code of Tutorials

```
GPpy.examples.tutorials.model_interaction(optimize=True, plot=True)
```

```
GPpy.examples.tutorials.tuto_GP_regression(optimize=True, plot=True)
```

The detailed explanations of the commands used in this file can be found in the tutorial section

```
GPpy.examples.tutorials.tuto_kernel_overview(optimize=True, plot=True)
```

The detailed explanations of the commands used in this file can be found in the tutorial section

Module contents

GPy.inference package

Submodules

GPy.inference.conjugate_gradient_descent module

Created on 24 Apr 2013

@author: maxz

```

class GPy.inference.conjugate_gradient_descent.AsyncOptimize
    Bases: object

    SENTINEL = 'SENTINEL'

    async_callback_collect(q)

    callback(*x)

    opt(f, df, x0, callback=None, update_rule=<class GPy.inference.gradient_descent_update_rules.FletcherReeves
        at 0x7f37ad8317a0>, messages=0, maxiter=5000.0, max_f_eval=15000.0, gtol=1e-06, re-
        port_every=10, *args, **kwargs)

    opt_async(f, df, x0, callback, update_rule=<class GPy.inference.gradient_descent_update_rules.PolakRibiere
        at 0x7f37ad831738>, messages=0, maxiter=5000.0, max_f_eval=15000.0, gtol=1e-06,
        report_every=10, *args, **kwargs)

    runsignal = <multiprocessing.synchronize.Event object at 0x7f37ae075450>

class GPy.inference.conjugate_gradient_descent.CGD
    Bases: GPy.inference.conjugate_gradient_descent.AsyncOptimize

    Conjugate gradient descent algorithm to minimize function f with gradients df, starting at x0 with update rule
    update_rule

    if df returns tuple (grad, natgrad) it will optimize according to natural gradient rules

    opt(*a, **kw)

        opt(self, f, df, x0, callback=None, update_rule=FletcherReeves, messages=0, maxiter=5e3,
            max_f_eval=15e3, gtol=1e-6, report_every=10, *args, **kwargs)

    Minimize f, calling callback every report_every iterations with following syntax:

        callback(xi, fi, gi, iteration, function_calls, gradient_calls, status_message)

    if df returns tuple (grad, natgrad) it will optimize according to natural gradient rules
    f, and df will be called with

        f(xi, *args, **kwargs) df(xi, *args, **kwargs)

    returns

        x_opt, f_opt, g_opt, iteration, function_calls, gradient_calls, status_message

    at end of optimization

    opt_async(*a, **kw)

        opt_async(self, f, df, x0, callback, update_rule=FletcherReeves, messages=0, maxiter=5e3,
            max_f_eval=15e3, gtol=1e-6, report_every=10, *args, **kwargs)

```

callback gets called every *report_every* iterations

callback(xi, fi, gi, iteration, function_calls, gradient_calls, status_message)

if df returns tuple (grad, natgrad) it will optimize according to natural gradient rules

f, and df will be called with

f(xi, *args, **kwargs) df(xi, *args, **kwargs)

Returns:

Started *Process* object, optimizing asynchronously

Calls:

callback(x_opt, f_opt, g_opt, iteration, function_calls, gradient_calls, status_message)

at end of optimization!

opt_name = 'Conjugate Gradient Descent'

GPy.inference.gradient_descent_update_rules module

Created on 24 Apr 2013

@author: maxz

```
class GPy.inference.gradient_descent_update_rules.FletcherReeves (initgrad, initgradnat=None)
```

Bases: GPy.inference.gradient_descent_update_rules.GDUpdateRule

Fletcher Reeves update rule for gamma

```
class GPy.inference.gradient_descent_update_rules.GDUpdateRule (initgrad, initgradnat=None)
```

```
class GPy.inference.gradient_descent_update_rules.PolakRibiere (initgrad, initgradnat=None)
```

Bases: GPy.inference.gradient_descent_update_rules.GDUpdateRule

Fletcher Reeves update rule for gamma

GPy.inference.optimization module

```
class GPy.inference.optimization.Optimizer (x_init, messages=False, model=None, max_f_eval=10000.0, max_iters=1000.0, ftol=None, gtol=None, xtol=None, bfgs_factor=None)
```

Superclass for all the optimizers.

Parameters

- **x_init** – initial set of parameters
- **f_fp** – function that returns the function AND the gradients at the same time
- **f** – function to optimize
- **fp** – gradients
- **messages** ((*True* | *False*)) – print messages from the optimizer?
- **max_f_eval** – maximum number of function evaluations

Return type optimizer object.

opt (*f_fp=None, f=None, fp=None*)

plot ()

run (***kwargs*)

GPy.inference.optimization.**get_optimizer** (*f_min*)

class GPy.inference.optimization.**opt_SCG** (**args, **kwargs*)

Bases: GPy.inference.optimization.Optimizer

opt (*f_fp=None, f=None, fp=None*)

class GPy.inference.optimization.**opt_lbfgsb** (**args, **kwargs*)

Bases: GPy.inference.optimization.Optimizer

opt (*f_fp=None, f=None, fp=None*)

Run the optimizer

class GPy.inference.optimization.**opt_rasm** (**args, **kwargs*)

Bases: GPy.inference.optimization.Optimizer

opt (*f_fp=None, f=None, fp=None*)

Run Rasmussen's Conjugate Gradient optimizer

class GPy.inference.optimization.**opt_simplex** (**args, **kwargs*)

Bases: GPy.inference.optimization.Optimizer

opt (*f_fp=None, f=None, fp=None*)

The simplex optimizer does not require gradients.

class GPy.inference.optimization.**opt_tnc** (**args, **kwargs*)

Bases: GPy.inference.optimization.Optimizer

opt (*f_fp=None, f=None, fp=None*)

Run the TNC optimizer

GPy.inference.samplers module

class GPy.inference.samplers.**Metropolis_Hastings** (*model, cov=None*)

new_chain (*start=None*)

predict (*function, args*)

Make a prediction for the function, to which we will pass the additional arguments

sample (*Ntotal, Nburn, Nthin, tune=True, tune_throughout=False, tune_interval=400*)

GPy.inference.scg module

GPy.inference.scg.**SCG** (*f, gradf, x, optargs=(), maxiters=500, max_f_eval=inf, display=True, xtol=None, ftol=None, gtol=None*)

Optimisation through Scaled Conjugate Gradients (SCG)

f: the objective function *gradf*: the gradient function (should return a 1D np.ndarray) *x*: the initial condition

Returns *x* the optimal value for *x* *flog*: a list of all the objective values *function_eval* number of *fn* evaluations

status: string describing convergence status

`GPy.inference.scg.exponents` (*fnow, current_grad*)

`GPy.inference.scg.print_out` (*len_maxiters, fnow, current_grad, beta, iteration*)

GPy.inference.sgd module

```
class GPy.inference.sgd.opt_SGD (start,      iterations=10,      learning_rate=0.0001,      momen-
                                tum=0.9,      model=None,      messages=False,      batch_size=1,
                                self_paced=False,      center=True,      iteration_file=None,      learn-
                                ing_rate_adaptation=None,      actual_iter=None,      schedule=None,
                                **kwargs)
```

Bases: `GPy.inference.optimization.Optimizer`

Optimize using stochastic gradient descent.

Parameters

- **Model** – reference to the Model object
- **iterations** – number of iterations
- **learning_rate** – learning rate
- **momentum** – momentum

`adapt_learning_rate` (*t, D*)

`check_for_missing` (*data*)

`get_param_shapes` (*N=None, input_dim=None*)

`non_null_samples` (*data*)

`opt` (*f_fp=None, f=None, fp=None*)

`plot_traces` ()

`restore_constraints` (*c*)

`shift_constraints` (*j*)

`step_with_missing_data` (*f_fp, X, step, shapes*)

`subset_parameter_vector` (*x, samples, param_shapes*)

Module contents

GPy.kern package

Subpackages

GPy.kern.parts package

Submodules

GPy.kern.parts.Brownian module

class GPy.kern.parts.Brownian.**Brownian** (*input_dim*, *variance=1.0*)

Bases: GPy.kern.parts.kernpart.Kernpart

Brownian Motion kernel.

Parameters

- **input_dim** (*int*) – the number of input dimensions
- **variance** (*float*) –

K (*X*, *X2*, *target*)

Kdiag (*X*, *target*)

dK_dX (*dL_dK*, *X*, *X2*, *target*)

dK_dtheta (*dL_dK*, *X*, *X2*, *target*)

dKdiag_dX (*dL_dKdiag*, *X*, *target*)

dKdiag_dtheta (*dL_dKdiag*, *X*, *target*)

GPy.kern.parts.Brownian.**theta** (*x*)

Heavisdie step function

GPy.kern.parts.Matern32 module

class GPy.kern.parts.Matern32.**Matern32** (*input_dim*, *variance=1.0*, *lengthscale=None*,
ARD=False)

Bases: GPy.kern.parts.kernpart.Kernpart

Matern 3/2 kernel:

$$k(r) = \sigma^2 (1 + \sqrt{3}r) \exp(-\sqrt{3}r) \quad \text{where } r = \sqrt{\sum_{i=1}^i \text{input_dim} \frac{(x_i - y_i)^2}{\ell_i^2}}$$

Parameters

- **input_dim** (*int*) – the number of input dimensions
- **variance** (*float*) – the variance σ^2
- **lengthscale** (*array or list of the appropriate size (or float if there is only one lengthscale parameter)*) – the vector of lengthscale ℓ_i
- **ARD** (*Boolean*) – Auto Relevance Determination. If equal to “False”, the kernel is isotropic (ie. one single lengthscale parameter ℓ), otherwise there is one lengthscale parameter per dimension.

Return type kernel object

Gram_matrix (*F*, *F1*, *F2*, *lower*, *upper*)

Return the Gram matrix of the vector of functions F with respect to the RKHS norm. The use of this function is limited to input_dim=1.

Parameters

- **F** (*np.array*) – vector of functions
- **F1** (*np.array*) – vector of derivatives of F
- **F2** (*np.array*) – vector of second derivatives of F
- **lower,upper** (*floats*) – boundaries of the input domain

K (*X*, *X2*, *target*)

Compute the covariance matrix between *X* and *X2*.

Kdiag (*X*, *target*)

Compute the diagonal of the covariance matrix associated to *X*.

dK_dX (*dL_dK*, *X*, *X2*, *target*)

derivative of the covariance matrix with respect to *X*.

dK_dtheta (*dL_dK*, *X*, *X2*, *target*)

derivative of the covariance matrix with respect to the parameters.

dKdiag_dX (*dL_dKdiag*, *X*, *target*)

dKdiag_dtheta (*dL_dKdiag*, *X*, *target*)

derivative of the diagonal of the covariance matrix with respect to the parameters.

GPy.kern.parts.Matern52 module

class GPy.kern.parts.Matern52.**Matern52** (*input_dim*, *variance=1.0*, *lengthscale=None*,
ARD=False)

Bases: GPy.kern.parts.kernpart.Kernpart

Matern 5/2 kernel:

$$k(r) = \sigma^2 \left(1 + \sqrt{5}r + \frac{5}{3}r^2 \right) \exp(-\sqrt{5}r) \quad \text{where } r = \sqrt{\sum_{i=1}^i \text{input_dim} \frac{(x_i - y_i)^2}{\ell_i^2}}$$

Parameters

- **input_dim** (*int*) – the number of input dimensions
- **variance** (*float*) – the variance σ^2
- **lengthscale** (*array or list of the appropriate size (or float if there is only one lengthscale parameter)*) – the vector of lengthscale ℓ_i
- **ARD** (*Boolean*) – Auto Relevance Determination. If equal to “False”, the kernel is isotropic (ie. one single lengthscale parameter ℓ), otherwise there is one lengthscale parameter per dimension.

Return type kernel object

Gram_matrix (*F*, *F1*, *F2*, *F3*, *lower*, *upper*)

Return the Gram matrix of the vector of functions *F* with respect to the RKHS norm. The use of this function is limited to input_dim=1.

Parameters

- **F** (*np.array*) – vector of functions
- **F1** (*np.array*) – vector of derivatives of *F*
- **F2** (*np.array*) – vector of second derivatives of *F*
- **F3** (*np.array*) – vector of third derivatives of *F*
- **lower,upper** (*floats*) – boundaries of the input domain

K (*X*, *X2*, *target*)

Compute the covariance matrix between *X* and *X2*.

Kdiag (*X*, *target*)

Compute the diagonal of the covariance matrix associated to *X*.

dK_dX (*dL_dK*, *X*, *X2*, *target*)

derivative of the covariance matrix with respect to *X*.

dK_dtheta (*dL_dK*, *X*, *X2*, *target*)

derivative of the covariance matrix with respect to the parameters.

dKdiag_dX (*dL_dKdiag*, *X*, *target*)

dKdiag_dtheta (*dL_dKdiag*, *X*, *target*)

derivative of the diagonal of the covariance matrix with respect to the parameters.

GPy.kern.parts.ODE_1 module

class GPy.kern.parts.ODE_1.ODE_1 (*input_dim=1*, *varianceU=1.0*, *varianceY=1.0*, *lengthscaleU=None*, *lengthscaleY=None*)

Bases: GPy.kern.parts.kernpart.Kernpart

kernel resultiong from a first order ODE with OU driving GP

Parameters

- **input_dim** (*int*) – the number of input dimension, has to be equal to one
- **varianceU** (*float*) – variance of the driving GP
- **lengthscaleU** (*float*) – lengthscale of the driving GP ($\sqrt{3}/\text{lengthscaleU}$)
- **varianceY** (*float*) – ‘variance’ of the transfer function
- **lengthscaleY** (*float*) – ‘lengthscale’ of the transfer function ($1/\text{lengthscaleY}$)

Return type kernel object

K (*X*, *X2*, *target*)

Compute the covariance matrix between *X* and *X2*.

Kdiag (*X*, *target*)

Compute the diagonal of the covariance matrix associated to *X*.

dK_dtheta (*dL_dK*, *X*, *X2*, *target*)

derivative of the covariance matrix with respect to the parameters.

GPy.kern.parts.ODE_UY module

class GPy.kern.parts.ODE_UY.ODE_UY (*input_dim=2*, *varianceU=1.0*, *varianceY=1.0*, *lengthscaleU=None*, *lengthscaleY=None*)

Bases: GPy.kern.parts.kernpart.Kernpart

kernel resultiong from a first order ODE with OU driving GP

Parameters

- **input_dim** (*int*) – the number of input dimension, has to be equal to one
- **input_lengthU** – the number of input U length
- **varianceU** (*float*) – variance of the driving GP
- **lengthscaleU** (*float*) – lengthscale of the driving GP ($\sqrt{3}/\text{lengthscaleU}$)
- **varianceY** (*float*) – ‘variance’ of the transfer function
- **lengthscaleY** (*float*) – ‘lengthscale’ of the transfer function ($1/\text{lengthscaleY}$)

Return type kernel object

K (*X*, *X2*, *target*)

Compute the covariance matrix between *X* and *X2*.

Kdiag (*X*, *target*)

Compute the diagonal of the covariance matrix associated to *X*.

dK_dtheta (*dL_dK*, *X*, *X2*, *target*)

derivative of the covariance matrix with respect to the parameters.

`GPy.kern.parts.ODE_UY.index_to_slices` (*index*)

take a numpy array of integers (*index*) and return a nested list of slices such that the slices describe the start, stop points for each integer in the index.

e.g. `>>> index = np.asarray([0,0,0,1,1,1,2,2,2])` returns `>>> [[slice(0,3,None)], [slice(3,6,None)], [slice(6,9,None)]]`

or, a more complicated example `>>> index = np.asarray([0,0,1,1,0,2,2,2,1,1])` returns `>>> [[slice(0,2,None), slice(4,5,None)], [slice(2,4,None), slice(8,10,None)], [slice(5,8,None)]]`

GPy.kern.parts.bias module

class `GPy.kern.parts.bias.Bias` (*input_dim*, *variance=1.0*)

Bases: `GPy.kern.parts.kernpart.Kernpart`

K (*X*, *X2*, *target*)

Kdiag (*X*, *target*)

dK_dX (*dL_dK*, *X*, *X2*, *target*)

dK_dtheta (*dL_dKdiag*, *X*, *X2*, *target*)

dKdiag_dX (*dL_dKdiag*, *X*, *target*)

dKdiag_dtheta (*dL_dKdiag*, *X*, *target*)

dpsi0_dZ (*dL_dpsi0*, *Z*, *mu*, *S*, *target*)

dpsi0_dmuS (*dL_dpsi0*, *Z*, *mu*, *S*, *target_mu*, *target_S*)

dpsi0_dtheta (*dL_dpsi0*, *Z*, *mu*, *S*, *target*)

dpsi1_dZ (*dL_dpsi1*, *Z*, *mu*, *S*, *target*)

dpsi1_dmuS (*dL_dpsi1*, *Z*, *mu*, *S*, *target_mu*, *target_S*)

dpsi1_dtheta (*dL_dpsi1*, *Z*, *mu*, *S*, *target*)

dpsi2_dZ (*dL_dpsi2*, *Z*, *mu*, *S*, *target*)

dpsi2_dmuS (*dL_dpsi2*, *Z*, *mu*, *S*, *target_mu*, *target_S*)

dpsi2_dtheta (*dL_dpsi2*, *Z*, *mu*, *S*, *target*)

psi0 (*Z*, *mu*, *S*, *target*)

psi1 (*Z*, *mu*, *S*, *target*)

psi2 (*Z*, *mu*, *S*, *target*)

GPy.kern.parts.coregionalize module

class `GPy.kern.parts.coregionalize.Coregionalize` (*output_dim*, *rank=1*, *W=None*, *kappa=None*)

Bases: `GPy.kern.parts.kernpart.Kernpart`

Covariance function for intrinsic/linear coregionalization models

This covariance has the form: .. math:

$$\mathbf{B} = \mathbf{W} \mathbf{W}^T + \text{diag}(\kappa)$$

An intrinsic/linear coregionalization covariance function of the form: .. math:

$$k_2(\mathbf{x}, \mathbf{y}) = \mathbf{B} k(\mathbf{x}, \mathbf{y})$$

it is obtained as the tensor product between a covariance function $k(\mathbf{x}, \mathbf{y})$ and \mathbf{B} .

Parameters

- **output_dim** (*int*) – number of outputs to coregionalize
- **rank** (*int*) – number of columns of the \mathbf{W} matrix (this parameter is ignored if parameter \mathbf{W} is not None)
- **W** (*numpy array of dimensionality (num_output, W_columns)*) – a low rank matrix that determines the correlations between the different outputs, together with κ it forms the coregionalization matrix \mathbf{B}
- **kappa** (*numpy array of dimensionality (output_dim,)*) – a vector which allows the outputs to behave independently

K (*index, index2, target*)

Kdiag (*index, target*)

dK_dX (*dL_dK, X, X2, target*)

dK_dtheta (*dL_dK, index, index2, target*)

dK_dtheta_old (*dL_dK, index, index2, target*)

dKdiag_dtheta (*dL_dKdiag, index, target*)

GPy.kern.parts.eq_ode1 module

class GPy.kern.parts.eq_ode1.**Eq_ode1** (*output_dim, W=None, rank=1, kappa=None, length-scale=1.0, decay=None, delay=None*)

Bases: GPy.kern.parts.kernpart.Kernpart

Covariance function for first order differential equation driven by an exponentiated quadratic covariance.

This outputs of this kernel have the form .. math:

$$\text{rac} \{ \text{ext}\{d\} y_j \} \{ \text{ext}\{d\} t \} = \sum_{i=1}^R w_{j,i} f_i(t - \delta_j) + \sqrt{\kappa_j} g_j(t) - d_{jy_j}(t)$$

where R is the rank of the system, $w_{j,i}$ is the sensitivity of the j 'th output to the i 'th latent function, d_j is the decay rate of the j 'th output and $f_i(t)$ and $g_i(t)$ are independent latent Gaussian processes governed by an exponentiated quadratic covariance.

param output_dim number of outputs driven by latent function.

type output_dim int

param W sensitivities of each output to the latent driving function.

type W ndarray (output_dim x rank).

param rank If rank is greater than 1 then there are assumed to be a total of rank latent forces independently driving the system, each with identical covariance.

type rank int

param decay decay rates for the first order system.

type decay array of length output_dim.

param delay delay between latent force and output response.

type delay array of length output_dim.

param kappa diagonal term that allows each latent output to have an independent component to the response.

type kappa array of length output_dim.

K (X, X2, target)

Kdiag (index, target)

dK_dX (dL_dK, X, X2, target)

dK_dtheta (dL_dK, X, X2, target)

dKdiag_dtheta (dL_dKdiag, index, target)

GPy.kern.parts.exponential module

class GPy.kern.parts.exponential.**Exponential** (input_dim, variance=1.0, lengthscale=None, ARD=False)

Bases: GPy.kern.parts.kernpart.Kernpart

Exponential kernel (aka Ornstein-Uhlenbeck or Matern 1/2)

$$k(r) = \sigma^2 \exp(-r) \quad \text{where } r = \sqrt{\sum_{i=1}^{\text{input_dim}} \frac{(x_i - y_i)^2}{\ell_i^2}}$$

Parameters

- **input_dim** (*int*) – the number of input dimensions
- **variance** (*float*) – the variance σ^2
- **lengthscale** (*array or list of the appropriate size (or float if there is only one lengthscale parameter)*) – the vector of lengthscale ℓ_i
- **ARD** (*Boolean*) – Auto Relevance Determination. If equal to “False”, the kernel is isotropic (ie. one single lengthscale parameter ℓ), otherwise there is one lengthscale parameter per dimension.

Return type kernel object

Gram_matrix (F, F1, lower, upper)

Return the Gram matrix of the vector of functions F with respect to the RKHS norm. The use of this function is limited to input_dim=1.

Parameters

- **F** (*np.array*) – vector of functions
- **F1** (*np.array*) – vector of derivatives of F
- **lower,upper** (*floats*) – boundaries of the input domain

K (X, X2, target)

Compute the covariance matrix between X and X2.

Kdiag (X, target)

Compute the diagonal of the covariance matrix associated to X.

dK_dX (*dL_dK*, *X*, *X2*, *target*)

derivative of the covariance matrix with respect to *X*.

dK_dtheta (*dL_dK*, *X*, *X2*, *target*)

derivative of the covariance matrix with respect to the parameters.

dKdiag_dX (*dL_dKdiag*, *X*, *target*)

dKdiag_dtheta (*dL_dKdiag*, *X*, *target*)

derivative of the diagonal of the covariance matrix with respect to the parameters.

GPy.kern.parts.finite_dimensional module

class GPy.kern.parts.finite_dimensional.**FiniteDimensional** (*input_dim*, *F*, *G*, *variance=1.0*, *weights=None*)

Bases: GPy.kern.parts.kernpart.Kernpart

K (*X*, *X2*, *target*)

Kdiag (*X*, *target*)

dK_dtheta (*X*, *X2*, *target*)

Return shape is NxMx(Ntheta)

dKdiag_dtheta (*X*, *target*)

GPy.kern.parts.fixed module

class GPy.kern.parts.fixed.**Fixed** (*input_dim*, *K*, *variance=1.0*)

Bases: GPy.kern.parts.kernpart.Kernpart

K (*X*, *X2*, *target*)

dK_dX (*partial*, *X*, *X2*, *target*)

dK_dtheta (*partial*, *X*, *X2*, *target*)

dKdiag_dX (*partial*, *X*, *target*)

GPy.kern.parts.gibbs module

class GPy.kern.parts.gibbs.**Gibbs** (*input_dim*, *variance=1.0*, *mapping=None*, *ARD=False*)

Bases: GPy.kern.parts.kernpart.Kernpart

Gibbs non-stationary covariance function.

$$r = \text{sqrt}((x_i - x_j)' * (x_i - x_j))$$

$$k(x_i, x_j) = \sigma^2 * Z * \exp(-r^2 / (l(x) * l(x) + l(x') * l(x')))$$

$$Z = (2 * l(x) * l(x') / (l(x) * l(x) + l(x') * l(x')^q)^{1/2}$$

where : $l(x)$ is a function giving the lengthscale as a function of space and : q is the dimensionality of the input space. The parameters are : σ^2 , the process variance, and the parameters of $l(x)$ which is a function that can be specified by : $param_{input_dim}$: the number of input dimensions : $type_{input_dim}$: `int` : $param_{variance}$: the variance : $math : \sigma^2$: `type`

See Mark Gibbs's thesis for more details: Gibbs, M. N. (1997). Bayesian Gaussian Processes for Regression and Classification. PhD thesis, Department of Physics, University of Cambridge. Or also see Page 93 of Gaussian Processes for Machine Learning by Rasmussen and Williams. Although note that we do not constrain the lengthscale to be positive by default. This allows anticorrelation to occur. The positive constraint can be included by the user manually.

K (*X*, *X2*, *target*)

Return covariance between *X* and *X2*.

Kdiag (*X*, *target*)

Compute the diagonal of the covariance matrix for *X*.

dK_dX (*dL_dK*, *X*, *X2*, *target*)

Derivative of the covariance matrix with respect to *X*.

dK_dtheta (*dL_dK*, *X*, *X2*, *target*)

Derivative of the covariance with respect to the parameters.

dKdiag_dX (*dL_dKdiag*, *X*, *target*)

Gradient of diagonal of covariance with respect to *X*.

dKdiag_dtheta (*dL_dKdiag*, *X*, *target*)

Gradient of diagonal of covariance with respect to parameters.

GPy.kern.parts.hetero module

class GPy.kern.parts.hetero.**Hetero** (*input_dim*, *mapping=None*, *transform=None*)

Bases: GPy.kern.parts.kernpart.Kernpart

TODO: Need to constrain the function outputs positive (still thinking of best way of doing this!!! Yes, intend to use transformations, but what's the *best* way). Currently just squaring output.

Heteroschedastic noise which depends on input location. See, for example, this paper by Goldberg et al.

$$k(x_i, x_j) = \delta_{i,j} \sigma^2(x_i)$$

where : $\sigma^2(x)$ 'is a function giving the variance as a function of input space' and : $\delta_{i,j}$ 'is the Kronecker delta'

The parameters are the parameters of $\sigma^2(x)$ which is a function that can be specified by the user, by default an multi-layer peceptron is used.

Parameters

- **input_dim** (*int*) – the number of input dimensions
- **mapping** (*GPy.core.Mapping*) – the mapping that gives the lengthscale across the input space (by default GPy.mappings.MLP is used with 20 hidden nodes).

Return type Kernpart object

See this paper:

Goldberg, P. W. Williams, C. K. I. and Bishop, C. M. (1998) Regression with Input-dependent Noise: a Gaussian Process Treatment In Advances in Neural Information Processing Systems, Volume 10, pp. 493-499. MIT Press

for a Gaussian process treatment of this problem.

K (*X*, *X2*, *target*)

Return covariance between *X* and *X2*.

Kdiag (*X*, *target*)

Compute the diagonal of the covariance matrix for *X*.

dK_dX (*dL_dK*, *X*, *X2*, *target*)

Derivative of the covariance matrix with respect to *X*.

dK_dtheta (*dL_dK*, *X*, *X2*, *target*)

Derivative of the covariance with respect to the parameters.

dKdiag_dX (*dL_dKdiag*, *X*, *target*)

Gradient of diagonal of covariance with respect to *X*.

dKdiag_dtheta (*dL_dKdiag*, *X*, *target*)

Gradient of diagonal of covariance with respect to parameters.

GPy.kern.parts.hierarchical module

class GPy.kern.parts.hierarchical.**Hierarchical** (*parts*)

Bases: GPy.kern.parts.kernpart.Kernpart

A kernel part which can represent a hierarchy of independence: a generalisation of independent_outputs

K (*X*, *X2*, *target*)

Kdiag (*X*, *target*)

dK_dX (*dL_dK*, *X*, *X2*, *target*)

dK_dtheta (*dL_dK*, *X*, *X2*, *target*)

dKdiag_dX (*dL_dKdiag*, *X*, *target*)

dKdiag_dtheta (*dL_dKdiag*, *X*, *target*)

GPy.kern.parts.independent_outputs module

class GPy.kern.parts.independent_outputs.**IndependentOutputs** (*k*)

Bases: GPy.kern.parts.kernpart.Kernpart

A kernel part which can represent several independent functions. this kernel ‘switches off’ parts of the matrix where the output indexes are different.

The index of the functions is given by the last column in the input *X* the rest of the columns of *X* are passed to the kernel for computation (in blocks).

K (*X*, *X2*, *target*)

Kdiag (*X*, *target*)

dK_dX (*dL_dK*, *X*, *X2*, *target*)

dK_dtheta (*dL_dK*, *X*, *X2*, *target*)

dKdiag_dX (*dL_dKdiag*, *X*, *target*)

dKdiag_dtheta (*dL_dKdiag*, *X*, *target*)

GPy.kern.parts.independent_outputs.**index_to_slices** (*index*)

take a numpy array of integers (*index*) and return a nested list of slices such that the slices describe the start, stop points for each integer in the index.

e.g. >>> *index* = np.asarray([0,0,0,1,1,1,2,2,2]) returns >>> [[slice(0,3,None)], [slice(3,6,None)], [slice(6,9,None)]]

or, a more complicated example >>> *index* = np.asarray([0,0,1,1,0,2,2,2,1,1]) returns >>> [[slice(0,2,None), slice(4,5,None)], [slice(2,4,None), slice(8,10,None)], [slice(5,8,None)]]

GPy.kern.parts.kernpart module

class GPy.kern.parts.kernpart.**Kernpart** (*input_dim*)

Bases: object

K (*X*, *X2*, *target*)

Kdiag (*X*, *target*)

dK_dX (*dL_dK*, *X*, *X2*, *target*)

dK_dtheta (*dL_dK*, *X*, *X2*, *target*)

dKdiag_dX (*dL_dK*, *X*, *target*)

dKdiag_dtheta (*dL_dKdiag*, *X*, *target*)

dpsi0_dmuS (*dL_dpsi0*, *Z*, *mu*, *S*, *target_mu*, *target_S*)

```

dpsi0_dtheta (dL_dpsi0, Z, mu, S, target)
dpsi1_dZ (dL_dpsi1, Z, mu, S, target)
dpsi1_dmuS (dL_dpsi1, Z, mu, S, target_mu, target_S)
dpsi1_dtheta (Z, mu, S, target)
dpsi2_dZ (dL_dpsi2, Z, mu, S, target)
dpsi2_dmuS (dL_dpsi2, Z, mu, S, target_mu, target_S)
dpsi2_dtheta (dL_dpsi2, Z, mu, S, target)
psi0 (Z, mu, S, target)
psi1 (Z, mu, S, target)
psi2 (Z, mu, S, target)
class GPy.kern.parts.kernpart.Kernpart_inner (input_dim)
    Bases: GPy.kern.parts.kernpart.Kernpart
class GPy.kern.parts.kernpart.Kernpart_stationary (input_dim, lengthscale=None,
    ARD=False)
    Bases: GPy.kern.parts.kernpart.Kernpart
dKdiag_dX (dL_dK, X, target)
dKdiag_dtheta (dL_dKdiag, X, target)

```

GPy.kern.parts.linear module

```

class GPy.kern.parts.linear.Linear (input_dim, variances=None, ARD=False)
    Bases: GPy.kern.parts.kernpart.Kernpart
    Linear kernel

```

$$k(x, y) = \sum_{i=1}^i nput_{dim} \sigma_i^2 x_i y_i$$

Parameters

- **input_dim** (*int*) – the number of input dimensions
- **variances** (*array or list of the appropriate size (or float if there is only one variance parameter)*) – the vector of variances σ_i^2
- **ARD** (*Boolean*) – Auto Relevance Determination. If equal to “False”, the kernel has only one variance parameter σ^2 , otherwise there is one variance parameter per dimension.

Return type

```

K (X, X2, target)
Kdiag (X, target)
dK_dX (dL_dK, X, X2, target)
dK_dtheta (dL_dK, X, X2, target)
dKdiag_dX (dL_dKdiag, X, target)
dKdiag_dtheta (dL_dKdiag, X, target)
dpsi0_dmuS (dL_dpsi0, Z, mu, S, target_mu, target_S)
dpsi0_dtheta (dL_dpsi0, Z, mu, S, target)

```


dpsi1_dZ (*dL_dpsi1*, *Z*, *mu*, *S*, *target*)

dpsi1_dmuS (*dL_dpsi1*, *Z*, *mu*, *S*, *target_mu*, *target_S*)
Do nothing for S, it does not affect psi1

dpsi1_dtheta (*dL_dpsi1*, *Z*, *mu*, *S*, *target*)
the variance, it does nothing

dpsi2_dZ (*dL_dpsi2*, *Z*, *mu*, *S*, *target*)

dpsi2_dmuS (*dL_dpsi2*, *Z*, *mu*, *S*, *target_mu*, *target_S*)
Think N,num_inducing,num_inducing,input_dim

dpsi2_dmuS_new (*dL_dpsi2*, *Z*, *mu*, *S*, *target_mu*, *target_S*)

dpsi2_dtheta (*dL_dpsi2*, *Z*, *mu*, *S*, *target*)

dpsi2_dtheta_new (*dL_dpsi2*, *Z*, *mu*, *S*, *target*)

psi0 (*Z*, *mu*, *S*, *target*)

psi1 (*Z*, *mu*, *S*, *target*)
the variance, it does nothing

psi2 (*Z*, *mu*, *S*, *target*)

psi2_new (*Z*, *mu*, *S*, *target*)

GPy.kern.parts.mlp module

class GPy.kern.parts.mlp.**MLP** (*input_dim*, *variance=1.0*, *weight_variance=None*,
bias_variance=100.0, *ARD=False*)

Bases: GPy.kern.parts.kernpart.Kernpart

Multi layer perceptron kernel (also known as arc sine kernel or neural network kernel)

$$k(x, y) = \sigma^2 \frac{2}{\pi} \operatorname{asin} \left(\frac{\sigma_w^2 x^\top y + \sigma_b^2}{\sqrt{\sigma_w^2 x^\top x + \sigma_b^2 + 1} \sqrt{\sigma_w^2 y^\top y + \sigma_b^2 + 1}} \right)$$

Parameters

- **input_dim** (*int*) – the number of input dimensions
- **variance** (*float*) – the variance σ^2
- **weight_variance** (*array or list of the appropriate size (or float if there is only one weight variance parameter)*) – the vector of the variances of the prior over input weights in the neural network σ_w^2
- **bias_variance** – the variance of the prior over bias parameters σ_b^2
- **ARD** (*Boolean*) – Auto Relevance Determination. If equal to “False”, the kernel is isotropic (ie. one weight variance parameter σ_w^2), otherwise there is one weight variance parameter per dimension.

Return type Kernpart object

K (*X*, *X2*, *target*)
Return covariance between X and X2.

Kdiag (*X*, *target*)
Compute the diagonal of the covariance matrix for X.

dK_dX (*dL_dK*, *X*, *X2*, *target*)
Derivative of the covariance matrix with respect to X

dK_dtheta (*dL_dK*, *X*, *X2*, *target*)
Derivative of the covariance with respect to the parameters.

dKdiag_dX (*dL_dKdiag*, *X*, *target*)
Gradient of diagonal of covariance with respect to *X*

GPy.kern.parts.periodic_Matern32 module

class GPy.kern.parts.periodic_Matern32.**PeriodicMatern32** (*input_dim=1*, *variance=1.0*,
lengthscale=None, *period=6.283185307179586*,
n_freq=10, *lower=0.0*, *upper=12.566370614359172*)

Bases: GPy.kern.parts.kernpart.Kernpart

Kernel of the periodic subspace (up to a given frequency) of a Matern 3/2 RKHS. Only defined for *input_dim=1*.

Parameters

- **input_dim** (*int*) – the number of input dimensions
- **variance** (*float*) – the variance of the Matern kernel
- **lengthscale** (*np.ndarray of size (input_dim,)*) – the lengthscale of the Matern kernel
- **period** (*float*) – the period
- **n_freq** (*int*) – the number of frequencies considered for the periodic subspace

Return type kernel object

Gram_matrix ()

K (*X*, *X2*, *target*)
Compute the covariance matrix between *X* and *X2*.

Kdiag (*X*, *target*)
Compute the diagonal of the covariance matrix associated to *X*.

dK_dtheta (**args*, ***kws*)
derivative of the covariance matrix with respect to the parameters (shape is *num_data* x *num_inducing* x *num_params*)

dKdiag_dtheta (**args*, ***kws*)
derivative of the diagonal covariance matrix with respect to the parameters

GPy.kern.parts.periodic_Matern52 module

class GPy.kern.parts.periodic_Matern52.**PeriodicMatern52** (*input_dim=1*, *variance=1.0*,
lengthscale=None, *period=6.283185307179586*,
n_freq=10, *lower=0.0*, *upper=12.566370614359172*)

Bases: GPy.kern.parts.kernpart.Kernpart

Kernel of the periodic subspace (up to a given frequency) of a Matern 5/2 RKHS. Only defined for *input_dim=1*.

Parameters

- **input_dim** (*int*) – the number of input dimensions
- **variance** (*float*) – the variance of the Matern kernel
- **lengthscale** (*np.ndarray of size (input_dim,)*) – the lengthscale of the Matern kernel

- **period** (*float*) – the period
- **n_freq** (*int*) – the number of frequencies considered for the periodic subspace

Return type kernel object

Gram_matrix ()

K (*X*, *X2*, *target*)

Compute the covariance matrix between *X* and *X2*.

Kdiag (*X*, *target*)

Compute the diagonal of the covariance matrix associated to *X*.

dK_dtheta (**args*, ***kws*)

derivative of the covariance matrix with respect to the parameters (shape is num_data x num_inducing x num_params)

dKdiag_dtheta (**args*, ***kws*)

derivative of the diagonal of the covariance matrix with respect to the parameters

GPy.kern.parts.periodic_exponential module

```
class GPy.kern.parts.periodic_exponential.PeriodicExponential (input_dim=1, variance=1.0, lengthscale=None, period=6.283185307179586, n_freq=10, lower=0.0, upper=12.566370614359172)
```

Bases: `GPy.kern.parts.kernpart.Kernpart`

Kernel of the periodic subspace (up to a given frequency) of a exponential (Matern 1/2) RKHS. Only defined for `input_dim=1`.

Parameters

- **input_dim** (*int*) – the number of input dimensions
- **variance** (*float*) – the variance of the Matern kernel
- **lengthscale** (*np.ndarray of size (input_dim,)*) – the lengthscale of the Matern kernel
- **period** (*float*) – the period
- **n_freq** (*int*) – the number of frequencies considered for the periodic subspace

Return type kernel object

Gram_matrix ()

K (*X*, *X2*, *target*)

Compute the covariance matrix between *X* and *X2*.

Kdiag (*X*, *target*)

Compute the diagonal of the covariance matrix associated to *X*.

dK_dtheta (**args*, ***kws*)

derivative of the covariance matrix with respect to the parameters (shape is N x num_inducing x num_params)

dKdiag_dtheta (**args*, ***kws*)

derivative of the diagonal of the covariance matrix with respect to the parameters

GPy.kern.parts.poly module

class GPy.kern.parts.poly.**POLY** (*input_dim*, *variance=1.0*, *weight_variance=None*, *bias_variance=1.0*, *degree=2*, *ARD=False*)

Bases: GPy.kern.parts.kernpart.Kernpart

Polynomial kernel parameter initialisation. Included for completeness, but generally not recommended, is the polynomial kernel:

$$k(x, y) = \sigma^2 (\sigma_w^2 x' y + \sigma_b^2)^d$$

The kernel parameters are σ^2 (variance), σ_w^2 (weight_variance), σ_b^2 (bias_variance) and d (degree). Only gradients of the first three are provided for kernel optimisation, it is assumed that polynomial degree would be set by hand.

The kernel is not recommended as it is badly behaved when the $\sigma_w^2 x' y + \sigma_b^2$ has a magnitude greater than one. For completeness there is an automatic relevance determination version of this kernel provided (NOTE YET IMPLEMENTED!). :param input_dim: the number of input dimensions :type input_dim: int :param variance: the variance σ^2 :type variance: float :param weight_variance: the vector of the variances of the prior over input weights in the neural network σ_w^2 :type weight_variance: array or list of the appropriate size (or float if there is only one weight variance parameter) :param bias_variance: the variance of the prior over bias parameters σ_b^2 :param degree: the degree of the polynomial. :type degree: int :param ARD: Auto Relevance Determination. If equal to “False”, the kernel is isotropic (ie. one weight variance parameter σ_w^2), otherwise there is one weight variance parameter per dimension. :type ARD: Boolean :rtype: Kernpart object

K (*X*, *X2*, *target*)

Return covariance between X and X2.

Kdiag (*X*, *target*)

Compute the diagonal of the covariance matrix for X.

dK_dX (*dL_dK*, *X*, *X2*, *target*)

Derivative of the covariance matrix with respect to X

dK_dtheta (*dL_dK*, *X*, *X2*, *target*)

Derivative of the covariance with respect to the parameters.

dKdiag_dX (*dL_dKdiag*, *X*, *target*)

Gradient of diagonal of covariance with respect to X

GPy.kern.parts.prod module

class GPy.kern.parts.prod.**Prod** (*k1*, *k2*, *tensor=False*)

Bases: GPy.kern.parts.kernpart.Kernpart

Computes the product of 2 kernels

Parameters

- **k2** (*k1*,) – the kernels to multiply
- **tensor** (*Boolean*) – The kernels are either multiply as functions defined on the same input space (default) or on the product of the input spaces

Return type kernel object

K (*X*, *X2*, *target*)

K1 (*X*, *X2*)

Compute the part of the kernel associated with k1.

K2 (*X*, *X2*)

Compute the part of the kernel associated with k2.

Kdiag (*X*, *target*)

Compute the diagonal of the covariance matrix associated to *X*.

dK_dX (*dL_dK*, *X*, *X2*, *target*)

derivative of the covariance matrix with respect to *X*.

dK_dtheta (*dL_dK*, *X*, *X2*, *target*)

Derivative of the covariance matrix with respect to the parameters.

dKdiag_dX (*dL_dKdiag*, *X*, *target*)

dKdiag_dtheta (*dL_dKdiag*, *X*, *target*)

GPy.kern.parts.prod_orthogonal module

class GPy.kern.parts.prod_orthogonal.**prod_orthogonal** (*k1*, *k2*)

Bases: GPy.kern.parts.kernpart.Kernpart

Computes the product of 2 kernels

Parameters **k2** (*k1*,) – the kernels to multiply

Return type kernel object

K (*X*, *X2*, *target*)

Kdiag (*X*, *target*)

Compute the diagonal of the covariance matrix associated to *X*.

dK_dX (*dL_dK*, *X*, *X2*, *target*)

derivative of the covariance matrix with respect to *X*.

dK_dtheta (*dL_dK*, *X*, *X2*, *target*)

derivative of the covariance matrix with respect to the parameters.

dKdiag_dX (*dL_dKdiag*, *X*, *target*)

dKdiag_dtheta (*dL_dKdiag*, *X*, *target*)

GPy.kern.parts.rational_quadratic module

class GPy.kern.parts.rational_quadratic.**RationalQuadratic** (*input_dim*, *variance=1.0*, *lengthscale=1.0*, *power=1.0*)

Bases: GPy.kern.parts.kernpart.Kernpart

rational quadratic kernel

$$k(r) = \sigma^2 \left(1 + \frac{r^2}{2\ell^2} \right)^{-\alpha} \quad \text{where } r^2 = (x - y)^2$$

Parameters

- **input_dim** (*int* (*input_dim=1* is the only value currently supported)) – the number of input dimensions
- **variance** (*float*) – the variance σ^2
- **lengthscale** (*float*) – the lengthscale ℓ
- **power** (*float*) – the power α

Return type Kernpart object

K (*X*, *X2*, *target*)

Kdiag (*X*, *target*)
dK_dX (*dL_dK*, *X*, *X2*, *target*)
 derivative of the covariance matrix with respect to *X*.
dK_dtheta (*dL_dK*, *X*, *X2*, *target*)
dKdiag_dX (*dL_dKdiag*, *X*, *target*)
dKdiag_dtheta (*dL_dKdiag*, *X*, *target*)

GPy.kern.parts.rbf module

class GPy.kern.parts.rbf.**RBF** (*input_dim*, *variance=1.0*, *lengthscale=None*, *ARD=False*)

Bases: GPy.kern.parts.kernpart.Kernpart

Radial Basis Function kernel, aka squared-exponential, exponentiated quadratic or Gaussian kernel:

$$k(r) = \sigma^2 \exp\left(-\frac{1}{2}r^2\right) \quad \text{where } r^2 = \sum_{i=1}^d \frac{(x_i - x'_i)^2}{\ell_i^2}$$

where ℓ_i is the lengthscale, σ^2 the variance and d the dimensionality of the input.

Parameters

- **input_dim** (*int*) – the number of input dimensions
- **variance** (*float*) – the variance of the kernel
- **lengthscale** (*array or list of the appropriate size (or float if there is only one lengthscale parameter)*) – the vector of lengthscale of the kernel
- **ARD** (*Boolean*) – Auto Relevance Determination. If equal to “False”, the kernel is isotropic (ie. one single lengthscale parameter ℓ), otherwise there is one lengthscale parameter per dimension.

Return type kernel object

K (*X*, *X2*, *target*)
Kdiag (*X*, *target*)
dK_dX (*dL_dK*, *X*, *X2*, *target*)
dK_dtheta (*dL_dK*, *X*, *X2*, *target*)
dKdiag_dX (*dL_dKdiag*, *X*, *target*)
dKdiag_dtheta (*dL_dKdiag*, *X*, *target*)
dpsi0_dmuS (*dL_dpsi0*, *Z*, *mu*, *S*, *target_mu*, *target_S*)
dpsi0_dtheta (*dL_dpsi0*, *Z*, *mu*, *S*, *target*)
dpsi1_dZ (*dL_dpsi1*, *Z*, *mu*, *S*, *target*)
dpsi1_dmuS (*dL_dpsi1*, *Z*, *mu*, *S*, *target_mu*, *target_S*)
dpsi1_dtheta (*dL_dpsi1*, *Z*, *mu*, *S*, *target*)
dpsi2_dZ (*dL_dpsi2*, *Z*, *mu*, *S*, *target*)
dpsi2_dmuS (*dL_dpsi2*, *Z*, *mu*, *S*, *target_mu*, *target_S*)
 Think *N*, *num_inducing*, *num_inducing*, *input_dim*
dpsi2_dtheta (*dL_dpsi2*, *Z*, *mu*, *S*, *target*)
 Shape *N*, *num_inducing*, *num_inducing*, *Ntheta*

```
psi0 (Z, mu, S, target)
psi1 (Z, mu, S, target)
psi2 (Z, mu, S, target)
weave_psi2 (mu, Zhat)
```

GPy.kern.parts.rbf_inv module

```
class GPy.kern.parts.rbf_inv.RBFInv(input_dim, variance=1.0, inv_lengthscale=None,
                                   ARD=False)
```

Bases: `GPy.kern.parts.rbf.RBF`

Radial Basis Function kernel, aka squared-exponential, exponentiated quadratic or Gaussian kernel. It only differs from RBF in that here the parametrization is wrt the inverse lengthscale:

$$k(r) = \sigma^2 \exp\left(-\frac{1}{2}r^2\right) \quad \text{where } r^2 = \sum_{i=1}^d \frac{(x_i - x'_i)^2}{\ell_i^2}$$

where ℓ_i is the lengthscale, σ^2 the variance and d the dimensionality of the input.

Parameters

- **input_dim** (*int*) – the number of input dimensions
- **variance** (*float*) – the variance of the kernel
- **lengthscale** (*array or list of the appropriate size (or float if there is only one lengthscale parameter)*) – the vector of lengthscale of the kernel
- **ARD** (*Boolean*) – Auto Relevance Determination. If equal to “False”, the kernel is isotropic (ie. one single lengthscale parameter ℓ), otherwise there is one lengthscale parameter per dimension.

Return type kernel object

```
dK_dX (dL_dK, X, X2, target)
dK_dtheta (dL_dK, X, X2, target)
dKdiag_dX (dL_dKdiag, X, target)
dpsi1_dZ (dL_dpsi1, Z, mu, S, target)
dpsi1_dmuS (dL_dpsi1, Z, mu, S, target_mu, target_S)
dpsi1_dtheta (dL_dpsi1, Z, mu, S, target)
dpsi2_dZ (dL_dpsi2, Z, mu, S, target)
dpsi2_dmuS (dL_dpsi2, Z, mu, S, target_mu, target_S)
    Think N,num_inducing,num_inducing,input_dim
dpsi2_dtheta (dL_dpsi2, Z, mu, S, target)
    Shape N,num_inducing,num_inducing,Ntheta
weave_psi2 (mu, Zhat)
```

GPy.kern.parts.rbfcos module

```
class GPy.kern.parts.rbfcos.RBFCos(input_dim, variance=1.0, frequencies=None, band-
                                   widths=None, ARD=False)
```

Bases: `GPy.kern.parts.kernpart.Kernpart`

```
K (X, X2, target)
```

```
Kdiag (X, target)  
dK_dX (dL_dK, X, X2, target)  
dK_dtheta (dL_dK, X, X2, target)  
dKdiag_dX (dL_dKdiag, X, target)  
dKdiag_dtheta (dL_dKdiag, X, target)
```

GPy.kern.parts.spline module

```
class GPy.kern.parts.spline.Spline (input_dim, variance=1.0, lengthscale=1.0)
```

Bases: `GPy.kern.parts.kernpart.Kernpart`

Spline kernel

Parameters

- **input_dim** (*int*) – the number of input dimensions (fixed to 1 right now TODO)
- **variance** (*float*) – the variance of the kernel

```
K (X, X2, target)  
Kdiag (X, target)  
dK_dtheta (X, X2, target)  
dKdiag_dX (X, target)  
dKdiag_dtheta (X, target)  
GPy.kern.parts.spline.theta (x)  
Heaviside step function
```

GPy.kern.parts.symmetric module

```
class GPy.kern.parts.symmetric.Symmetric (k, transform=None)
```

Bases: `GPy.kern.parts.kernpart.Kernpart`

Symmetrical kernels

Parameters

- **k** (*Kernpart*) – the kernel to symmetrify
- **transform** (*A numpy array (input_dim x input_dim) specifying the transform*) – the transform to use in symmetrification (allows symmetry on specified axes)

Return type `Kernpart`

```
K (X, X2, target)  
Compute the covariance matrix between X and X2.  
Kdiag (X, target)  
Compute the diagonal of the covariance matrix associated to X.  
dK_dX (dL_dK, X, X2, target)  
derivative of the covariance matrix with respect to X.  
dK_dtheta (dL_dK, X, X2, target)  
derivative of the covariance matrix with respect to the parameters.  
dKdiag_dX (dL_dKdiag, X, target)  
dKdiag_dtheta (dL_dKdiag, X, target)  
Compute the diagonal of the covariance matrix associated to X.
```



```
dKdiag_dtheta (dL_dKdiag, X, target)
dpsi0_dmuS (dL_dpsi0, Z, mu, S, target_mu, target_S)
dpsi0_dtheta (dL_dpsi0, Z, mu, S, target)
dpsi1_dZ (dL_dpsi1, Z, mu, S, target)
dpsi1_dmuS (dL_dpsi1, Z, mu, S, target_mu, target_S)
dpsi1_dtheta (dL_dpsi1, Z, mu, S, target)
dpsi2_dZ (dL_dpsi2, Z, mu, S, target)
dpsi2_dmuS (dL_dpsi2, Z, mu, S, target_mu, target_S)
dpsi2_dtheta (dL_dpsi2, Z, mu, S, target)
psi0 (Z, mu, S, target)
psi1 (Z, mu, S, target)
psi2 (Z, mu, S, target)
```

Module contents

Submodules

GPy.kern.constructors module

`GPy.kern.constructors.Brownian` (*input_dim*, *variance=1.0*)
Construct a Brownian motion kernel.

Parameters

- **input_dim** (*int*) – Dimensionality of the kernel
- **variance** (*float*) – the variance of the kernel

`GPy.kern.constructors.Matern32` (*input_dim*, *variance=1.0*, *lengthscale=None*, *ARD=False*)
Construct a Matern 3/2 kernel.

Parameters

- **input_dim** (*int*) – dimensionality of the kernel, obligatory
- **variance** (*float*) – the variance of the kernel
- **lengthscale** (*float*) – the lengthscale of the kernel
- **ARD** (*Boolean*) – Auto Relevance Determination (one lengthscale per dimension)

`GPy.kern.constructors.Matern52` (*input_dim*, *variance=1.0*, *lengthscale=None*, *ARD=False*)
Construct a Matern 5/2 kernel.

Parameters

- **input_dim** (*int*) – dimensionality of the kernel, obligatory
- **variance** (*float*) – the variance of the kernel
- **lengthscale** (*float*) – the lengthscale of the kernel

- **ARD** (*Boolean*) – Auto Relevance Determination (one lengthscale per dimension)

`GPY.kern.constructors.ODE_1` (*input_dim=1, varianceU=1.0, varianceY=1.0, lengthscaleU=None, lengthscaleY=None*)

kernel resultiong from a first order ODE with OU driving GP

Parameters

- **input_dim** (*int*) – the number of input dimension, has to be equal to one
- **varianceU** (*float*) – variance of the driving GP
- **lengthscaleU** (*float*) – lengthscale of the driving GP
- **varianceY** (*float*) – ‘variance’ of the transfer function
- **lengthscaleY** (*float*) – ‘lengthscale’ of the transfer function

Return type kernel object

`GPY.kern.constructors.ODE_UY` (*input_dim=2, varianceU=1.0, varianceY=1.0, lengthscaleU=None, lengthscaleY=None*)

kernel resultiong from a first order ODE with OU driving GP :param input_dim: the number of input dimension, has to be equal to one :type input_dim: int :param input_lengthU: the number of input U length :param varianceU: variance of the driving GP :type varianceU: float :param varianceY: ‘variance’ of the transfer function :type varianceY: float :param lengthscaleY: ‘lengthscale’ of the transfer function :type lengthscaleY: float :rtype: kernel object

`GPY.kern.constructors.bias` (*input_dim, variance=1.0*)

Construct a bias kernel.

Parameters

- **input_dim** (*int*) – dimensionality of the kernel, obligatory
- **variance** (*float*) – the variance of the kernel

`GPY.kern.constructors.build_lcm` (*input_dim, output_dim, kernel_list=[], rank=1, W=None, kappa=None*)

Builds a kernel of a linear coregionalization model

Input_dim Input dimensionality

Output_dim Number of outputs

Kernel_list List of coregionalized kernels, each element in the list will be multiplied by a different coregionalization matrix

Parameters rank (*integer*) – number tuples of the coregionalization parameters ‘coregion_W’

..note the kernels dimensionality is overwritten to fit input_dim

`GPY.kern.constructors.coregionalize` (*output_dim, rank=1, W=None, kappa=None*)

Coregionlization matrix B, of the form:

$$\mathbf{B} = \mathbf{W}\mathbf{W}^o p + \text{kappa}\mathbf{I}$$

An intrinsic/linear coregionalization kernel of the form:

$$k_2(x, y) = \mathbf{B}k(x, y)$$

it is obtained as the tensor product between a kernel k(x,y) and B.

Parameters

- **output_dim** (*int*) – the number of outputs to coregionalize
- **rank** (*int*) – number of columns of the W matrix (this parameter is ignored if parameter W is not `None`)
- **W** (*numpy array of dimensionality (num_outputs, rank)*) – a low rank matrix that determines the correlations between the different outputs, together with κ it forms the coregionalization matrix B
- **kappa** (*numpy array of dimensionality (output_dim,)*) – a vector which allows the outputs to behave independently

Return type kernel object

`GPy.kern.constructors.eq_sympy(input_dim, output_dim, ARD=False)`

Latent force model covariance, exponentiated quadratic with multiple outputs. Derived from a diffusion equation with the initial spatial condition layed down by a Gaussian process with lengthscale given by `shared_lengthscale`.

See IEEE Trans Pattern Anal Mach Intell. 2013 Nov;35(11):2693-705. doi: 10.1109/TPAMI.2013.86. Linear latent force models using Gaussian processes. Alvarez MA, Luengo D, Lawrence ND.

Parameters

- **input_dim** (*int*) – Dimensionality of the kernel
- **output_dim** (*int*) – number of outputs in the covariance function.
- **ARD** (*bool*) – whether or not to user ARD (default `False`).

`GPy.kern.constructors.exponential(input_dim, variance=1.0, lengthscale=None, ARD=False)`

Construct an exponential kernel

Parameters

- **input_dim** (*int*) – dimensionality of the kernel, obligatory
- **variance** (*float*) – the variance of the kernel
- **lengthscale** (*float*) – the lengthscale of the kernel
- **ARD** (*Boolean*) – Auto Relevance Determination (one lengthscale per dimension)

`GPy.kern.constructors.finite_dimensional(input_dim, F, G, variances=1.0, weights=None)`

Construct a finite dimensional kernel.

Parameters

- **input_dim** (*int*) – the number of input dimensions
- **F** (*np.array*) – `np.array` of functions with shape `(n,)` - the n basis functions
- **G** (*np.array*) – `np.array` with shape `(n,n)` - the Gram matrix associated to F
- **variances** – `np.ndarray` with shape `(n,)`

Type `np.ndarray`

`GPy.kern.constructors.fixed(input_dim, K, variance=1.0)`

Construct a Fixed effect kernel.

Parameters

- **input_dim** (*int (input_dim=1 is the only value currently supported)*) – the number of input dimensions
- **K** (*np.array*) – the variance σ^2

- **variance** (*float*) – kernel variance

Return type kern object

`GPy.kern.constructors.gibbs(input_dim, variance=1.0, mapping=None)`

Gibbs and MacKay non-stationary covariance function.

$$r = \sqrt{((x_i - x_j)' * (x_i - x_j))}$$

$$k(x_i, x_j) = \sigma^2 * Z * \exp(-r^2 / (l(x) * l(x) + l(x') * l(x')))$$

$$Z = \sqrt{2 * l(x) * l(x') / (l(x) * l(x) + l(x') * l(x'))}$$

Where $l(x)$ is a function giving the length scale as a function of space.

This is the non stationary kernel proposed by Mark Gibbs in his 1997 thesis. It is similar to an RBF but has a length scale that varies with input location. This leads to an additional term in front of the kernel.

The parameters are σ^2 , the process variance, and the parameters of $l(x)$ which is a function that can be specified by the user, by default an multi-layer perceptron is used.

Parameters

- **input_dim** (*int*) – the number of input dimensions
- **variance** (*float*) – the variance σ^2
- **mapping** (*GPy.core.Mapping*) – the mapping that gives the lengthscale across the input space.
- **ARD** (*Boolean*) – Auto Relevance Determination. If equal to “False”, the kernel is isotropic (ie. one weight variance parameter σ_w^2), otherwise there is one weight variance parameter per dimension.

Return type Kernpart object

`GPy.kern.constructors.hetero(input_dim, mapping=None, transform=None)`

`GPy.kern.constructors.hierarchical(k)`

TODO This can't be right! Construct a kernel with independent outputs from an existing kernel

`GPy.kern.constructors.independent_outputs(k)`

Construct a kernel with independent outputs from an existing kernel

`GPy.kern.constructors.linear(input_dim, variances=None, ARD=False)`

Construct a linear kernel.

Parameters

- **input_dim** (*int*) – dimensionality of the kernel, obligatory
- **variances** (*np.ndarray*) –
- **ARD** (*Boolean*) – Auto Relevance Determination (one lengthscale per dimension)

`GPy.kern.constructors.mlp(input_dim, variance=1.0, weight_variance=None, bias_variance=100.0, ARD=False)`

Construct an MLP kernel

Parameters

- **input_dim** (*int*) – dimensionality of the kernel, obligatory
- **variance** (*float*) – the variance of the kernel

- **weight_scale** (vector of weight variances for input weights in neural network (length 1 if kernel is isotropic)) – the lengthscale of the kernel
- **bias_variance** (float) – the variance of the biases in the neural network.
- **ARD** (Boolean) – Auto Relevance Determination (allows for ARD version of covariance)

`GPy.kern.constructors.ode1_eq(output_dim=1)`

Latent force model covariance, first order differential equation driven by exponentiated quadratic.

See N. D. Lawrence, G. Sanguinetti and M. Rattray. (2007) ‘Modelling transcriptional regulation using Gaussian processes’ in B. Schoelkopf, J. C. Platt and T. Hofmann (eds) *Advances in Neural Information Processing Systems*, MIT Press, Cambridge, MA, pp 785–792.

Parameters **output_dim** (int) – number of outputs in the covariance function.

`GPy.kern.constructors.periodic_Matern32(input_dim, variance=1.0, lengthscale=None, period=6.283185307179586, n_freq=10, lower=0.0, upper=12.566370614359172)`

Construct a periodic Matern 3/2 kernel.

Parameters

- **input_dim** (int) – dimensionality, only defined for input_dim=1
- **variance** (float) – the variance of the kernel
- **lengthscale** (float) – the lengthscale of the kernel
- **period** (float) – the period
- **n_freq** (int) – the number of frequencies considered for the periodic subspace

`GPy.kern.constructors.periodic_Matern52(input_dim, variance=1.0, lengthscale=None, period=6.283185307179586, n_freq=10, lower=0.0, upper=12.566370614359172)`

Construct a periodic Matern 5/2 kernel.

Parameters

- **input_dim** (int) – dimensionality, only defined for input_dim=1
- **variance** (float) – the variance of the kernel
- **lengthscale** (float) – the lengthscale of the kernel
- **period** (float) – the period
- **n_freq** (int) – the number of frequencies considered for the periodic subspace

`GPy.kern.constructors.periodic_exponential(input_dim=1, variance=1.0, lengthscale=None, period=6.283185307179586, n_freq=10, lower=0.0, upper=12.566370614359172)`

Construct an periodic exponential kernel

Parameters

- **input_dim** (int) – dimensionality, only defined for input_dim=1
- **variance** (float) – the variance of the kernel
- **lengthscale** (float) – the lengthscale of the kernel
- **period** (float) – the period
- **n_freq** (int) – the number of frequencies considered for the periodic subspace

`GPy.kern.constructors.poly(input_dim, variance=1.0, weight_variance=None, bias_variance=1.0, degree=2, ARD=False)`

Construct a polynomial kernel

Parameters

- **input_dim** (*int*) – dimensionality of the kernel, obligatory
- **variance** (*float*) – the variance of the kernel
- **weight_scale** (*vector of weight variances for input weights.*) – the lengthscale of the kernel
- **bias_variance** (*float*) – the variance of the biases.
- **degree** (*int*) – the degree of the polynomial
- **ARD** (*Boolean*) – Auto Relevance Determination (allows for ARD version of covariance)

`GPy.kern.constructors.prod(k1, k2, tensor=False)`

Construct a product kernel over input_dim from two kernels over input_dim

Parameters

- **k2** (*k1,*) – the kernels to multiply
- **tensor** (*Boolean*) – The kernels are either multiply as functions defined on the same input space (default) or on the product of the input spaces

Return type kernel object

`GPy.kern.constructors.rational_quadratic(input_dim, variance=1.0, lengthscale=1.0, power=1.0)`

Construct rational quadratic kernel.

Parameters

- **input_dim** (*int (input_dim=1 is the only value currently supported)*) – the number of input dimensions
- **variance** (*float*) – the variance σ^2
- **lengthscale** (*float*) – the lengthscale ℓ

Return type kern object

`GPy.kern.constructors.rbf(input_dim, variance=1.0, lengthscale=None, ARD=False)`

Construct an RBF kernel

Parameters

- **input_dim** (*int*) – dimensionality of the kernel, obligatory
- **variance** (*float*) – the variance of the kernel
- **lengthscale** (*float*) – the lengthscale of the kernel
- **ARD** (*Boolean*) – Auto Relevance Determination (one lengthscale per dimension)

`GPy.kern.constructors.rbf_inv(input_dim, variance=1.0, inv_lengthscale=None, ARD=False)`

Construct an RBF kernel

Parameters

- **input_dim** (*int*) – dimensionality of the kernel, obligatory

- **variance** (*float*) – the variance of the kernel
- **lengthscale** (*float*) – the lengthscale of the kernel
- **ARD** (*Boolean*) – Auto Relevance Determination (one lengthscale per dimension)

`GPy.kern.constructors.rbf_sympy` (*input_dim*, *ARD=False*, *variance=1.0*, *lengthscale=1.0*)
Radial Basis Function covariance.

`GPy.kern.constructors.rbfcos` (*input_dim*, *variance=1.0*, *frequencies=None*, *bandwidths=None*,
ARD=False)
construct a rbfcos kernel

`GPy.kern.constructors.spline` (*input_dim*, *variance=1.0*)
Construct a spline kernel.

Parameters

- **input_dim** (*int*) – Dimensionality of the kernel
- **variance** (*float*) – the variance of the kernel

`GPy.kern.constructors.symmetric` (*k*)
Construct a symmetric kernel from an existing kernel

The symmetric kernel works by adding two GP functions together, and computing the overall covariance.

Let $f \sim \text{GP}(x \mid 0, k(x, x'))$. Now let $g = f(x) + f(-x)$.

It's easy to see that g is a symmetric function: $g(x) = g(-x)$.

by construction, g , is a gaussian Process with mean 0 and covariance

$$k(x, x') + k(-x, x') + k(x, -x') + k(-x, -x')$$

This constructor builds a covariance function of this form from the initial kernel

`GPy.kern.constructors.sympykern` (*input_dim*, *k=None*, *output_dim=1*, *name=None*,
param=None)

A base kernel object, where all the hard work is done by sympy.

Parameters **k** (*a positive definite sympy function of x1, z1, x2, z2...*) – the covariance function

To construct a new sympy kernel, you'll need to define:

- a kernel function using a sympy object. Ensure that the kernel is of the form $k(x, z)$.
- that's it! we'll extract the variables from the function k .

Note:

- to handle multiple inputs, call them x_1, z_1 , etc
- to handle multiple correlated outputs, you'll need to define each covariance function and 'cross' variance function. TODO

`GPy.kern.constructors.white` (*input_dim*, *variance=1.0*)

Construct a white kernel.

Parameters

- **input_dim** (*int*) – dimensionality of the kernel, obligatory
- **variance** (*float*) – the variance of the kernel

GPy.kern.kern module

class GPy.kern.kern.**Kern_check_dK_dX** (*kernel=None, dL_dK=None, X=None, X2=None*)

Bases: GPy.kern.kern.Kern_check_model

This class allows gradient checks for the gradient of a kernel with respect to X.

class GPy.kern.kern.**Kern_check_dK_dtheta** (*kernel=None, dL_dK=None, X=None, X2=None*)

Bases: GPy.kern.kern.Kern_check_model

This class allows gradient checks for the gradient of a kernel with respect to parameters.

class GPy.kern.kern.**Kern_check_dKdiag_dX** (*kernel=None, dL_dK=None, X=None, X2=None*)

Bases: GPy.kern.kern.Kern_check_model

This class allows gradient checks for the gradient of a kernel diagonal with respect to X.

log_likelihood()

class GPy.kern.kern.**Kern_check_dKdiag_dtheta** (*kernel=None, dL_dK=None, X=None*)

Bases: GPy.kern.kern.Kern_check_model

This class allows gradient checks of the gradient of the diagonal of a kernel with respect to the parameters.

log_likelihood()

class GPy.kern.kern.**Kern_check_model** (*kernel=None, dL_dK=None, X=None, X2=None*)

Bases: GPy.core.model.Model

This is a dummy model class used as a base class for checking that the gradients of a given kernel are implemented correctly. It enables checkgradient() to be called independently on a kernel.

is_positive_definite()

log_likelihood()

class GPy.kern.kern.**kern** (*input_dim, parts=[], input_slices=None*)

Bases: GPy.core.parameterized.Parameterized

K (*X, X2=None, which_parts='all'*)

Compute the kernel function.

Parameters

- **X** – the first set of inputs to the kernel
- **X2** – (optional) the second set of arguments to the kernel. If X2 is None, this is passed through to the ‘part’ object, which handles this as X2 == X.
- **which_parts** – a list of booleans detailing whether to include each of the part functions. By default, ‘all’ indicates [True]*self.num_parts

Kdiag (*X, which_parts='all'*)

Compute the diagonal of the covariance function for inputs X.

add (*other, tensor=False*)

Add another kernel to this one.

If Tensor is False, both kernels are defined on the same `_space_`. then the created kernel will have the same number of inputs as self and other (which must be the same).

If Tensor is True, then the dimensions are stacked ‘horizontally’, so that the resulting kernel has `self.input_dim + other.input_dim`

Parameters **other** (*GPy.kern*) – the other kernel to be added

compute_param_slices()

Create a set of slices that can index the parameters of each part.

dK_dX (*dL_dK*, *X*, *X2=None*)

Compute the gradient of the objective function with respect to *X*.

Parameters

- **dL_dK** (*np.ndarray* (*num_samples* *x* *num_inducing*)) – An array of gradients of the objective function with respect to the covariance function.
- **X** (*np.ndarray* (*num_samples* *x* *input_dim*)) – Observed data inputs
- **X2** (*np.ndarray* (*num_inducing* *x* *input_dim*)) – Observed data inputs (optional, defaults to *X*)

dK_dtheta (*dL_dK*, *X*, *X2=None*)

Compute the gradient of the covariance function with respect to the parameters.

Parameters

- **dL_dK** (*np.ndarray* (*num_samples* *x* *num_inducing*)) – An array of gradients of the objective function with respect to the covariance function.
- **X** (*np.ndarray* (*num_samples* *x* *input_dim*)) – Observed data inputs
- **X2** (*np.ndarray* (*num_inducing* *x* *input_dim*)) – Observed data inputs (optional, defaults to *X*)

returns: *dL_dtheta*

dKdiag_dX (*dL_dKdiag*, *X*)

dKdiag_dtheta (*dL_dKdiag*, *X*)

Compute the gradient of the diagonal of the covariance function with respect to the parameters.

dpsi0_dZ (*dL_dpsi0*, *Z*, *mu*, *S*)

dpsi0_dmuS (*dL_dpsi0*, *Z*, *mu*, *S*)

dpsi0_dtheta (*dL_dpsi0*, *Z*, *mu*, *S*)

dpsi1_dZ (*dL_dpsi1*, *Z*, *mu*, *S*)

dpsi1_dmuS (*dL_dpsi1*, *Z*, *mu*, *S*)

return shapes are *num_samples*, *num_inducing*, *input_dim*

dpsi1_dtheta (*dL_dpsi1*, *Z*, *mu*, *S*)

dpsi2_dZ (*dL_dpsi2*, *Z*, *mu*, *S*)

dpsi2_dmuS (*dL_dpsi2*, *Z*, *mu*, *S*)

dpsi2_dtheta (*dL_dpsi2*, *Z*, *mu*, *S*)

getstate()

Get the current state of the class, here just all the indices, rest can get recomputed

plot (*x=None*, *plot_limits=None*, *which_parts='all'*, *resolution=None*, **args*, ***kwargs*)

plot_ARD (*fignum=None*, *ax=None*, *title=''*, *legend=False*)

If an ARD kernel is present, plot a bar representation using matplotlib

Parameters

- **fignum** – figure number of the plot
- **ax** – matplotlib axis to plot on

- **title** – title of the plot, pass ‘’ to not print a title pass None for a generic title

prod (*other, tensor=False*)

Multiply two kernels (either on the same space, or on the tensor product of the input space).

Parameters

- **other** (*GPy.kern*) – the other kernel to be added
- **tensor** (*bool*) – whether or not to use the tensor space (default is false).

psi0 (*Z, mu, S*)

psi1 (*Z, mu, S*)

psi2 (*Z, mu, S*)

Parameters

- **Z** – np.ndarray of inducing inputs (M x Q)
- **S** (*mu,*) – np.ndarrays of means and variances (each N x Q)

Returns psi2 np.ndarray (N,M,M)

setstate (*state*)

`GPy.kern.kern.kern_test` (*kern, X=None, X2=None, output_ind=None, verbose=False, X_positive=False*)

This function runs on kernels to check the correctness of their implementation. It checks that the covariance function is positive definite for a randomly generated data set.

Parameters

- **kern** (*GPy.kern.Kernpart*) – the kernel to be tested.
- **X** (*ndarray*) – X input values to test the covariance function.
- **X2** (*ndarray*) – X2 input values to test the covariance function.

Module contents

GPy.likelihoods package

Subpackages

GPy.likelihoods.noise_models package

Submodules

GPy.likelihoods.noise_models.bernoulli_noise module

class `GPy.likelihoods.noise_models.bernoulli_noise.Bernoulli` (*gp_link=None, analytical_mean=False, analytical_variance=False*)

Bases: `GPy.likelihoods.noise_models.noise_distributions.NoiseDistribution`

Bernoulli likelihood

$$p(y_i|\lambda(f_i)) = \lambda(f_i)^{y_i} (1 - f_i)^{1-y_i}$$

Note: Y is expected to take values in {-1,1} Probit likelihood usually used

d2logpdf_dlink2 (*link_f*, *y*, *extra_data=None*)

Hessian at *y*, given *link_f*, w.r.t *link_f* the hessian will be 0 unless *i == j* i.e. second derivative logpdf at *y* given *link(f_i)* *link(f_j)* w.r.t *link(f_i)* and *link(f_j)*

$$\frac{d^2 \ln p(y_i | \lambda(f_i))}{d\lambda(f)^2} = \frac{-y_i}{\lambda(f)^2} - \frac{(1 - y_i)}{(1 - \lambda(f))^2}$$

Parameters

- **link_f** (*Nx1 array*) – latent variables *link(f)*
- **y** (*Nx1 array*) – data
- **extra_data** – *extra_data* not used in bernoulli

Returns Diagonal of log hessian matrix (second derivative of log likelihood evaluated at points *link(f)*)

Return type *Nx1 array*

Note: Will return diagonal of hessian, since every where else it is 0, as the likelihood factorizes over cases (the distribution for *y_i* depends only on *link(f_i)* not on *link(f_{j!=i})*)

d3logpdf_dlink3 (*link_f*, *y*, *extra_data=None*)

Third order derivative log-likelihood function at *y* given *link(f)* w.r.t *link(f)*

$$\frac{d^3 \ln p(y_i | \lambda(f_i))}{d^3 \lambda(f)} = \frac{2y_i}{\lambda(f)^3} - \frac{2(1 - y_i)}{(1 - \lambda(f))^3}$$

Parameters

- **link_f** (*Nx1 array*) – latent variables *link(f)*
- **y** (*Nx1 array*) – data
- **extra_data** – *extra_data* not used in bernoulli

Returns third derivative of log likelihood evaluated at points *link(f)*

Return type *Nx1 array*

dlogpdf_dlink (*link_f*, *y*, *extra_data=None*)

Gradient of the pdf at *y*, given *link(f)* w.r.t *link(f)*

$$\frac{d \ln p(y_i | \lambda(f_i))}{d\lambda(f)} = \frac{y_i}{\lambda(f_i)} - \frac{(1 - y_i)}{(1 - \lambda(f_i))}$$

Parameters

- **link_f** (*Nx1 array*) – latent variables *link(f)*
- **y** (*Nx1 array*) – data
- **extra_data** – *extra_data* not used in bernoulli

Returns gradient of log likelihood evaluated at points *link(f)*

Return type *Nx1 array*

logpdf_link (*link_f*, *y*, *extra_data=None*)

Log Likelihood function given link(f)

$$\ln p(y_i|\lambda(f_i)) = y_i \log \lambda(f_i) + (1 - y_i) \log(1 - f_i)$$

Parameters

- **link_f** (*Nx1 array*) – latent variables link(f)
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data not used in bernoulli

Returns log likelihood evaluated at points link(f)

Return type float

pdf_link (*link_f*, *y*, *extra_data=None*)

Likelihood function given link(f)

$$p(y_i|\lambda(f_i)) = \lambda(f_i)^{y_i} (1 - f_i)^{1-y_i}$$

Parameters

- **link_f** (*Nx1 array*) – latent variables link(f)
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data not used in bernoulli

Returns likelihood evaluated for this point

Return type float

samples (*gp*)

Returns a set of samples of observations based on a given value of the latent variable.

Parameters **gp** – latent variable

GPy.likelihoods.noise_models.exponential_noise module

class GPy.likelihoods.noise_models.exponential_noise.**Exponential** (*gp_link=None*,
*analyti-
cal_mean=False*,
*analyti-
cal_variance=False*)

Bases: GPy.likelihoods.noise_models.noise_distributions.NoiseDistribution

Exponential likelihood Y is expected to take values in {0,1,2,...} — $L(x) = \exp(\lambda) * \lambda^{Y_i} / Y_i!$

d2logpdf_dlink2 (*link_f*, *y*, *extra_data=None*)

Hessian at y, given link(f), w.r.t link(f) i.e. second derivative logpdf at y given link(f_i) and link(f_j) w.r.t link(f_i) and link(f_j) The hessian will be 0 unless i == j

$$\frac{d^2 \ln p(y_i|\lambda(f_i))}{d^2 \lambda(f)} = -\frac{1}{\lambda(f_i)^2}$$

Parameters

- **link_f** (*Nx1 array*) – latent variables link(f)
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data which is not used in exponential distribution

Returns Diagonal of hessian matrix (second derivative of likelihood evaluated at points f)

Return type Nx1 array

Note: Will return diagonal of hessian, since every where else it is 0, as the likelihood factorizes over cases (the distribution for y_i depends only on $\text{link}(f_i)$ not on $\text{link}(f_{j \neq i})$)

d3logpdf_dlink3 (*link_f*, *y*, *extra_data=None*)

Third order derivative log-likelihood function at y given $\text{link}(f)$ w.r.t $\text{link}(f)$

$$\frac{d^3 \ln p(y_i | \lambda(f_i))}{d^3 \lambda(f)} = \frac{2}{\lambda(f_i)^3}$$

Parameters

- **link_f** (*Nx1 array*) – latent variables $\text{link}(f)$
- **y** (*Nx1 array*) – data
- **extra_data** – *extra_data* which is not used in exponential distribution

Returns third derivative of likelihood evaluated at points f

Return type Nx1 array

dlogpdf_dlink (*link_f*, *y*, *extra_data=None*)

Gradient of the log likelihood function at y , given $\text{link}(f)$ w.r.t $\text{link}(f)$

$$\frac{d \ln p(y_i | \lambda(f_i))}{d \lambda(f)} = \frac{1}{\lambda(f)} - y_i$$

Parameters

- **link_f** (*Nx1 array*) – latent variables (f)
- **y** (*Nx1 array*) – data
- **extra_data** – *extra_data* which is not used in exponential distribution

Returns gradient of likelihood evaluated at points

Return type Nx1 array

logpdf_link (*link_f*, *y*, *extra_data=None*)

Log Likelihood Function given $\text{link}(f)$

$$\ln p(y_i | \lambda(f_i)) = \ln \lambda(f_i) - y_i \lambda(f_i)$$

Parameters

- **link_f** (*Nx1 array*) – latent variables ($\text{link}(f)$)
- **y** (*Nx1 array*) – data
- **extra_data** – *extra_data* which is not used in exponential distribution

Returns likelihood evaluated for this point

Return type float

pdf_link (*link_f*, *y*, *extra_data=None*)

Likelihood function given $\text{link}(f)$

$$p(y_i | \lambda(f_i)) = \lambda(f_i) \exp(-y_i \lambda(f_i))$$

Parameters

- **link_f** (*Nx1 array*) – latent variables link(f)
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data which is not used in exponential distribution

Returns likelihood evaluated for this point

Return type float

samples (*gp*)

Returns a set of samples of observations based on a given value of the latent variable.

Parameters **gp** – latent variable

GPy.likelihoods.noise_models.gamma_noise module

class GPy.likelihoods.noise_models.gamma_noise.**Gamma** (*gp_link=None, analyti-
cal_mean=False, analyti-
cal_variance=False, beta=1.0*)

Bases: GPy.likelihoods.noise_models.noise_distributions.NoiseDistribution

Gamma likelihood

$$p(y_i|\lambda(f_i)) = \frac{\beta^{\alpha_i}}{\Gamma(\alpha_i)} y_i^{\alpha_i-1} e^{-\beta y_i}$$

$$\alpha_i = \beta y_i$$

d2logpdf_dlink2 (*link_f, y, extra_data=None*)

Hessian at y, given link(f), w.r.t link(f) i.e. second derivative logpdf at y given link(f_i) and link(f_j) w.r.t link(f_i) and link(f_j) The hessian will be 0 unless i == j

$$\frac{d^2 \ln p(y_i|\lambda(f_i))}{d^2 \lambda(f)} = -\beta^2 \frac{d^2 \Psi(\alpha_i)}{d\alpha_i}$$

$$\alpha_i = \beta y_i$$

Parameters

- **link_f** (*Nx1 array*) – latent variables link(f)
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data which is not used in gamma distribution

Returns Diagonal of hessian matrix (second derivative of likelihood evaluated at points f)

Return type Nx1 array

Note: Will return diagonal of hessian, since every where else it is 0, as the likelihood factorizes over cases (the distribution for y_i depends only on link(f_i) not on link(f_(j!=i)))

d3logpdf_dlink3 (*link_f, y, extra_data=None*)

Third order derivative log-likelihood function at y given link(f) w.r.t link(f)

$$\frac{d^3 \ln p(y_i|\lambda(f_i))}{d^3 \lambda(f)} = -\beta^3 \frac{d^3 \Psi(\alpha_i)}{d\alpha_i}$$

$$\alpha_i = \beta y_i$$

Parameters

- **link_f** (*Nx1 array*) – latent variables link(f)
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data which is not used in gamma distribution

Returns third derivative of likelihood evaluated at points f

Return type Nx1 array

dlogpdf_dlink (*link_f, y, extra_data=None*)

Gradient of the log likelihood function at y, given link(f) w.r.t link(f)

$$\frac{d \ln p(y_i | \lambda(f_i))}{d \lambda(f)} = \beta (\log \beta y_i) - \Psi(\alpha_i) \beta$$

$$\alpha_i = \beta y_i$$

Parameters

- **link_f** (*Nx1 array*) – latent variables (f)
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data which is not used in gamma distribution

Returns gradient of likelihood evaluated at points

Return type Nx1 array

logpdf_link (*link_f, y, extra_data=None*)

Log Likelihood Function given link(f)

$$\ln p(y_i | \lambda(f_i)) = \alpha_i \log \beta - \log \Gamma(\alpha_i) + (\alpha_i - 1) \log y_i - \beta y_i$$

$$\alpha_i = \beta y_i$$

Parameters

- **link_f** (*Nx1 array*) – latent variables (link(f))
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data which is not used in poisson distribution

Returns likelihood evaluated for this point

Return type float

pdf_link (*link_f, y, extra_data=None*)

Likelihood function given link(f)

$$p(y_i | \lambda(f_i)) = \frac{\beta^{\alpha_i}}{\Gamma(\alpha_i)} y_i^{\alpha_i - 1} e^{-\beta y_i}$$

$$\alpha_i = \beta y_i$$

Parameters

- **link_f** (*Nx1 array*) – latent variables link(f)
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data which is not used in poisson distribution

Returns likelihood evaluated for this point

Return type float

GPy.likelihoods.noise_models.gaussian_noise module

class GPy.likelihoods.noise_models.gaussian_noise.**Gaussian** (*gp_link=None, analytical_mean=False, analytical_variance=False, variance=1.0, D=None, N=None*)

Bases: GPy.likelihoods.noise_models.noise_distributions.NoiseDistribution

Gaussian likelihood

$$\ln p(y_i|\lambda(f_i)) = -\frac{N \ln 2\pi}{2} - \frac{\ln |K|}{2} - \frac{(y_i - \lambda(f_i))^T \sigma^{-2} (y_i - \lambda(f_i))}{2}$$

Parameters

- **variance** – variance value of the Gaussian distribution
- **N** (*int*) – Number of data points

d2logpdf_dlink2 (*link_f, y, extra_data=None*)

Hessian at y, given link_f, w.r.t link_f. i.e. second derivative logpdf at y given link(f_i) link(f_j) w.r.t link(f_i) and link(f_j)

The hessian will be 0 unless i == j

$$\frac{d^2 \ln p(y_i|\lambda(f_i))}{d^2 f} = -\frac{1}{\sigma^2}$$

Parameters

- **link_f** (*Nx1 array*) – latent variables link(f)
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data not used in gaussian

Returns Diagonal of log hessian matrix (second derivative of log likelihood evaluated at points link(f))

Return type Nx1 array

Note: Will return diagonal of hessian, since every where else it is 0, as the likelihood factorizes over cases (the distribution for y_i depends only on link(f_i) not on link(f_j!=i))

d2logpdf_dlink2_dtheta (*f, y, extra_data=None*)

d2logpdf_dlink2_dvar (*link_f, y, extra_data=None*)

Gradient of the hessian (d2logpdf_dlink2) w.r.t variance parameter (noise_variance)

$$\frac{d}{d\sigma^2} \left(\frac{d^2 \ln p(y_i|\lambda(f_i))}{d^2 \lambda(f)} \right) = \frac{1}{\sigma^4}$$

Parameters

- **link_f** (*Nx1 array*) – latent variables link(f)
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data not used in gaussian

Returns derivative of log hessian evaluated at points link(f_i) and link(f_j) w.r.t variance parameter

Return type Nx1 array

d3logpdf_dlink3 (*link_f*, *y*, *extra_data=None*)

Third order derivative log-likelihood function at *y* given *link(f)* w.r.t *link(f)*

$$\frac{d^3 \ln p(y_i | \lambda(f_i))}{d^3 \lambda(f)} = 0$$

Parameters

- **link_f** (*Nx1 array*) – latent variables *link(f)*
- **y** (*Nx1 array*) – data
- **extra_data** – *extra_data* not used in gaussian

Returns third derivative of log likelihood evaluated at points *link(f)*

Return type *Nx1 array*

dlogpdf_dlink (*link_f*, *y*, *extra_data=None*)

Gradient of the pdf at *y*, given *link(f)* w.r.t *link(f)*

$$\frac{d \ln p(y_i | \lambda(f_i))}{d \lambda(f)} = \frac{1}{\sigma^2} (y_i - \lambda(f_i))$$

Parameters

- **link_f** (*Nx1 array*) – latent variables *link(f)*
- **y** (*Nx1 array*) – data
- **extra_data** – *extra_data* not used in gaussian

Returns gradient of log likelihood evaluated at points *link(f)*

Return type *Nx1 array*

dlogpdf_dlink_dtheta (*f*, *y*, *extra_data=None*)

dlogpdf_dlink_dvar (*link_f*, *y*, *extra_data=None*)

Derivative of the *dlogpdf_dlink* w.r.t variance parameter (*noise_variance*)

$$\frac{d}{d\sigma^2} \left(\frac{d \ln p(y_i | \lambda(f_i))}{d \lambda(f)} \right) = \frac{1}{\sigma^4} (-y_i + \lambda(f_i))$$

Parameters

- **link_f** (*Nx1 array*) – latent variables *link(f)*
- **y** (*Nx1 array*) – data
- **extra_data** – *extra_data* not used in gaussian

Returns derivative of log likelihood evaluated at points *link(f)* w.r.t variance parameter

Return type *Nx1 array*

dlogpdf_link_dtheta (*f*, *y*, *extra_data=None*)

dlogpdf_link_dvar (*link_f*, *y*, *extra_data=None*)

Gradient of the log-likelihood function at *y* given *link(f)*, w.r.t variance parameter (*noise_variance*)

$$\frac{d \ln p(y_i | \lambda(f_i))}{d \sigma^2} = -\frac{N}{2\sigma^2} + \frac{(y_i - \lambda(f_i))^2}{2\sigma^4}$$

Parameters

- **link_f** (*Nx1 array*) – latent variables *link(f)*

- **y** (*Nx1 array*) – data
- **extra_data** – extra_data not used in gaussian

Returns derivative of log likelihood evaluated at points link(f) w.r.t variance parameter

Return type float

logpdf_link (*link_f, y, extra_data=None*)

Log likelihood function given link(f)

$$\ln p(y_i | \lambda(f_i)) = -\frac{N \ln 2\pi}{2} - \frac{\ln |K|}{2} - \frac{(y_i - \lambda(f_i))^T \sigma^{-2} (y_i - \lambda(f_i))}{2}$$

Parameters

- **link_f** (*Nx1 array*) – latent variables link(f)
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data not used in gaussian

Returns log likelihood evaluated for this point

Return type float

pdf_link (*link_f, y, extra_data=None*)

Likelihood function given link(f)

$$\ln p(y_i | \lambda(f_i)) = -\frac{N \ln 2\pi}{2} - \frac{\ln |K|}{2} - \frac{(y_i - \lambda(f_i))^T \sigma^{-2} (y_i - \lambda(f_i))}{2}$$

Parameters

- **link_f** (*Nx1 array*) – latent variables link(f)
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data not used in gaussian

Returns likelihood evaluated for this point

Return type float

samples (*gp*)

Returns a set of samples of observations based on a given value of the latent variable.

Parameters **gp** – latent variable

GPy.likelihoods.noise_models.gp_transformations module

class GPy.likelihoods.noise_models.gp_transformations.**GPTransformation**

Bases: object

Link function class for doing non-Gaussian likelihoods approximation

Parameters **Y** – observed output (Nx1 numpy.darray)

Note: Y values allowed depend on the likelihood_function used

d2transf_df2 (*f*)

second derivative of transf(f) w.r.t. f

d3transf_df3 (*f*)

third derivative of transf(f) w.r.t. f

dtransf_df (*f*)
derivative of transf(*f*) w.r.t. *f*

transf (*f*)
Gaussian process tranformation function, latent space -> output space

class GPy.likelihoods.noise_models.gp_transformations.**Heaviside**
Bases: GPy.likelihoods.noise_models.gp_transformations.GPTransformation

$$g(f) = I_{x \in A}$$

d2transf_df2 (*f*)

dtransf_df (*f*)

transf (*f*)

class GPy.likelihoods.noise_models.gp_transformations.**Identity**
Bases: GPy.likelihoods.noise_models.gp_transformations.GPTransformation

$$g(f) = f$$

d2transf_df2 (*f*)

d3transf_df3 (*f*)

dtransf_df (*f*)

transf (*f*)

class GPy.likelihoods.noise_models.gp_transformations.**Log**
Bases: GPy.likelihoods.noise_models.gp_transformations.GPTransformation

$$g(f) = \log(\mu)$$

d2transf_df2 (*f*)

d3transf_df3 (*f*)

dtransf_df (*f*)

transf (*f*)

class GPy.likelihoods.noise_models.gp_transformations.**Log_ex_1**
Bases: GPy.likelihoods.noise_models.gp_transformations.GPTransformation

$$g(f) = \log(\exp(\mu) - 1)$$

d2transf_df2 (*f*)

d3transf_df3 (*f*)

dtransf_df (*f*)

transf (*f*)

class GPy.likelihoods.noise_models.gp_transformations.**Probit**

Bases: GPy.likelihoods.noise_models.gp_transformations.GPTransformation

$$g(f) = \Phi^{-1}(\mu)$$

d2transf_df2(f)

d3transf_df3(f)

dtransf_df(f)

transf(f)

class GPy.likelihoods.noise_models.gp_transformations.**Reciprocal**

Bases: GPy.likelihoods.noise_models.gp_transformations.GPTransformation

d2transf_df2(f)

d3transf_df3(f)

dtransf_df(f)

transf(f)

GPy.likelihoods.noise_models.noise_distributions module

class GPy.likelihoods.noise_models.noise_distributions.**NoiseDistribution**(gp_link,
 ana-
 lyti-
 cal_mean=False,
 ana-
 lyti-
 cal_variance=False)

Bases: object

Likelihood class for doing approximations

d2logpdf_df2(f, y, extra_data=None)

Evaluates the link function link(f) then computes the second derivative of log likelihood using it Uses the Faa di Bruno's formula for the chain rule

$$\frac{d^2 \log p(y|\lambda(f))}{df^2} = \frac{d^2 \log p(y|\lambda(f))}{d^2 \lambda(f)} \left(\frac{d\lambda(f)}{df} \right)^2 + \frac{d \log p(y|\lambda(f))}{d\lambda(f)} \frac{d^2 \lambda(f)}{df^2}$$

Parameters

- **f** (Nx1 array) – latent variables f
- **y** (Nx1 array) – data
- **extra_data** – extra_data which is not used in student t distribution - not used

Returns second derivative of log likelihood evaluated for this point (diagonal only)

Return type 1xN array

d2logpdf_df2_dtheta(f, y, extra_data=None)

TODO: Doc strings

d2logpdf_dlink2(link_f, y, extra_data=None)

d2logpdf_dlink2_dtheta(link_f, y, extra_data=None)

d3logpdf_df3 (*f*, *y*, *extra_data=None*)

Evaluates the link function $\text{link}(f)$ then computes the third derivative of log likelihood using it Uses the Faa di Bruno's formula for the chain rule

$$\frac{d^3 \log p(y|\lambda(f))}{df^3} = \frac{d^3 \log p(y|\lambda(f))}{d\lambda(f)^3} \left(\frac{d\lambda(f)}{df} \right)^3 + 3 \frac{d^2 \log p(y|\lambda(f))}{d\lambda(f)^2} \frac{d\lambda(f)}{df} \frac{d^2 \lambda(f)}{df^2} + \frac{d \log p(y|\lambda(f))}{d\lambda(f)} \frac{d^3 \lambda(f)}{df^3}$$

Parameters

- **f** (*Nx1 array*) – latent variables *f*
- **y** (*Nx1 array*) – data
- **extra_data** – *extra_data* which is not used in student t distribution - not used

Returns third derivative of log likelihood evaluated for this point

Return type float

d3logpdf_dlink3 (*link_f*, *y*, *extra_data=None*)

dlogpdf_df (*f*, *y*, *extra_data=None*)

Evaluates the link function $\text{link}(f)$ then computes the derivative of log likelihood using it Uses the Faa di Bruno's formula for the chain rule

$$\frac{d \log p(y|\lambda(f))}{df} = \frac{d \log p(y|\lambda(f))}{d\lambda(f)} \frac{d\lambda(f)}{df}$$

Parameters

- **f** (*Nx1 array*) – latent variables *f*
- **y** (*Nx1 array*) – data
- **extra_data** – *extra_data* which is not used in student t distribution - not used

Returns derivative of log likelihood evaluated for this point

Return type 1xN array

dlogpdf_df_dtheta (*f*, *y*, *extra_data=None*)

TODO: Doc strings

dlogpdf_dlink (*link_f*, *y*, *extra_data=None*)

dlogpdf_dlink_dtheta (*link_f*, *y*, *extra_data=None*)

dlogpdf_dtheta (*f*, *y*, *extra_data=None*)

TODO: Doc strings

dlogpdf_link_dtheta (*link_f*, *y*, *extra_data=None*)

log_predictive_density (*y_test*, *mu_star*, *var_star*)

Calculation of the log predictive density

Parameters

- **y_test** (*(Nx1 array)*) – test observations ($y_{\{*\}}$)
- **mu_star** (*(Nx1 array)*) – predictive mean of gaussian $p(f_{\{*\}}|\mu_{\{*\}}, \text{var}_{\{*\}})$
- **var_star** (*(Nx1 array)*) – predictive variance of gaussian $p(f_{\{*\}}|\mu_{\{*\}}, \text{var}_{\{*\}})$

logpdf (*f*, *y*, *extra_data=None*)

Evaluates the link function $\text{link}(f)$ then computes the log likelihood (log pdf) using it

Parameters

- **f** (*Nx1 array*) – latent variables f
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data which is not used in student t distribution - not used

Returns log likelihood evaluated for this point

Return type float

logpdf_link (*link_f, y, extra_data=None*)

pdf (*f, y, extra_data=None*)

Evaluates the link function link(f) then computes the likelihood (pdf) using it

Parameters

- **f** (*Nx1 array*) – latent variables f
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data which is not used in student t distribution - not used

Returns likelihood evaluated for this point

Return type float

pdf_link (*link_f, y, extra_data=None*)

predictive_values (*mu, var, full_cov=False, sampling=False, num_samples=10000*)

Compute mean, variance and confidence interval (percentiles 5 and 95) of the prediction.

Parameters

- **mu** – mean of the latent variable, f, of posterior
- **var** – variance of the latent variable, f, of posterior
- **full_cov** (*Boolean*) – whether to use the full covariance or just the diagonal
- **num_samples** (*integer*) – number of samples to use in computing quantiles and possibly mean variance
- **sampling** (*Boolean*) – Whether to use samples for mean and variances anyway

samples (*gp*)

Returns a set of samples of observations based on a given value of the latent variable.

Parameters **gp** – latent variable

GPy.likelihoods.noise_models.poisson_noise module

class GPy.likelihoods.noise_models.poisson_noise.**Poisson** (*gp_link=None, analytical_mean=False, analytical_variance=False*)

Bases: GPy.likelihoods.noise_models.noise_distributions.NoiseDistribution

Poisson likelihood

$$p(y_i | \lambda(f_i)) = \frac{\lambda(f_i)^{y_i}}{y_i!} e^{-\lambda(f_i)}$$

Note: Y is expected to take values in {0,1,2,...}

d2logpdf_dlink2 (*link_f*, *y*, *extra_data=None*)

Hessian at *y*, given *link(f)*, w.r.t *link(f)* i.e. second derivative logpdf at *y* given *link(f_i)* and *link(f_j)* w.r.t *link(f_i)* and *link(f_j)* The hessian will be 0 unless *i == j*

$$\frac{d^2 \ln p(y_i | \lambda(f_i))}{d^2 \lambda(f)} = \frac{-y_i}{\lambda(f_i)^2}$$

Parameters

- **link_f** (*Nx1 array*) – latent variables *link(f)*
- **y** (*Nx1 array*) – data
- **extra_data** – *extra_data* which is not used in poisson distribution

Returns Diagonal of hessian matrix (second derivative of likelihood evaluated at points *f*)

Return type *Nx1 array*

Note: Will return diagonal of hessian, since every where else it is 0, as the likelihood factorizes over cases (the distribution for *y_i* depends only on *link(f_i)* not on *link(f_(j!=i))*)

d3logpdf_dlink3 (*link_f*, *y*, *extra_data=None*)

Third order derivative log-likelihood function at *y* given *link(f)* w.r.t *link(f)*

$$\frac{d^3 \ln p(y_i | \lambda(f_i))}{d^3 \lambda(f)} = \frac{2y_i}{\lambda(f_i)^3}$$

Parameters

- **link_f** (*Nx1 array*) – latent variables *link(f)*
- **y** (*Nx1 array*) – data
- **extra_data** – *extra_data* which is not used in poisson distribution

Returns third derivative of likelihood evaluated at points *f*

Return type *Nx1 array*

dlogpdf_dlink (*link_f*, *y*, *extra_data=None*)

Gradient of the log likelihood function at *y*, given *link(f)* w.r.t *link(f)*

$$\frac{d \ln p(y_i | \lambda(f_i))}{d \lambda(f)} = \frac{y_i}{\lambda(f_i)} - 1$$

Parameters

- **link_f** (*Nx1 array*) – latent variables (*f*)
- **y** (*Nx1 array*) – data
- **extra_data** – *extra_data* which is not used in poisson distribution

Returns gradient of likelihood evaluated at points

Return type *Nx1 array*

logpdf_link (*link_f*, *y*, *extra_data=None*)

Log Likelihood Function given *link(f)*

$$\ln p(y_i | \lambda(f_i)) = -\lambda(f_i) + y_i \log \lambda(f_i) - \log y_i!$$

Parameters

- **link_f** (*Nx1 array*) – latent variables (link(f))
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data which is not used in poisson distribution

Returns likelihood evaluated for this point

Return type float

pdf_link (*link_f, y, extra_data=None*)
Likelihood function given link(f)

$$p(y_i|\lambda(f_i)) = \frac{\lambda(f_i)^{y_i}}{y_i!} e^{-\lambda(f_i)}$$

Parameters

- **link_f** (*Nx1 array*) – latent variables link(f)
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data which is not used in poisson distribution

Returns likelihood evaluated for this point

Return type float

samples (*gp*)

Returns a set of samples of observations based on a given value of the latent variable.

Parameters **gp** – latent variable

GPy.likelihoods.noise_models.student_t_noise module

class GPy.likelihoods.noise_models.student_t_noise.**StudentT** (*gp_link=None, analytical_mean=True, analytical_variance=True, deg_free=5, sigma2=2*)

Bases: GPy.likelihoods.noise_models.noise_distributions.NoiseDistribution

Student T likelihood

For nonmanclature see Bayesian Data Analysis 2003 p576

$$p(y_i|\lambda(f_i)) = \frac{\Gamma\left(\frac{v+1}{2}\right)}{\Gamma\left(\frac{v}{2}\right)\sqrt{v\pi\sigma^2}} \left(1 + \frac{1}{v} \left(\frac{(y_i - f_i)^2}{\sigma^2}\right)\right)^{-\frac{v+1}{2}}$$

d2logpdf_dlink2 (*link_f, y, extra_data=None*)

Hessian at y, given link(f), w.r.t link(f) i.e. second derivative logpdf at y given link(f_i) and link(f_j) w.r.t link(f_i) and link(f_j) The hessian will be 0 unless i == j

$$\frac{d^2 \ln p(y_i|\lambda(f_i))}{d^2 \lambda(f)} = \frac{(v+1)((y_i - \lambda(f_i))^2 - \sigma^2 v)}{((y_i - \lambda(f_i))^2 + \sigma^2 v)^2}$$

Parameters

- **link_f** (*Nx1 array*) – latent variables link(f)
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data which is not used in student t distribution

Returns Diagonal of hessian matrix (second derivative of likelihood evaluated at points f)

Return type Nx1 array

Note: Will return diagonal of hessian, since every where else it is 0, as the likelihood factorizes over cases (the distribution for y_i depends only on $\text{link}(f_i)$ not on $\text{link}(f_{(j \neq i)})$)

d2logpdf_dlink2_dtheta ($f, y, \text{extra_data}=\text{None}$)

d2logpdf_dlink2_dvar ($\text{link_f}, y, \text{extra_data}=\text{None}$)

Gradient of the hessian (d2logpdf_dlink2) w.r.t variance parameter (t_{noise})

$$\frac{d}{d\sigma^2} \left(\frac{d^2 \ln p(y_i | \lambda(f_i))}{d^2 f} \right) = \frac{v(v+1)(\sigma^2 v - 3(y_i - \lambda(f_i))^2)}{(\sigma^2 v + (y_i - \lambda(f_i))^2)^3}$$

Parameters

- **link_f** ($N \times 1$ array) – latent variables $\text{link}(f)$
- **y** ($N \times 1$ array) – data
- **extra_data** – extra_data which is not used in student t distribution

Returns derivative of hessian evaluated at points f and f_j w.r.t variance parameter

Return type Nx1 array

d3logpdf_dlink3 ($\text{link_f}, y, \text{extra_data}=\text{None}$)

Third order derivative log-likelihood function at y given $\text{link}(f)$ w.r.t $\text{link}(f)$

$$\frac{d^3 \ln p(y_i | \lambda(f_i))}{d^3 \lambda(f)} = \frac{-2(v+1)((y_i - \lambda(f_i))^3 - 3(y_i - \lambda(f_i))\sigma^2 v)}{((y_i - \lambda(f_i)) + \sigma^2 v)^3}$$

Parameters

- **link_f** ($N \times 1$ array) – latent variables $\text{link}(f)$
- **y** ($N \times 1$ array) – data
- **extra_data** – extra_data which is not used in student t distribution

Returns third derivative of likelihood evaluated at points f

Return type Nx1 array

dlogpdf_dlink ($\text{link_f}, y, \text{extra_data}=\text{None}$)

Gradient of the log likelihood function at y , given $\text{link}(f)$ w.r.t $\text{link}(f)$

$$\frac{d \ln p(y_i | \lambda(f_i))}{d \lambda(f)} = \frac{(v+1)(y_i - \lambda(f_i))}{(y_i - \lambda(f_i))^2 + \sigma^2 v}$$

Parameters

- **link_f** ($N \times 1$ array) – latent variables (f)
- **y** ($N \times 1$ array) – data
- **extra_data** – extra_data which is not used in student t distribution

Returns gradient of likelihood evaluated at points

Return type Nx1 array

dlogpdf_dlink_dtheta ($f, y, \text{extra_data}=\text{None}$)

dlogpdf_dlink_dvar (*link_f*, *y*, *extra_data=None*)

Derivative of the dlogpdf_dlink w.r.t variance parameter (*t_noise*)

$$\frac{d}{d\sigma^2} \left(\frac{d \ln p(y_i | \lambda(f_i))}{df} \right) = \frac{-2\sigma v(v+1)(y_i - \lambda(f_i))}{(y_i - \lambda(f_i))^2 + \sigma^2 v}$$

Parameters

- **link_f** (*Nx1 array*) – latent variables link_f
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data which is not used in student t distribution

Returns derivative of likelihood evaluated at points f w.r.t variance parameter

Return type *Nx1 array*

dlogpdf_link_dtheta (*f*, *y*, *extra_data=None*)

dlogpdf_link_dvar (*link_f*, *y*, *extra_data=None*)

Gradient of the log-likelihood function at y given f, w.r.t variance parameter (*t_noise*)

$$\frac{d \ln p(y_i | \lambda(f_i))}{d\sigma^2} = \frac{v((y_i - \lambda(f_i))^2 - \sigma^2)}{2\sigma^2(\sigma^2 v + (y_i - \lambda(f_i))^2)}$$

Parameters

- **link_f** (*Nx1 array*) – latent variables link(f)
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data which is not used in student t distribution

Returns derivative of likelihood evaluated at points f w.r.t variance parameter

Return type float

logpdf_link (*link_f*, *y*, *extra_data=None*)

Log Likelihood Function given link(f)

$$\ln p(y_i | \lambda(f_i)) = \ln \Gamma\left(\frac{v+1}{2}\right) - \ln \Gamma\left(\frac{v}{2}\right) - \ln \sqrt{v\pi\sigma^2} - \frac{v+1}{2} \ln \left(1 + \frac{1}{v} \left(\frac{(y_i - \lambda(f_i))^2}{\sigma^2}\right)\right)$$

Parameters

- **link_f** (*Nx1 array*) – latent variables (link(f))
- **y** (*Nx1 array*) – data
- **extra_data** – extra_data which is not used in student t distribution

Returns likelihood evaluated for this point

Return type float

pdf_link (*link_f*, *y*, *extra_data=None*)

Likelihood function given link(f)

$$p(y_i | \lambda(f_i)) = \frac{\Gamma\left(\frac{v+1}{2}\right)}{\Gamma\left(\frac{v}{2}\right) \sqrt{v\pi\sigma^2}} \left(1 + \frac{1}{v} \left(\frac{(y_i - \lambda(f_i))^2}{\sigma^2}\right)\right)^{-\frac{v+1}{2}}$$

Parameters

- **link_f** (*Nx1 array*) – latent variables link(f)

- **y** (*NxI array*) – data
- **extra_data** – extra_data which is not used in student t distribution

Returns likelihood evaluated for this point

Return type float

samples (*gp*)

Returns a set of samples of observations based on a given value of the latent variable.

Parameters **gp** – latent variable

variance

Module contents

Submodules

GPy.likelihoods.ep module

class GPy.likelihoods.ep.**EP** (*data, noise_model*)

Bases: GPy.likelihoods.likelihood.likelihood

fit_DTC (*Kmm, Kmn, epsilon=0.001, power_ep=[1.0, 1.0]*)

The expectation-propagation algorithm with sparse pseudo-input. For nomenclature see ... 2013.

Parameters

- **epsilon** (*float*) – Convergence criterion, maximum squared difference allowed between mean updates to stop iterations (float)
- **power_ep** (*list of floats*) – Power EP parameters

fit_FITC (*Kmm, Kmn, Knn_diag, epsilon=0.001, power_ep=[1.0, 1.0]*)

The expectation-propagation algorithm with sparse pseudo-input. For nomenclature see Naish-Guzman and Holden, 2008.

Parameters

- **epsilon** (*float*) – Convergence criterion, maximum squared difference allowed between mean updates to stop iterations (float)
- **power_ep** (*list of floats*) – Power EP parameters

fit_full (*K, epsilon=0.001, power_ep=[1.0, 1.0]*)

The expectation-propagation algorithm. For nomenclature see Rasmussen & Williams 2006.

Parameters

- **epsilon** (*float*) – Convergence criterion, maximum squared difference allowed between mean updates to stop iterations (float)
- **power_ep** (*list of floats*) – Power EP parameters

log_predictive_density (*y_test, mu_star, var_star*)

Calculation of the log predictive density

Parameters

- **y_test** (*(NxI array)*) – test observations ($y_{\{*\}}$)
- **mu_star** (*(NxI array)*) – predictive mean of gaussian $p(f_{\{*\}}|\mu_{\{*\}}, \text{var}_{\{*\}})$

- **var_star** ($(Nx1)$ array) – predictive variance of gaussian $p(f_{\{*\}}|\mu_{\{*\}}, \text{var}_{\{*\}})$

predictive_values (*mu*, *var*, *full_cov*, ***noise_args*)

restart ()

GPy.likelihoods.ep_mixed_noise module

class GPy.likelihoods.ep_mixed_noise.**EP_Mixed_Noise** (*data_list*, *noise_model_list*, *epsilon=0.001*, *power_ep=[1.0, 1.0]*)

Bases: GPy.likelihoods.likelihood.likelihood

fit_DTC (*Kmm*, *Kmn*)

The expectation-propagation algorithm with sparse pseudo-input. For nomenclature see ... 2013.

fit_FITC (*Kmm*, *Kmn*, *Knn_diag*)

The expectation-propagation algorithm with sparse pseudo-input. For nomenclature see Naish-Guzman and Holden, 2008.

fit_full (*K*)

The expectation-propagation algorithm. For nomenclature see Rasmussen & Williams 2006.

predictive_values (*mu*, *var*, *full_cov*, *noise_model*)

Predicts the output given the GP

Parameters

- **mu** – GP’s mean
- **var** – GP’s variance
- **full_cov** (*False|True*) – whether to return the full covariance matrix, or just the diagonal
- **noise_model** (*integer*) – noise model to use

restart ()

GPy.likelihoods.gaussian module

class GPy.likelihoods.gaussian.**Gaussian** (*data*, *variance=1.0*, *normalize=False*)

Bases: GPy.likelihoods.likelihood.likelihood

Likelihood class for doing Expectation propagation

Parameters

- **data** ($(Nx1)$ *numpy.darray*) – observed output
- **variance** – noise parameter
- **normalize** (*False|True*) – whether to normalize the data before computing (predictions will be in original scales)

log_predictive_density (*y_test*, *mu_star*, *var_star*)

Calculation of the log predictive density

Parameters

- **y_test** ($(Nx1)$ array) – test observations ($y_{\{*\}}$)
- **mu_star** ($(Nx1)$ array) – predictive mean of gaussian $p(f_{\{*\}}|\mu_{\{*\}}, \text{var}_{\{*\}})$
- **var_star** ($(Nx1)$ array) – predictive variance of gaussian $p(f_{\{*\}}|\mu_{\{*\}}, \text{var}_{\{*\}})$

predictive_values (*mu, var, full_cov, **likelihood_args*)

Un-normalize the prediction and add the likelihood variance, then return the 5%, 95% interval

set_data (*data*)

GPy.likelihoods.gaussian_mixed_noise module

class GPy.likelihoods.gaussian_mixed_noise.**Gaussian_Mixed_Noise** (*data_list,*
noise_params=None,
normalize=True)

Bases: GPy.likelihoods.likelihood.likelihood

Gaussian Likelihood for multiple outputs

This is a wrapper around likelihood.Gaussian class

Parameters

- **data_list** (*list of numpy arrays (num_data_output_i x 1), one array per output*) – data observations
- **noise_params** (*list of floats, one per output*) – noise parameters of each output
- **normalize** (*False|True*) – whether to normalize the data before computing (predictions will be in original scales)

predictive_values (*mu, var, full_cov, noise_model*)

Predicts the output given the GP

Parameters

- **mu** – GP's mean
- **var** – GP's variance
- **full_cov** (*False|True*) – whether to return the full covariance matrix, or just the diagonal
- **noise_model** (*integer*) – noise model to use

set_data (*data_list*)

GPy.likelihoods.laplace module

class GPy.likelihoods.laplace.**Laplace** (*data, noise_model, extra_data=None*)

Bases: GPy.likelihoods.likelihood.likelihood

Laplace approximation to a posterior

fit_full (*K*)

The laplace approximation algorithm, find K and expand hessian For nomenclature see Rasmussen & Williams 2006 - modified for numerical stability

Parameters **K** (*NxN matrix*) – Prior covariance matrix evaluated at locations X

log_predictive_density (*y_test, mu_star, var_star*)

Calculation of the log predictive density

Parameters

- **y_test** (*(Nx1) array*) – test observations (y_{*})
- **mu_star** (*(Nx1) array*) – predictive mean of gaussian $p(f_{*}|\mu_{*}, \text{var}_{*})$

- **var_star** ((*Nx1*) array) – predictive variance of gaussian $p(f_{\{*\}}|\mu_{\{*\}}, \text{var}_{\{*\}})$

predictive_values (*mu*, *var*, *full_cov*, ***noise_args*)

rasm_mode (*K*, *MAX_ITER=40*)

Rasmussen’s numerically stable mode finding For nomenclature see Rasmussen & Williams 2006 Influenced by GPML (BSD) code, all errors are our own

Parameters

- **K** (*NxD matrix*) – Covariance matrix evaluated at locations *X*
- **MAX_ITER** (*scalar*) – Maximum number of iterations of newton-raphson before forcing finish of optimisation

Returns *f_hat*, mode on which to make laplace approximation

Return type *NxD matrix*

restart ()

Reset likelihood variables to their defaults

GPy.likelihoods.likelihood module

class GPy.likelihoods.likelihood.**likelihood**

Bases: GPy.core.parameterized.Parameterized

The atom for a likelihood class

This object interfaces the GP and the data. The most basic likelihood (Gaussian) inherits directly from this, as does the EP algorithm

Some things must be defined for this to work properly:

- **self.Y** : the effective Gaussian target of the GP
- **self.N**, **self.D** : *Y*.shape
- **self.covariance_matrix** : the effective (noise) covariance of the GP targets
- **self.Z** : a factor which gets added to the likelihood (0 for a Gaussian, *Z_EP* for EP)
- **self.is_heteroscedastic** : enables significant computational savings in GP
- **self.precision** : a scalar or vector representation of the effective target precision
- **self.YYT** : (optional) = $\text{np.dot}(\text{self.Y}, \text{self.Y.T})$ enables computational savings for $D > N$
- **self.V** : $\text{self.precision} * \text{self.Y}$

fit_full (*K*)

No approximations needed by default

log_predictive_density (*y_test*, *mu_star*, *var_star*)

Calculation of the predictive density

Parameters

- **y_test** ((*Nx1*) array) – test observations ($y_{\{*\}}$)
- **mu_star** ((*Nx1*) array) – predictive mean of gaussian $p(f_{\{*\}}|\mu_{\{*\}}, \text{var}_{\{*\}})$
- **var_star** ((*Nx1*) array) – predictive variance of gaussian $p(f_{\{*\}}|\mu_{\{*\}}, \text{var}_{\{*\}})$

predictive_values (*mu*, *var*)

restart ()

No need to restart if not an approximation

GPY.likelihoods.noise_model_constructors module

GPY.likelihoods.noise_model_constructors.**bernoulli** (*gp_link=None*)

Construct a bernoulli likelihood

Parameters *gp_link* – a GPY *gp_link* function

GPY.likelihoods.noise_model_constructors.**exponential** (*gp_link=None*)

Construct a exponential likelihood

Parameters *gp_link* – a GPY *gp_link* function

GPY.likelihoods.noise_model_constructors.**gamma** (*gp_link=None, beta=1.0*)

Construct a Gamma likelihood

Parameters

- **gp_link** – a GPY *gp_link* function
- **beta** – scalar

GPY.likelihoods.noise_model_constructors.**gaussian** (*gp_link=None, variance=2, D=None, N=None*)

Construct a Gaussian likelihood

Parameters

- **gp_link** – a GPY *gp_link* function
- **variance** (*scalar*) – variance

Returns Gaussian noise model:

GPY.likelihoods.noise_model_constructors.**gaussian_ep** (*gp_link=None, variance=1.0*)

Construct a gaussian likelihood

Parameters

- **gp_link** – a GPY *gp_link* function
- **variance** – scalar

GPY.likelihoods.noise_model_constructors.**poisson** (*gp_link=None*)

Construct a Poisson likelihood

Parameters *gp_link* – a GPY *gp_link* function

GPY.likelihoods.noise_model_constructors.**student_t** (*gp_link=None, deg_free=5, sigma2=2*)

Construct a Student t likelihood

Parameters

- **gp_link** – a GPY *gp_link* function
- **deg_free** (*scalar*) – degrees of freedom of student-t
- **sigma2** (*scalar*) – variance

Returns Student-T noise model

Module contents

GPy.mappings package

Submodules

GPy.mappings.kernel module

class GPy.mappings.kernel.**Kernel** (*X*, *output_dim=1*, *kernel=None*)

Bases: GPy.core.mapping.Mapping

Mapping based on a kernel/covariance function.

$$f(\mathbf{x}^*) = \mathbf{A}\mathbf{k}(\mathbf{X}, \mathbf{x}^*) + \mathbf{b}$$

Parameters

- **X** (*ndarray*) – input observations containing **X**
- **output_dim** (*int*) – dimension of output.
- **kernel** (*GPy.kern.kern*) – a GPy kernel, defaults to GPy.kern.rbf

df_dX (*dL_df*, *X*)

df_dtheta (*dL_df*, *X*)

f (*X*)

randomize ()

GPy.mappings.linear module

class GPy.mappings.linear.**Linear** (*input_dim=1*, *output_dim=1*)

Bases: GPy.core.mapping.Mapping

Mapping based on a linear model.

$$f(\mathbf{x}^*) = \mathbf{W}\mathbf{x}^* + \mathbf{b}$$

Parameters

- **X** (*ndarray*) – input observations
- **output_dim** (*int*) – dimension of output.

df_dX (*dL_df*, *X*)

df_dtheta (*dL_df*, *X*)

f (*X*)

randomize ()

GPy.mappings.mlp module

class GPy.mappings.mlp.**MLP** (*input_dim=1*, *output_dim=1*, *hidden_dim=3*)

Bases: GPy.core.mapping.Mapping

Mapping based on a multi-layer perceptron neural network model.

$$f(\mathbf{x}^*) = \mathbf{W}^0 \phi(\mathbf{W}^1 \mathbf{x} + \mathbf{b}^1)^* + \mathbf{b}^0$$

where

$$\phi(\cdot) = \tanh(\cdot)$$

Parameters

- **X** (*ndarray*) – input observations
- **output_dim** (*int*) – dimension of output.
- **hidden_dim** (*int or list of ints.*) – dimension of hidden layer. If it is an int, there is one hidden layer of the given dimension. If it is a list of ints there are as many hidden layers as the length of the list, each with the given number of hidden nodes in it.

df_dX (*dL_df, X*)

df_dtheta (*dL_df, X*)

f (*X*)

randomize ()

Module contents

GPy.models_modules package

Submodules

GPy.models_modules.bayesian_gplvm module

```
class GPy.models_modules.bayesian_gplvm.BayesianGPLVM(likelihood_or_Y, input_dim,
X=None, X_variance=None,
init='PCA', num_inducing=10,
Z=None, kernel=None,
**kwargs)
```

Bases: `GPy.core.sparse_gp.SparseGP`, `GPy.models_modules.gplvm.GPLVM`

Bayesian Gaussian Process Latent Variable Model

Parameters

- **Y** (*np.ndarray| GPy.likelihood instance*) – observed data (*np.ndarray*) or *GPy.likelihood*
- **input_dim** (*int*) – latent dimensionality
- **init** (*'PCA'|'random'*) – initialisation method for the latent space

KL_divergence ()

dKL_dmuS ()

dL_dmuS ()

dmu_dX (*Xnew*)

Calculate the gradient of the prediction at *Xnew* w.r.t *Xnew*.

dmu_dXnew (*Xnew*)

Individual gradient of prediction at *Xnew* w.r.t. each sample in *Xnew*

do_test_latents (*Y*)

Compute the latent representation for a set of new points *Y*

Notes: This will only work with a univariate Gaussian likelihood (for now)

getstate ()

Get the current state of the class, here just all the indices, rest can get recomputed

log_likelihood ()

plot_X_1d (*fignum=None, ax=None, colors=None*)

Plot latent space *X* in 1D:

- if *fig* is given, create *input_dim* subplots in *fig* and plot in these
- if *ax* is given plot *input_dim* 1D latent space plots of *X* into each *axis*
- if neither *fig* nor *ax* is given create a figure with *fignum* and plot in there

colors: colors of different latent space dimensions *input_dim*

plot_latent (*plot_inducing=True, *args, **kwargs*)

plot_steepest_gradient_map (*fignum=None, ax=None, which_indices=None, labels=None, data_labels=None, data_marker='o', data_s=40, resolution=20, aspect='auto', updates=False, **kwargs*)

setstate (*state*)

```
class GPy.models_modules.bayesian_gplvm.BayesianGPLVMWithMissingData (likelihood_or_Y,
                                                                    input_dim,
                                                                    X=None,
                                                                    X_variance=None,
                                                                    init='PCA',
                                                                    num_inducing=10,
                                                                    Z=None,
                                                                    ker-
                                                                    nel=None,
                                                                    **kwargs)
```

Bases: `GPy.core.model.Model`

Bayesian Gaussian Process Latent Variable Model with missing data support. NOTE: Missing data is assumed to be missing at random!

This extension comes with a large memory and computing time deficiency. Use only if fraction of missing data at random is higher than 60%. Otherwise, try filtering data before using this extension.

Y can hold missing data as given by *missing*, standard is *nan*.

If likelihood is given for *Y*, this likelihood will be discarded, but the parameters of the likelihood will be taken. Also every effort of creating the same likelihood will be done.

Parameters

- **likelihood_or_Y** (*ndarray* | *likelihood* instance) – observed data (*np.ndarray*) or *GPy.likelihood*
- **input_dim** (*int*) – latent dimensionality
- **init** (*'PCA'* | *'random'*) – initialisation method for the latent space

getstate ()

log_likelihood ()

setstate (*state*)

`GPy.models_modules.bayesian_gplvm.latent_cost(mu_S, kern, Z, dL_dpsi0, dL_dpsi1, dL_dpsi2)`

objective function for fitting the latent variables (negative log-likelihood: should be minimised!) This is the same as `latent_cost_and_grad` but only for the objective

`GPy.models_modules.bayesian_gplvm.latent_cost_and_grad(mu_S, kern, Z, dL_dpsi0, dL_dpsi1, dL_dpsi2)`

objective function for fitting the latent variables for test points (negative log-likelihood: should be minimised!)

`GPy.models_modules.bayesian_gplvm.latent_grad(mu_S, kern, Z, dL_dpsi0, dL_dpsi1, dL_dpsi2)`

This is the same as `latent_cost_and_grad` but only for the grad

GPy.models_modules.bcgplvm module

class `GPy.models_modules.bcgplvm.BCGPLVM` (*Y, input_dim, init='PCA', X=None, kernel=None, normalize_Y=False, mapping=None*)

Bases: `GPy.models_modules.gplvm.GPLVM`

Back constrained Gaussian Process Latent Variable Model

Parameters

- **Y** (*np.ndarray*) – observed data
- **input_dim** (*int*) – latent dimensionality
- **init** (*'PCA'|'random'*) – initialisation method for the latent space
- **mapping** (*GPy.core.Mapping object*) – mapping for back constraint

GPy.models_modules.fitc_classification module

class `GPy.models_modules.fitc_classification.FITCClassification` (*X, Y=None, likelihood=None, kernel=None, normalize_X=False, normalize_Y=False, Z=None, num_inducing=10*)

Bases: `GPy.core.fitc.FITC`

FITC approximation for classification

This is a thin wrapper around the FITC class, with a set of sensible defaults

Parameters

- **X** – input observations
- **Y** – observed values
- **likelihood** – a GPy likelihood, defaults to Bernoulli with probit link function
- **kernel** – a GPy kernel, defaults to rbf+white
- **normalize_X** (*False|True*) – whether to normalize the input data before computing (predictions will be in original scales)

- **normalize_Y** (*False|True*) – whether to normalize the input data before computing (predictions will be in original scales)

Return type model object

GPpy.models_modules.gp_classification module

```
class GPpy.models_modules.gp_classification.GPClassification(X, Y=None, likelihood=None, kernel=None, normalize_X=False, normalize_Y=False)
```

Bases: `GPpy.core.gp.GP`

Gaussian Process classification

This is a thin wrapper around the models.GP class, with a set of sensible defaults

Parameters

- **X** – input observations
- **Y** – observed values, can be None if likelihood is not None
- **likelihood** – a GPpy likelihood, defaults to Bernoulli with Probit link_function
- **kernel** – a GPpy kernel, defaults to rbf
- **normalize_X** (*False|True*) – whether to normalize the input data before computing (predictions will be in original scales)
- **normalize_Y** (*False|True*) – whether to normalize the input data before computing (predictions will be in original scales)

Note: Multiple independent outputs are allowed using columns of Y

GPpy.models_modules.gp_multioutput_regression module

```
class GPpy.models_modules.gp_multioutput_regression.GPMultioutputRegression(X_list, Y_list, kernel_list=None, noise_variance_list=None, normalize_X=False, normalize_Y=False, rank=1)
```

Bases: `GPpy.core.gp.GP`

Multiple output Gaussian process with Gaussian noise

This is a wrapper around the models.GP class, with a set of sensible defaults

Parameters

- **X_list** (*list of numpy arrays (num_data_output_i x input_dim), one array per output*) – input observations
- **Y_list** (*list of numpy arrays (num_data_output_i x 1), one array per output*) – observed values
- **kernel_list** (*list of GPy kernels*) – GPy kernels, defaults to rbf
- **noise_variance_list** (*list of floats*) – noise parameters per output, defaults to 1.0 for every output
- **normalize_X** (*False|True*) – whether to normalize the input data before computing (predictions will be in original scales)
- **normalize_Y** (*False|True*) – whether to normalize the input data before computing (predictions will be in original scales)
- **rank** (*integer*) – number tuples of the coregionalization parameters ‘coregion_W’ (see coregionalize kernel documentation)

GPy.models_modules.gp_regression module

```
class GPy.models_modules.gp_regression.GPRegression(X, Y, kernel=None, normalize_X=False, normalize_Y=False, likelihood=None)
```

Bases: `GPy.core.gp.GP`

Gaussian Process model for regression

This is a thin wrapper around the models.GP class, with a set of sensible defaults

Parameters

- **X** – input observations
- **Y** – observed values
- **kernel** – a GPy kernel, defaults to rbf
- **normalize_X** (*False|True*) – whether to normalize the input data before computing (predictions will be in original scales)
- **normalize_Y** (*False|True*) – whether to normalize the input data before computing (predictions will be in original scales)

Note: Multiple independent outputs are allowed using columns of Y

getstate ()

setstate (state)

GPy.models_modules.gplvm module

```
class GPy.models_modules.gplvm.GPLVM(Y, input_dim, init='PCA', X=None, kernel=None, normalize_Y=False)
```

Bases: `GPy.core.gp.GP`

Gaussian Process Latent Variable Model

Parameters

- **Y** (*np.ndarray*) – observed data

- **input_dim** (*int*) – latent dimensionality
- **init** (*'pca'* | *'random'*) – initialisation method for the latent space

```

getstate()
jacobian(X)
magnification(X)
plot()
plot_latent(*args, **kwargs)
plot_magnification(*args, **kwargs)
setstate(state)

```

`GPy.models_modules.gplvm.initialise_latent` (*init*, *input_dim*, *Y*)

GPy.models_modules.gradient_checker module

Created on 17 Jul 2013

@author: maxz

```

class GPy.models_modules.gradient_checker.GradientChecker(f, df, x0, names=None,
                                                         *args, **kwargs)

```

Bases: `GPy.core.model.Model`

```

log_likelihood()

```

`GPy.models_modules.gradient_checker.at_least_one_element` (*x*)

`GPy.models_modules.gradient_checker.flatten_if_needed` (*x*)

`GPy.models_modules.gradient_checker.get_shape` (*x*)

GPy.models_modules.mrd module

Created on 10 Apr 2013

@author: Max Zwiessele

```

class GPy.models_modules.mrd.MRD(likelihood_or_Y_list, input_dim, num_inducing=10,
                                   names=None, kernels=None, initx='PCA', initz='permute',
                                   _debug=False, **kw)

```

Bases: `GPy.core.model.Model`

Do MRD on given Datasets in Ylist. All Ys in likelihood_list are in [N x Dn], where Dn can be different per Yn, N must be shared across datasets though.

Parameters

- **likelihood_list** (`[likelihood ndarray]`) – list of observed datasets (`Gaussian` if not supplied directly)
- **names** (`[str]`) – names for different gplvm models
- **input_dim** (*int*) – latent dimensionality
- **initx** (`['concat' | 'single' | 'random']`) – initialisation method for the latent space :
 - ‘concat’ - pca on concatenation of all datasets

- ‘single’ - Concatenation of pca on datasets, respectively
- ‘random’ - Random draw from a normal
- **initz** (*‘permute’|‘random’*) – initialisation method for inducing inputs
- **X** – Initial latent space
- **X_variance** – Initial latent space variance
- **Z** – initial inducing inputs
- **num_inducing** – number of inducing inputs to use
- **kernels** (*[GPy.kern.kern] | GPy.kern.kern | None (default)*) – list of kernels or kernel shared for all BGPLVMS

X

X_variance

Z

auto_scale_factor

set auto_scale_factor for all gplvms :param b: auto_scale_factor :type b:

getstate ()

likelihood_list

log_likelihood ()

plot_X (*fignum=None, ax=None*)

plot_X_1d (**a, **kw*)

plot_latent (*fignum=None, ax=None, *args, **kwargs*)

plot_predict (*fignum=None, ax=None, sharex=False, sharey=False, **kwargs*)

plot_scales (*fignum=None, ax=None, titles=None, sharex=False, sharey=True, *args, **kwargs*)
TODO: Explain other parameters

Parameters titles – titles for axes of datasets

propagate_param (***kwargs*)

randomize (*initx='concat', initz='permute', *args, **kw*)

setstate (*state*)

update_likelihood_approximation ()

GPy.models_modules.sparse_gp_classification module

```

class GPy.models_modules.sparse_gp_classification.SparseGPClassification(X,
                                                                    Y=None,
                                                                    like-
                                                                    li-
                                                                    hood=None,
                                                                    ker-
                                                                    nel=None,
                                                                    nor-
                                                                    mal-
                                                                    ize_X=False,
                                                                    nor-
                                                                    mal-
                                                                    ize_Y=False,
                                                                    Z=None,
                                                                    num_inducing=10)

```

Bases: `GPy.core.sparse_gp.SparseGP`

sparse Gaussian Process model for classification

This is a thin wrapper around the `sparse_GP` class, with a set of sensible defaults

Parameters

- **X** – input observations
- **Y** – observed values
- **likelihood** – a GPy likelihood, defaults to Bernoulli with probit `link_function`
- **kernel** – a GPy kernel, defaults to `rbf+white`
- **normalize_X** (*False|True*) – whether to normalize the input data before computing (predictions will be in original scales)
- **normalize_Y** (*False|True*) – whether to normalize the input data before computing (predictions will be in original scales)

Return type model object

getstate ()

setstate (*state*)

GPy.models_modules.sparse_gp_multioutput_regression module

```
class GPy.models_modules.sparse_gp_multioutput_regression.SparseGPMultioutputRegression(X_list, Y_list, kernel_list, noise_variance_list, normalize_X=False, normalize_Y=False, Z_list=None, num_inducing=10, rank=1)
```

Bases: `GPy.core.sparse_gp.SparseGP`

Sparse multiple output Gaussian process with Gaussian noise

This is a wrapper around the `models.SparseGP` class, with a set of sensible defaults

Parameters

- **X_list** (*list of numpy arrays (num_data_output_i x input_dim), one array per output*) – input observations
- **Y_list** (*list of numpy arrays (num_data_output_i x 1), one array per output*) – observed values
- **kernel_list** (*list of GPy kernels*) – GPy kernels, defaults to rbf
- **noise_variance_list** (*list of floats*) – noise parameters per output, defaults to 1.0 for every output
- **normalize_X** (*False|True*) – whether to normalize the input data before computing (predictions will be in original scales)
- **normalize_Y** (*False|True*) – whether to normalize the input data before computing (predictions will be in original scales)
- **Z_list** (*list of numpy arrays (num_inducing_output_i x input_dim), one array per output | empty list*) – inducing inputs (optional)
- **num_inducing** (*integer*) – number of inducing inputs per output, defaults to 10 (ignored if `Z_list` is not empty)
- **rank** (*integer*) – number tuples of the coregionalization parameters ‘coregion_W’ (see coregionalize kernel documentation)

GPy.models_modules.sparse_gp_regression module

```
class GPy.models_modules.sparse_gp_regression.SparseGPRegression(X, Y, kernel=None, normalize_X=False, normalize_Y=False, Z=None, num_inducing=10, X_variance=None)
```

Bases: `GPy.core.sparse_gp.SparseGP`

Gaussian Process model for regression

This is a thin wrapper around the SparseGP class, with a set of sensible defaults

Parameters

- **X** – input observations
- **Y** – observed values
- **kernel** – a GPy kernel, defaults to rbf+white
- **normalize_X** (*False|True*) – whether to normalize the input data before computing (predictions will be in original scales)
- **normalize_Y** (*False|True*) – whether to normalize the input data before computing (predictions will be in original scales)
- **Z** (*np.ndarray (num_inducing x input_dim) | None*) – inducing inputs (optional, see note)
- **X_variance** (*np.ndarray (num_data x input_dim) | None*) – The uncertainty in the measurements of X (Gaussian variance)

Return type model object

Note: Multiple independent outputs are allowed using columns of Y

getstate ()

setstate (state)

GPy.models_modules.sparse_gplvm module

```
class GPy.models_modules.sparse_gplvm.SparseGPLVM(Y, input_dim, kernel=None, init='PCA', num_inducing=10)
```

Bases: `GPy.models_modules.sparse_gp_regression.SparseGPRegression`,
`GPy.models_modules.gplvm.GPLVM`

Sparse Gaussian Process Latent Variable Model

Parameters

- **Y** (*np.ndarray*) – observed data
- **input_dim** (*int*) – latent dimensionality
- **init** (*'PCA'|'random'*) – initialisation method for the latent space

dL_dX ()

```
getstate()
log_likelihood()
plot()
plot_latent(*args, **kwargs)
setstate(state)
```

GPy.models_modules.svgp_regression module

```
class GPy.models_modules.svgp_regression.SVIGPRegression(X, Y, kernel=None, Z=None,
num_inducing=10, q_u=None, batchsize=10, normalize_Y=False)
```

Bases: `GPy.core.svgp.SVIGP`

Gaussian Process model for regression

This is a thin wrapper around the SVIGP class, with a set of sensible defaults

Parameters

- **X** – input observations
- **Y** – observed values
- **kernel** – a GPy kernel, defaults to rbf+white
- **normalize_X** (*False|True*) – whether to normalize the input data before computing (predictions will be in original scales)
- **normalize_Y** (*False|True*) – whether to normalize the input data before computing (predictions will be in original scales)

Return type model object

Note: Multiple independent outputs are allowed using columns of Y

```
getstate()
setstate(state)
```

GPy.models_modules.warped_gp module

```
class GPy.models_modules.warped_gp.WarpedGP(X, Y, kernel=None, warping_function=None,
warping_terms=3, normalize_X=False, normalize_Y=False)
```

Bases: `GPy.core.gp.GP`

```
getstate()
log_likelihood()
plot_warping()
predict(Xnew, which_parts='all', full_cov=False, pred_init=None)
setstate(state)
```

```
transform_data()
warping_function_gradients(Kiy)
```

Module contents

GPy.testing package

Submodules

GPy.testing.bcgplvm_tests module

```
class GPy.testing.bcgplvm_tests.BCGPLVMTests (methodName='runTest')
    Bases: unittest.case.TestCase
    test_kernel_backconstraint()
    test_linear_backconstraint()
    test_mlp_backconstraint()
```

GPy.testing.bgplvm_tests module

```
class GPy.testing.bgplvm_tests.BGPLVMTests (methodName='runTest')
    Bases: unittest.case.TestCase
    test_bias_kern()
    test_linear_bias_kern()
    test_linear_kern()
    test_rbf_bias_kern()
    test_rbf_kern()
    test_rbf_line_kern()
```

GPy.testing.cgd_tests module

Created on 26 Apr 2013

@author: maxz

```
class GPy.testing.cgd_tests.Test (methodName='runTest')
    Bases: unittest.case.TestCase
    testMinimizeSquare()
    testRosen()
```

GPy.testing.examples_tests module

```
class GPy.testing.examples_tests.ExamplesTests (methodName='runTest')
    Bases: unittest.case.TestCase
GPy.testing.examples_tests.flatten_nested(lst)
```

```
GPpy.testing.examples_tests.model_checkgrads(model)
GPpy.testing.examples_tests.model_instance(model)
GPpy.testing.examples_tests.test_models()
```

GPpy.testing.gp_transformation_tests module

```
class GPpy.testing.gp_transformation_tests.TestTransformations
    Bases: object
    Generic transformations checker
    setUp()
    t_d2transf_df2(transformation, f)
    t_d3transf_df3(transformation, f)
    t_dtransf_df(transformation, f)
    tearDown()
    test_transformations()
```

GPpy.testing.gplvm_tests module

```
class GPpy.testing.gplvm_tests.GPLVMTests(methodName='runTest')
    Bases: unittest.case.TestCase
    test_bias_kern()
    test_linear_kern()
    test_rbf_kern()
```

GPpy.testing.kernel_tests module

```
class GPpy.testing.kernel_tests.KernelTests(methodName='runTest')
    Bases: unittest.case.TestCase
    test_Matern32kernel()
    test_Matern52kernel()
    test_eq_sympykernel()
    test_fixedkernel()
        Fixed effect kernel test
    test_gibbskernel()
    test_heterokernel()
    test_kernel_tie()
    test_linearkernel()
    test_mlpkernel()
    test_odel_eqkernel()
    test_periodic_Matern32kernel()
```

```

test_periodic_Matern52kernel ()
test_periodic_exponentialkernel ()
test_polykernel ()
test_rational_quadratickernel ()
test_rbf_invkernel ()
test_rbf_sympykernel ()
test_rbfkernel ()

```

GPy.testing.likelihood_tests module

```

class GPy.testing.likelihood_tests.LaplaceTests (methodName='runTest')
    Bases: unittest.case.TestCase

```

Specific likelihood tests, not general enough for the above tests

```

setUp ()
tearDown ()
test_gaussian_d2logpdf_df2_2 ()
test_laplace_log_likelihood ()

```

```

class GPy.testing.likelihood_tests.TestNoiseModels
    Bases: object

```

Generic model checker

```

setUp ()
t_d2logpdf2_df2_dparams (model, Y, f, params, param_constraints)
t_d2logpdf2_dlink2_dparams (model, Y, f, params, param_constraints)
t_d2logpdf_df2 (model, Y, f)
t_d2logpdf_dlink2 (model, Y, f, link_f_constraints)
t_d3logpdf_df3 (model, Y, f)
t_d3logpdf_dlink3 (model, Y, f, link_f_constraints)
t_dlogpdf_df (model, Y, f)
t_dlogpdf_df_dparams (model, Y, f, params, param_constraints)
t_dlogpdf_dlink (model, Y, f, link_f_constraints)
t_dlogpdf_dlink_dparams (model, Y, f, params, param_constraints)
t_dlogpdf_dparams (model, Y, f, params, param_constraints)
t_dlogpdf_link_dparams (model, Y, f, params, param_constraints)
t_ep_fit_rbf_white (model, X, Y, f, step, param_vals, param_names, constraints)
t_laplace_fit_rbf_white (model, X, Y, f, step, param_vals, param_names, constraints)
t_logpdf (model, Y, f)
tearDown ()

```

```
test_noise_models()
```

```
GPy.testing.likelihood_tests.dparam_checkgrad(func, dfunc, params, args, constraints=None, randomize=False, verbose=False)
```

checkgrad expects a $f: \mathbb{R}^N \rightarrow \mathbb{R}^1$ and $df: \mathbb{R}^N \rightarrow \mathbb{R}^N$ However if we are holding other parameters fixed and moving something else We need to check the gradient of each of the fixed parameters (f and y for example) separately, whilst moving another parameter. Otherwise f : gives back \mathbb{R}^N and

df : gives back $\mathbb{R}^N \times M$ where M is

The number of parameters and N is the number of data Need to take a slice out from f and a slice out of df

```
GPy.testing.likelihood_tests.dparam_partial(inst_func, *args)
```

If we have a instance method that needs to be called but that doesn't take the parameter we wish to change to checkgrad, then this function will change the variable using set params.

inst_func: should be a instance function of an object that we would like to change

param: the param that will be given to set_params args: anything else that needs to be given to the function (for example

the f or Y that are being used in the function whilst we tweak the param

GPy.testing.mapping_tests module

```
class GPy.testing.mapping_tests.MappingTests(methodName='runTest')
```

```
Bases: unittest.case.TestCase
```

```
test_kernelmapping()
```

```
test_linearmapping()
```

```
test_mlpmapping()
```

GPy.testing.mrd_tests module

Created on 10 Apr 2013

@author: maxz

```
class GPy.testing.mrd_tests.MRDTests(methodName='runTest')
```

```
Bases: unittest.case.TestCase
```

```
test_gradients()
```

GPy.testing.prior_tests module

```
class GPy.testing.prior_tests.PriorTests(methodName='runTest')
```

```
Bases: unittest.case.TestCase
```

```
test_Gamma()
```

```
test_incompatibility()
```

```
test_lognormal()
```


GPy.testing.psi_stat_expectation_tests module

Created on 26 Apr 2013

@author: maxz

```
class GPy.testing.psi_stat_expectation_tests.Test (methodName='runTest')
    Bases: unittest.case.TestCase

    N = 300

    Nsamples = 1000000.0

    input_dim = 9

    num_inducing = 13

    setUp()

    test_psi0()

    test_psi1()

    test_psi2()
```

```
GPy.testing.psi_stat_expectation_tests.ard(p)
```

GPy.testing.psi_stat_gradient_tests module

Created on 22 Apr 2013

@author: maxz

```
class GPy.testing.psi_stat_gradient_tests.DPsiStatTest (methodName='runTest')
    Bases: unittest.case.TestCase

    N = 50

    X = array([[ 1.69052570e+00, -4.65937371e-01, 3.28201637e-02, 4.07516283e-01, -7.88923029e-01, 2.06557291e-03, -8.90385
    X_var = array([[ 0.85395518, 0.5 , 0.9 , 0.53474051, 0.80066698, 0.9 , 0.5 , 0.5 , 0.67069375, 0.5 , 0.5 , 0.5 , 0.5627431 , 0.5 ,
    Y = array([[ -2.20747097e+00, -5.73524833e-01, 7.73270442e-01, -3.29475962e+00, 3.30574015e+00, -6.65849736e-01, 2.664
    Z = array([[ 0.05342464, -0.85023395, 1.16623315, -0.43332385, -0.72547002, -1.56199585, 0.007915 , -1.04764369, -0.51913
    input_dim = 20

    kernels = [<GPy.kern.kern.kern object at 0x7f37ab632a50>, <GPy.kern.kern.kern object at 0x7f37ab632bd0>, <GPy.ke
    num_inducing = 10

    testPsi0()

    testPsi1()

    testPsi2_bia()

    testPsi2_lin()

    testPsi2_lin_bia()

    testPsi2_rbf()

    testPsi2_rbf_bia()
```

```
class GPy.testing.psi_stat_gradient_tests.PsiStatModel (which, X, X_variance, Z,
                                                    num_inducing, kernel)
    Bases: GPy.core.model.Model
    log_likelihood()
```

GPy.testing.sparse_gplvm_tests module

```
class GPy.testing.sparse_gplvm_tests.sparse_GPLVMTests (methodName='runTest')
    Bases: unittest.case.TestCase
    test_bias_kern()
    test_linear_kern()
    test_rbf_kern()
```

GPy.testing.unit_tests module

```
class GPy.testing.unit_tests.GradientTests (methodName='runTest')
    Bases: unittest.case.TestCase
    check_model (kern, model_type='GPRegression', dimension=1, uncertain_inputs=False)
    multioutput_regression_1D()
    multioutput_sparse_regression_1D()
    setUp()
    test_GPLVM_rbf_bias_white_kern_2D()
        Testing GPLVM with rbf + bias kernel
    test_GPLVM_rbf_linear_white_kern_2D()
        Testing GPLVM with rbf + bias kernel
    test_GPRegression_bias_kern_1D()
        Testing the GP regression with bias kernel on 1d data
    test_GPRegression_bias_kern_2D()
        Testing the GP regression with bias kernel on 2d data
    test_GPRegression_exponential_1D()
        Testing the GP regression with exponential kernel on 1d data
    test_GPRegression_exponential_2D()
        Testing the GP regression with exponential kernel on 2d data
    test_GPRegression_exponential_ARD_2D()
        Testing the GP regression with exponential kernel on 2d data
    test_GPRegression_linear_kern_1D()
        Testing the GP regression with linear kernel on 1d data
    test_GPRegression_linear_kern_1D_ARD()
        Testing the GP regression with linear kernel on 1d data
    test_GPRegression_linear_kern_2D()
        Testing the GP regression with linear kernel on 2d data
    test_GPRegression_linear_kern_2D_ARD()
        Testing the GP regression with linear kernel on 2d data
```

```

test_GPRegression_matern32_1D()
    Testing the GP regression with matern32 kernel on 1d data

test_GPRegression_matern32_2D()
    Testing the GP regression with matern32 kernel on 2d data

test_GPRegression_matern32_ARD_2D()
    Testing the GP regression with matern32 kernel on 2d data

test_GPRegression_matern52_1D()
    Testing the GP regression with matern52 kernel on 1d data

test_GPRegression_matern52_2D()
    Testing the GP regression with matern52 kernel on 2d data

test_GPRegression_matern52_ARD_2D()
    Testing the GP regression with matern52 kernel on 2d data

test_GPRegression_mlp_1d()
    Testing the GP regression with mlp kernel with white kernel on 1d data

test_GPRegression_poly_1d()
    Testing the GP regression with polynomial kernel with white kernel on 1d data

test_GPRegression_rbf_1d()
    Testing the GP regression with rbf kernel with white kernel on 1d data

test_GPRegression_rbf_2D()
    Testing the GP regression with rbf kernel on 2d data

test_GPRegression_rbf_ARD_2D()
    Testing the GP regression with rbf kernel on 2d data

test_GP_EP_probit()

test_SparseGPRegression_rbf_linear_white_kern_1D()
    Testing the sparse GP regression with rbf kernel on 2d data

test_SparseGPRegression_rbf_linear_white_kern_1D_uncertain_inputs()
    Testing the sparse GP regression with rbf, linear kernel on 1d data with uncertain inputs

test_SparseGPRegression_rbf_linear_white_kern_2D()
    Testing the sparse GP regression with rbf kernel on 2d data

test_SparseGPRegression_rbf_linear_white_kern_2D_uncertain_inputs()
    Testing the sparse GP regression with rbf, linear kernel on 2d data with uncertain inputs

test_SparseGPRegression_rbf_white_kern_1d()
    Testing the sparse GP regression with rbf kernel with white kernel on 1d data

test_SparseGPRegression_rbf_white_kern_2D()
    Testing the sparse GP regression with rbf kernel on 2d data

test_generalized_FITC()

test_sparse_EP_DTC_probit()

```

Module contents

MaxZ

GPy.testing.deepTest(*reason*)

GPy.util package

Subpackages

GPy.util.latent_space_visualizations package

Subpackages

GPy.util.latent_space_visualizations.controllers package

Submodules

GPy.util.latent_space_visualizations.controllers.axis_event_controller module Created on 24 Jul 2013

@author: maxz

class GPy.util.latent_space_visualizations.controllers.axis_event_controller.**AxisChangedControl**

Bases: GPy.util.latent_space_visualizations.controllers.axis_event_controller.AxisEventControl

Buffered control of axis limit changes

extent (*lim*)

lim_changed (*axlim*, *savedlim*)

update (*ax*)

xlim_changed (*ax*)

ylim_changed (*ax*)

class GPy.util.latent_space_visualizations.controllers.axis_event_controller.**AxisEventControl**

Bases: object

activate ()

deactivate ()

xlim_changed (*ax*)

ylim_changed (*ax*)

class GPy.util.latent_space_visualizations.controllers.axis_event_controller.**BufferedAxisChar**

Bases: GPy.util.latent_space_visualizations.controllers.axis_event_controller.AxisChangedControl

get_grid ()

```

recompute_X()
update(ax)
update_view(view, X, xmin, xmax, ymin, ymax)

```

GPy.util.latent_space_visualizations.controllers.imshow_controller module Created on 24 Jul 2013

@author: maxz

```

class GPy.util.latent_space_visualizations.controllers.imshow_controller.ImAnnotateController

```

Bases: GPy.util.latent_space_visualizations.controllers.imshow_controller.ImshowController

```

update_view(view, X, xmin, xmax, ymin, ymax)

```

```

class GPy.util.latent_space_visualizations.controllers.imshow_controller.ImshowController(ax,
                                             plot,
                                             plot,
                                             res-
                                             o-
                                             lu-
                                             tion
                                             up-
                                             date
                                             **k

```

Bases: GPy.util.latent_space_visualizations.controllers.axis_event_controller.BufferedAxis

```

update_view(view, X, xmin, xmax, ymin, ymax)

```

Module contents

Module contents

Submodules

GPy.util.Tango module

```

GPy.util.Tango.currentDark()
GPy.util.Tango.currentLight()
GPy.util.Tango.currentMedium()
GPy.util.Tango.fewerXticks(ax=None, divideby=2)
GPy.util.Tango.hex2rgb(hexcolor)

```

```
GPy.util.Tango.nextDark()
GPy.util.Tango.nextLight()
GPy.util.Tango.nextMedium()
GPy.util.Tango.removeRightTicks(ax=None)
GPy.util.Tango.removeUpperTicks(ax=None)
GPy.util.Tango.reset()
GPy.util.Tango.setDarkFigures()
GPy.util.Tango.setLightFigures()
```

GPy.util.block_matrices module

```
GPy.util.block_matrices.get_blocks(A, blocksizes)
```

GPy.util.classification module

```
GPy.util.classification.conf_matrix(p, labels, names=['1', '0'], threshold=0.5, show=True)
```

Returns error rate and true/false positives in a binary classification problem - Actual classes are displayed by column. - Predicted classes are displayed by row.

Parameters

- **p** – array of class ‘1’ probabilities.
- **labels** – array of actual classes.
- **names** – list of class names, defaults to ['1', '0'].
- **threshold** – probability value used to decide the class.
- **show** (*False|True*) – whether the matrix should be shown or not

GPy.util.config module

GPy.util.datasets module

```
GPy.util.datasets.authorize_download(dataset_name=None)
    Check with the user that they are happy with terms and conditions for the data set.
GPy.util.datasets.boston_housing(data_set='boston_housing')
GPy.util.datasets.brendan_faces(data_set='brendan_faces')
GPy.util.datasets.cmu_mocap(subject, train_motions, test_motions=[], sample_every=4,
                           data_set='cmu_mocap')
    Load a given subject's training and test motions from the CMU motion capture data.
GPy.util.datasets.cmu_mocap_35_walk_jog(data_set='cmu_mocap')
    Load CMU subject 35's walking and jogging motions, the same data that was used by Taylor, Roweis and Hinton at NIPS 2007. but without their preprocessing. Also used by Lawrence at AISTATS 2007.
GPy.util.datasets.cmu_mocap_49_balance(data_set='cmu_mocap')
    Load CMU subject 49's one legged balancing motion that was used by Alvarez, Luengo and Lawrence at AISTATS 2009.
```

`GPy.util.datasets.cmu_urls_files` (*subj_motions*, *messages=True*)
Find which resources are missing on the local disk for the requested CMU motion capture motions.

`GPy.util.datasets.creep_data` (*data_set='creep_rupture'*)
Brun and Yoshida's metal creep rupture data.

`GPy.util.datasets.crescent_data` (*num_data=200*, *seed=10000*)
Data set formed from a mixture of four Gaussians. In each class two of the Gaussians are elongated at right angles to each other and offset to form an approximation to the crescent data that is popular in semi-supervised learning as a toy problem.

param num_data_part number of data to be sampled (default is 200).

type num_data int

param seed random seed to be used for data generation.

type seed int

`GPy.util.datasets.data_available` (*dataset_name=None*)
Check if the data set is available on the local machine already.

`GPy.util.datasets.data_details_return` (*data*, *data_set*)
Update the data component of the data dictionary with details drawn from the data_resources.

`GPy.util.datasets.della_gatta_TRP63_gene_expression` (*data_set='della_gatta'*,
gene_number=None)

`GPy.util.datasets.download_data` (*dataset_name=None*)
Check with the user that they are happy with terms and conditions for the data set, then download it.

`GPy.util.datasets.download_rogers_girolami_data` (*data_set='rogers_girolami_data'*)

`GPy.util.datasets.download_url` (*url*, *store_directory*, *save_name=None*, *messages=True*, *suffix=''*)
Download a file from a url and save it to disk.

`GPy.util.datasets.hapmap3` (*data_set='hapmap3'*)

`GPy.util.datasets.isomap_faces` (*num_samples=698*, *data_set='isomap_face_data'*)

`GPy.util.datasets.oil` (*data_set='three_phase_oil_flow'*)
The three phase oil data from Bishop and James (1993).

`GPy.util.datasets.oil_100` (*seed=10000*, *data_set='three_phase_oil_flow'*)

`GPy.util.datasets.olivetti_faces` (*data_set='olivetti_faces'*)

`GPy.util.datasets.olympic_100m_men` (*data_set='rogers_girolami_data'*)

`GPy.util.datasets.olympic_100m_women` (*data_set='rogers_girolami_data'*)

`GPy.util.datasets.olympic_200m_men` (*data_set='rogers_girolami_data'*)

`GPy.util.datasets.olympic_200m_women` (*data_set='rogers_girolami_data'*)

`GPy.util.datasets.olympic_400m_men` (*data_set='rogers_girolami_data'*)

`GPy.util.datasets.olympic_400m_women` (*data_set='rogers_girolami_data'*)

`GPy.util.datasets.olympic_marathon_men` (*data_set='olympic_marathon_men'*)

`GPy.util.datasets.olympic_sprints` (*data_set='rogers_girolami_data'*)
All olympics sprint winning times for multiple output prediction.

`GPy.util.datasets.osu_run1` (*data_set='osu_run1'*, *sample_every=4*)

`GPy.util.datasets.prompt_user(prompt)`

Ask user for agreeing to data set licenses.

`GPy.util.datasets.pumadyn(seed=10000, data_set='pumadyn-32nm')`

`GPy.util.datasets.reporthook(a, b, c)`

`GPy.util.datasets.ripley_synth(data_set='ripley_prnn_data')`

`GPy.util.datasets.robot_wireless(data_set='robot_wireless')`

`GPy.util.datasets.sample_class(f)`

`GPy.util.datasets.silhouette(data_set='ankur_pose_data')`

`GPy.util.datasets.simulation_BGPLVM()`

`GPy.util.datasets.swiss_roll(num_samples=3000, data_set='swiss_roll')`

`GPy.util.datasets.swiss_roll_1000()`

`GPy.util.datasets.swiss_roll_generated(num_samples=1000, sigma=0.0)`

`GPy.util.datasets.toy_linear_1d_classification(seed=10000)`

`GPy.util.datasets.toy_rbf_1d(seed=10000, num_samples=500)`

Samples values of a function from an RBF covariance with very small noise for inputs uniformly distributed between -1 and 1.

Parameters

- **seed** (*int*) – seed to use for random sampling.
- **num_samples** (*int*) – number of samples to sample in the function (default 500).

`GPy.util.datasets.toy_rbf_1d_50(seed=10000)`

`GPy.util.datasets.xw_pen(data_set='xw_pen')`

GPy.util.decorators module

`GPy.util.decorators.silence_errors(f)`

This wraps a function and it silences numpy errors that happen during the execution. After the function has exited, it restores the previous state of the warnings.

GPy.util.diag module

`GPy.util.diag.add(A, b, offset=0)`

Add b to the view of A in place (!). Returns modified A. Broadcasting is allowed, thus b can be scalar.

if offset is not zero, make sure b is of right shape!

Parameters

- **A** (*ndarray*) – 2 dimensional array
- **b** (*ndarray-like*) – either one dimensional or scalar
- **offset** (*int*) – same as in view.

Return type view of A, which is adjusted inplace

`GPy.util.diag.divide(A, b, offset=0)`

Divide the view of A by b in place (!). Returns modified A Broadcasting is allowed, thus b can be scalar.

if offset is not zero, make sure b is of right shape!

Parameters

- **A** (*ndarray*) – 2 dimensional array
- **b** (*ndarray-like*) – either one dimensional or scalar
- **offset** (*int*) – same as in view.

Return type view of A, which is adjusted inplace

`GPy.util.diag.multiply(A, b, offset=0)`

Times the view of A with b in place (!). Returns modified A Broadcasting is allowed, thus b can be scalar.

if offset is not zero, make sure b is of right shape!

Parameters

- **A** (*ndarray*) – 2 dimensional array
- **b** (*ndarray-like*) – either one dimensional or scalar
- **offset** (*int*) – same as in view.

Return type view of A, which is adjusted inplace

`GPy.util.diag.subtract(A, b, offset=0)`

Subtract b from the view of A in place (!). Returns modified A. Broadcasting is allowed, thus b can be scalar.

if offset is not zero, make sure b is of right shape!

Parameters

- **A** (*ndarray*) – 2 dimensional array
- **b** (*ndarray-like*) – either one dimensional or scalar
- **offset** (*int*) – same as in view.

Return type view of A, which is adjusted inplace

`GPy.util.diag.times(A, b, offset=0)`

Times the view of A with b in place (!). Returns modified A Broadcasting is allowed, thus b can be scalar.

if offset is not zero, make sure b is of right shape!

Parameters

- **A** (*ndarray*) – 2 dimensional array
- **b** (*ndarray-like*) – either one dimensional or scalar
- **offset** (*int*) – same as in view.

Return type view of A, which is adjusted inplace

`GPy.util.diag.view(A, offset=0)`

Get a view on the diagonal elements of a 2D array.

This is actually a view (!) on the diagonal of the array, so you can in-place adjust the view.

:param *ndarray* A: 2 dimensional numpy array :param *int* offset: view offset to give back (negative entries allowed) :rtype: *ndarray* view of diag(A)

```
>>> import numpy as np
>>> X = np.arange(9).reshape(3,3)
>>> view(X)
array([0, 4, 8])
>>> d = view(X)
>>> d += 2
>>> view(X)
array([ 2,  6, 10])
>>> view(X, offset=-1)
array([3, 7])
>>> subtract(X, 3, offset=-1)
array([[ 2,  1,  2],
       [ 0,  6,  5],
       [ 6,  4, 10]])
```

GPy.util.erfcx module

GPy.util.erfcx.**erfcx**(arg)

GPy.util.linalg module

GPy.util.linalg.**DSYR**(*args, **kwargs)

GPy.util.linalg.**DSYR_blas**(A, x, alpha=1.0)

Performs a symmetric rank-1 update operation: $A \leftarrow A + \alpha * \text{np.dot}(x, x.T)$

Parameters

- **A** – Symmetric NxN np.array
- **x** – Nx1 np.array
- **alpha** – scalar

GPy.util.linalg.**DSYR_numpy**(A, x, alpha=1.0)

Performs a symmetric rank-1 update operation: $A \leftarrow A + \alpha * \text{np.dot}(x, x.T)$

Parameters

- **A** – Symmetric NxN np.array
- **x** – Nx1 np.array
- **alpha** – scalar

GPy.util.linalg.**backsub_both_sides**(L, X, transpose='left')

Return $L^{-T} * X * L^{-1}$, assuming X is symmetrical and L is lower cholesky

GPy.util.linalg.**chol_inv**(L)

Inverts a Cholesky lower triangular matrix

Parameters **L** – lower triangular matrix

Return type inverse of L

GPy.util.linalg.**cholupdate**(L, x)

update the LOWER cholesky factor of a pd matrix IN PLACE

if L is the lower chol. of K, then this function computes L_* where L_* is the lower chol of $K + x * x^T$

`GPy.util.linalg.dpotri` (*A*, *lower=0*)

Wrapper for lapack dpotri function

Parameters

- **A** – Matrix A
- **lower** – is matrix lower (true) or upper (false)

Returns A inverse

`GPy.util.linalg.dpotrs` (*A*, *B*, *lower=0*)

Wrapper for lapack dpotrs function

Parameters

- **A** – Matrix A
- **B** – Matrix B
- **lower** – is matrix lower (true) or upper (false)

Returns

`GPy.util.linalg.dtrtrs` (*A*, *B*, *lower=0*, *trans=0*, *unitdiag=0*)

Wrapper for lapack dtrtrs function

Parameters

- **A** – Matrix A
- **B** – Matrix B
- **lower** – is matrix lower (true) or upper (false)

Returns

`GPy.util.linalg.jitchol` (*A*, *maxtries=5*)

`GPy.util.linalg.jitchol_old` (*A*, *maxtries=5*)

Parameters **A** – An almost pd square matrix

Rval **L** the Cholesky decomposition of A

`GPy.util.linalg.mdot` (*args)

Multiply all the arguments using matrix product rules. The output is equivalent to multiplying the arguments one by one from left to right using dot(). Precedence can be controlled by creating tuples of arguments, for instance `mdot(a,((b,c),d))` multiplies a $(a*((b*c)*d))$. Note that this means the output of `dot(a,b)` and `mdot(a,b)` will differ if a or b is a pure tuple of numbers.

`GPy.util.linalg.multiple_pdiv` (*A*)

Parameters **A** – A DxDxN numpy array (each $A[:, :, i]$ is pd)

Rval **invs** the inverses of A

Rtype **invs** np.ndarray

Rval **hld** 0.5* the log of the determinants of A

Rtype **hld** np.array

`GPy.util.linalg.pca` (*Y*, *input_dim*)

Principal component analysis: maximum likelihood solution by SVD

Parameters

- **Y** – NxN np.array of data

- **input_dim** – int, dimension of projection

Rval X

- Nxinput_dim np.array of dimensionality reduced data

Rval W

- input_dimxD mapping from X to Y

GPy.util.linalg.**pddet**(A)

Determinant of a positive definite matrix, only symmetric matrices though

GPy.util.linalg.**pdinv**(A, *args)

Parameters A – A DxD pd numpy array

Rval Ai the inverse of A

Rtype Ai np.ndarray

Rval L the Cholesky decomposition of A

Rtype L np.ndarray

Rval Li the Cholesky decomposition of Ai

Rtype Li np.ndarray

Rval logdet the log of the determinant of A

Rtype logdet float64

GPy.util.linalg.**ppca**(Y, Q, iterations=100)

EM implementation for probabilistic pca.

Parameters

- **Y** (*array-like*) – Observed Data
- **Q** (*int*) – Dimensionality for reduced array
- **iterations** (*int*) – number of iterations for EM

GPy.util.linalg.**ppca_missing_data_at_random**(Y, Q, iters=100)

EM implementation of Probabilistic pca for when there is missing data.

Taken from <SheffieldML, <https://github.com/SheffieldML>>

Returns X, W, sigma^2

GPy.util.linalg.**symmetrify**(A, upper=False)

Take the square matrix A and make it symmetrical by copying elements from the lower half to the upper works IN PLACE.

GPy.util.linalg.**symmetrify_murray**(A)

GPy.util.linalg.**tdot**(*args, **kwargs)

GPy.util.linalg.**tdot_blas**(mat, out=None)

returns np.dot(mat, mat.T), but faster for large 2D arrays of doubles.

GPy.util.linalg.**tdot_numpy**(mat, out=None)

GPy.util.linalg.**trace_dot**(a, b)

Efficiently compute the trace of the matrix product of a and b

GPy.util.ln_diff_erfs module

`GPy.util.ln_diff_erfs.ln_diff_erfs(x1, x2, return_sign=False)`

Function for stably computing the log of difference of two erfs in a numerically stable manner. :param x1 : argument of the positive erf :type x1: ndarray :param x2 : argument of the negative erf :type x2: ndarray :return: tuple containing (log(abs(erf(x1) - erf(x2))), sign(erf(x1) - erf(x2)))

Based on MATLAB code that was written by Antti Honkela and modified by David Luengo and originally derived from code by Neil Lawrence.

GPy.util.misc module

`GPy.util.misc.chain_1(df_dg, dg_dx)`

Generic chaining function for first derivative

$$\frac{d(f.g)}{dx} = \frac{df}{dg} \frac{dg}{dx}$$

`GPy.util.misc.chain_2(d2f_dg2, dg_dx, df_dg, d2g_dx2)`

Generic chaining function for second derivative

$$\frac{d^2(f.g)}{dx^2} = \frac{d^2f}{dg^2} \left(\frac{dg}{dx}\right)^2 + \frac{df}{dg} \frac{d^2g}{dx^2}$$

`GPy.util.misc.chain_3(d3f_dg3, dg_dx, d2f_dg2, d2g_dx2, df_dg, d3g_dx3)`

Generic chaining function for third derivative

$$\frac{d^3(f.g)}{dx^3} = \frac{d^3f}{dg^3} \left(\frac{dg}{dx}\right)^3 + 3 \frac{d^2f}{dg^2} \frac{dg}{dx} \frac{d^2g}{dx^2} + \frac{df}{dg} \frac{d^3g}{dx^3}$$

`GPy.util.misc.fast_array_equal(A, B)`

`GPy.util.misc.fast_array_equal2(A, B)`

`GPy.util.misc.kmm_init(X, m=10)`

This is the same initialization algorithm that is used in Kmeans++. It's quite simple and very useful to initialize the locations of the inducing points in sparse GPs.

Parameters

- **X** – data
- **m** – number of inducing points

`GPy.util.misc.linear_grid(D, n=100, min_max=(-100, 100))`

Creates a D-dimensional grid of n linearly spaced points

Parameters

- **D** – dimension of the grid
- **n** – number of points
- **min_max** – (min, max) list

`GPy.util.misc.opt_wrapper(m, **kwargs)`

This function just wraps the optimization procedure of a GPy object so that optimize() pickleable (necessary for multiprocessing).

GPY.util.mocap module

class GPY.util.mocap.**acclaim_skeleton** (*file_name=None*)

Bases: GPY.util.mocap.skeleton

get_child_xyz (*ind, channels*)

load_channels (*file_name*)

load_skel (*file_name*)

Loads an ASF file into a skeleton structure.

Parameters **file_name** – The file name to load in.

read_bonedata (*fid*)

Read bone data from an acclaim skeleton file stream.

read_channels (*fid*)

Read channels from an acclaim file.

read_documentation (*fid*)

Read documentation from an acclaim skeleton file stream.

read_hierarchy (*fid*)

Read hierarchy information from acclaim skeleton file stream.

read_line (*fid*)

Read a line from a file string and check it isn't either empty or commented before returning.

read_root (*fid*)

Read the root node from an acclaim skeleton file stream.

read_skel (*fid*)

Loads an acclaim skeleton format from a file stream.

read_units (*fid*)

Read units from an acclaim skeleton file stream.

resolve_indices (*index, start_val*)

Get indices for the skeleton from the channels when loading in channel data.

set_rotation_matrices ()

Set the meta information at each vertex to contain the correct matrices C and Cinv as prescribed by the rotations and rotation orders.

to_xyz (*channels*)

GPY.util.mocap.**load_text_data** (*dataset, directory, centre=True*)

Load in a data set of marker points from the Ohio State University C3D motion capture files (http://accad.osu.edu/research/mocap/mocap_data.htm).

GPY.util.mocap.**parse_text** (*file_name*)

Parse data from Ohio State University text mocap files (http://accad.osu.edu/research/mocap/mocap_data.htm).

GPY.util.mocap.**read_connections** (*file_name, point_names*)

Read a file detailing which markers should be connected to which for motion capture data.

GPY.util.mocap.**rotation_matrix** (*xangle, yangle, zangle, order='zy', degrees=False*)

Compute the rotation matrix for an angle in each direction. This is a helper function for computing the rotation matrix for a given set of angles in a given order.

Parameters

- **xangle** – rotation for x-axis.

- **yangle** – rotation for y-axis.
- **zangle** – rotation for z-axis.
- **order** – the order for the rotations.

```
class GPy.util.mocap.skeleton
    Bases: GPy.util.mocap.tree
```

```
    connection_matrix()
```

```
    finalize()
```

After loading in a skeleton ensure parents are correct, vertex orders are correct and rotation matrices are correct.

```
    smooth_angle_channels(channels)
```

Remove discontinuities in angle channels so that they don't cause artifacts in algorithms that rely on the smoothness of the functions.

```
    to_xyz(channels)
```

```
class GPy.util.mocap.tree
```

```
    branch_str(index, indent='')
```

```
    find_children()
```

Take a tree and set the children according to the parents.

Takes a tree structure which lists the parents of each vertex and computes the children for each vertex and places them in.

```
    find_parents()
```

Take a tree and set the parents according to the children

Takes a tree structure which lists the children of each vertex and computes the parents for each vertex and places them in.

```
    find_root()
```

Finds the index of the root node of the tree.

```
    get_index_by_id(id)
```

Give the index associated with a given vertex id.

```
    get_index_by_name(name)
```

Give the index associated with a given vertex name.

```
    order_vertices()
```

Order vertices in the graph such that parents always have a lower index than children.

```
    swap_vertices(i, j)
```

Swap two vertices in the tree structure array. swap_vertex swaps the location of two vertices in a tree structure array.

Parameters

- **tree** – the tree for which two vertices are to be swapped.
- **i** – the index of the first vertex to be swapped.
- **j** – the index of the second vertex to be swapped.

Rval tree the tree structure with the two vertex locations swapped.

```
class GPy.util.mocap.vertex(name, id, parents=[], children=[], meta={})
```

GPY.util.multioutput module

`GPY.util.multioutput.build_lcm(input_dim, num_outputs, CK=[], NC=[], W_columns=1, W=None, kappa=None)`

Builds a kernel for a linear coregionalization model

Input_dim Input dimensionality

Num_outputs Number of outputs

Parameters

- **CK** – List of coregionalized kernels (i.e., this will be multiplied by a coregionalize kernel).
- **K** – List of kernels that will be added up together with CK, but won't be multiplied by a coregionalize kernel
- **W_columns** (*integer*) – number tuples of the corregionalization parameters 'coregion_W'

GPY.util.netpbmfile module

Read and write image data from respectively to Netpbm files.

This implementation follows the Netpbm format specifications at <http://netpbm.sourceforge.net/doc/>. No gamma correction is performed.

The following image formats are supported: PBM (bi-level), PGM (grayscale), PPM (color), PAM (arbitrary), XV thumbnail (RGB32, read-only).

Author Christoph Gohlke

Organization Laboratory for Fluorescence Dynamics, University of California, Irvine

Version 2013.01.18

Requirements

- CPython 2.7, 3.2 or 3.3
- Numpy 1.7
- Matplotlib 1.2 (optional for plotting)

Examples

```
>>> im1 = numpy.array([[0, 1], [65534, 65535]], dtype=numpy.uint16)
>>> imsave('_tmp.pgm', im1)
>>> im2 = imread('_tmp.pgm')
>>> assert numpy.all(im1 == im2)
```

`GPY.util.netpbmfile.imread(filename, *args, **kwargs)`

Return image data from Netpbm file as numpy array.

args and *kwargs* are arguments to `NetpbmFile.asarray()`.

```
>>> image = imread('_tmp.pgm')
```

`GPY.util.netpbmfile.imsave(filename, data, maxval=None, pam=False)`

Write image data to Netpbm file.


```
>>> image = numpy.array([[0, 1],[65534, 65535]], dtype=numpy.uint16)
>>> imsave('_tmp.pgm', image)
```

class GPy.util.netpbmfile.**NetpbmFile** (*arg=None, **kwargs*)

Bases: object

Read and write Netpbm PAM, PBM, PGM, PPM, files.

asarray (*copy=True, cache=False, **kwargs*)

Return image data from file as numpy array.

close ()

Close open file. Future asarray calls might fail.

write (*arg, **kwargs*)

Write instance to file.

GPy.util.plot module

GPy.util.plot.**align_subplot_array** (*axes, xlim=None, ylim=None*)

make all of the axes in the array have the same limits, turn off unnecessary ticks

use `pb.subplots()` to get an array of axes

GPy.util.plot.**align_subplots** (*N, M, xlim=None, ylim=None*)

make all of the subplots have the same limits, turn off unnecessary ticks

GPy.util.plot.**fewerXticks** (*ax=None, divideby=2*)

GPy.util.plot.**gpplot** (*x, mu, lower, upper, edgocol='#204a87', fillcol='#729fcf', axes=None, **kwargs*)

GPy.util.plot.**removeRightTicks** (*ax=None*)

GPy.util.plot.**removeUpperTicks** (*ax=None*)

GPy.util.plot.**x_frame1D** (*X, plot_limits=None, resolution=None*)

Internal helper function for making plots, returns a set of input values to plot as well as lower and upper limits

GPy.util.plot.**x_frame2D** (*X, plot_limits=None, resolution=None*)

Internal helper function for making plots, returns a set of input values to plot as well as lower and upper limits

GPy.util.plot_latent module

GPy.util.plot_latent.**most_significant_input_dimensions** (*model, which_indices*)

GPy.util.plot_latent.**plot_latent** (*model, labels=None, which_indices=None, resolution=50, ax=None, marker='o', s=40, fignum=None, plot_inducing=False, legend=True, aspect='auto', updates=False*)

Parameters

- **labels** – a np.array of size `model.num_data` containing labels for the points (can be number, strings, etc)
- **resolution** – the resolution of the grid on which to evaluate the predictive variance

```
GPy.util.plot_latent.plot_magnification(model, labels=None, which_indices=None,
                                         resolution=60, ax=None, marker='o', s=40,
                                         fignum=None, plot_inducing=False, legend=True,
                                         aspect='auto', updates=False)
```

Parameters

- **labels** – a np.array of size model.num_data containing labels for the points (can be number, strings, etc)
- **resolution** – the resolution of the grid on which to evaluate the predictive variance

GPy.util.squashers module

```
GPy.util.squashers.sigmoid(x)
```

```
GPy.util.squashers.single_softmax(x)
```

```
GPy.util.squashers.softmax(x)
```

GPy.util.subarray_and_sorting module

```
GPy.util.subarray_and_sorting.common_subarrays(X, axis=0)
```

Find common subarrays of 2 dimensional X, where axis is the axis to apply the search over. Common subarrays are returned as a dictionary of <subarray, [index]> pairs, where the subarray is a tuple representing the subarray and the index is the index for the subarray in X, where index is the index to the remaining axis.

:param np.ndarray X: 2d array to check for common subarrays in :param int axis: axis to apply subarray detection over.

When the index is 0, compare rows, columns, otherwise.

In a 2d array: >>> import numpy as np >>> X = np.zeros((3,6), dtype=bool) >>> X[[1,1,1],[0,4,5]] = 1; X[1:,[2,3]] = 1 >>> X.array([[False, False, False, False, False, False],

[True, False, True, True, True, True], [False, False, True, True, False, False]], dtype=bool)

```
>>> d = common_subarrays(X,axis=1)
>>> len(d)
3
>>> X[:, d[tuple(X[:,0])]]
array([[False, False, False],
       [ True,  True,  True],
       [False, False, False]], dtype=bool)
>>> d[tuple(X[:,4])] == d[tuple(X[:,0])] == [0, 4, 5]
True
>>> d[tuple(X[:,1])]
[1]
```

GPy.util.symbolic module

```
class GPy.util.symbolic.dh_dd_i
```

Bases: Function

```
classmethod eval(t, tprime, d_i, d_j, l)
```

```
nargs = 5
```

```

class GPy.util.symbolic.dh_dd_j
    Bases: Function

    classmethod eval (t, tprime, d_i, d_j, l)

    nargs = 5

class GPy.util.symbolic.dh_dl
    Bases: Function

    classmethod eval (t, tprime, d_i, d_j, l)

    nargs = 5

class GPy.util.symbolic.dh_dt
    Bases: Function

    classmethod eval (t, tprime, d_i, d_j, l)

    nargs = 5

class GPy.util.symbolic.dh_dtprime
    Bases: Function

    classmethod eval (t, tprime, d_i, d_j, l)

    nargs = 5

class GPy.util.symbolic.erfc
    Bases: Function

    classmethod eval (arg)

    nargs = 1

class GPy.util.symbolic.erfcx
    Bases: Function

    classmethod eval (arg)

    nargs = 1

class GPy.util.symbolic.h
    Bases: Function

    classmethod eval (t, tprime, d_i, d_j, l)

    fdiff (argindex=5)

    nargs = 5

class GPy.util.symbolic.ln_diff_erf
    Bases: Function

    classmethod eval (x0, x1)

    fdiff (argindex=2)

    nargs = 2

```

GPy.util.univariate_Gaussian module

```

GPy.util.univariate_Gaussian.inv_std_norm_cdf(x)
    Inverse cumulative standard Gaussian distribution Based on Winitzki, S. (2008)

```

`GPy.util.univariate_Gaussian.std_norm_cdf(x)`

Cumulative standard Gaussian distribution Based on Abramowitz, M. and Stegun, I. (1970)

`GPy.util.univariate_Gaussian.std_norm_pdf(x)`

Standard Gaussian density function

GPy.util.visualize module

`GPy.util.visualize.data_play(Y, visualizer, frame_rate=30)`

Play a data set using the data_show object given.

Y the data set to be visualized.

Parameters **visualizer** (*data_show*) – the data show object whether to display during optimisation

Example usage:

This example loads in the CMU mocap database (<http://mocap.cs.cmu.edu>) subject number 35 motion number 01. It then plays it using the mocap_show visualize object.

```
data = GPy.util.datasets.cmu_mocap(subject='35', train_motions=['01'])
Y = data['Y']
Y[:, 0:3] = 0. # Make figure walk in place
visualize = GPy.util.visualize.skeleton_show(Y[0, :], data['skel'])
GPy.util.visualize.data_play(Y, visualize)
```

class `GPy.util.visualize.data_show(vals)`

The data_show class is a base class which describes how to visualize a particular data set. For example, motion capture data can be plotted as a stick figure, or images are shown using imshow. This class enables latent to data visualizations for the GP-LVM.

close()

modify (*vals*)

class `GPy.util.visualize.image_show(vals, axes=None, dimensions=(16, 16), transpose=False, order='C', invert=False, scale=False, palette=[], preset_mean=0.0, preset_std=1.0, select_image=0)`

Bases: `GPy.util.visualize.matplotlib_show`

Show a data vector as an image. This visualizer reshapes the output vector and displays it as an image.

Parameters

- **vals** (*axes handle*) – the values of the output to display.
- **axes** – the axes to show the output on.
- **dimensions** (*tuple*) – the dimensions that the image needs to be transposed to for display.
- **transpose** – whether to transpose the image before display.
- **order** (*string*) – whether array is in Fortran ordering ('F') or Python ordering ('C'). Default is python ('C').
- **invert** (*bool*) – whether to invert the pixels or not (default False).
- **palette** – a palette to use for the image.
- **preset_mean** (*double*) – the preset mean of a scaled image.
- **preset_std** (*double*) – the preset standard deviation of a scaled image.

modify (*vals*)

```

    set_image (vals)

```

class GPy.util.visualize.**lvm** (vals, model, data_visualize, latent_axes=None, sense_axes=None, latent_index=[0, 1])

Bases: GPy.util.visualize.matplotlib_show

```

    modify (vals)
        When latent values are modified update the latent representation and also update the output visualization.

    on_click (event)

    on_enter (event)

    on_leave (event)

    on_move (event)

    show_sensitivities ()

```

class GPy.util.visualize.**lvm_dimselect** (vals, model, data_visualize, latent_axes=None, sense_axes=None, latent_index=[0, 1], labels=None)

Bases: GPy.util.visualize.lvm

A visualizer for latent variable models which allows selection of the latent dimensions to use by clicking on a bar chart of their length scales.

For an example of the visualizer's use try:

```

GPy.examples.dimensionality_reduction.BGPVLM_oil()

```

```

    on_click (event)

    on_leave (event)

```

class GPy.util.visualize.**lvm_subplots** (vals, Model, data_visualize, latent_axes=None, sense_axes=None)

Bases: GPy.util.visualize.lvm

latent_axes is a np array of dimension np.ceil(input_dim/2), one for each pair of the latent dimensions.

class GPy.util.visualize.**matplotlib_show** (vals, axes=None)

Bases: GPy.util.visualize.data_show

the matplotlib_show class is a base class for all visualization methods that use matplotlib. It is initialized with an axis. If the axis is set to None it creates a figure window.

```

    close ()

```

class GPy.util.visualize.**mocap_data_show** (vals, axes=None, connect=None)

Bases: GPy.util.visualize.matplotlib_show

Base class for visualizing motion capture data.

```

    draw_edges ()

    draw_vertices ()

    finalize_axes ()

    finalize_axes_modify ()

    initialize_axes ()
        Set up the axes with the right limits and scaling.

    initialize_axes_modify ()

    modify (vals)

    process_values ()

```

```
class GPy.util.visualize.mocap_data_show_vpython(vals, scene=None, connect=None, radius=0.1)
```

Bases: `GPy.util.visualize.vpython_show`

Base class for visualizing motion capture data using visual module.

`draw_edges()`

`draw_vertices()`

`modify(vals)`

`modify_edges()`

`modify_vertices()`

`pos_axis(i,j)`

`process_values()`

```
class GPy.util.visualize.skeleton_show(vals, skel, scene=None, padding=0)
```

Bases: `GPy.util.visualize.mocap_data_show_vpython`

`data_show` class for visualizing motion capture data encoded as a skeleton with angles.

`process_values()`

Takes a set of angles and converts them to the x,y,z coordinates in the internal prerepresentation of the class, ready for plotting.

Parameters `vals` – the values that are being modelled.

`wrap_around(lim, connect)`

```
class GPy.util.visualize.stick_show(vals, connect=None, scene=None)
```

Bases: `GPy.util.visualize.mocap_data_show_vpython`

Show a three dimensional point cloud as a figure. Connect elements of the figure together using the matrix connect.

`process_values()`

```
class GPy.util.visualize.vector_show(vals, axes=None)
```

Bases: `GPy.util.visualize.matplotlib_show`

A base visualization class that just shows a data vector as a plot of vector elements alongside their indices.

`modify(vals)`

```
class GPy.util.visualize.vpython_show(vals, scene=None)
```

Bases: `GPy.util.visualize.data_show`

the `vpython_show` class is a base class for all visualization methods that use `vpython` to display. It is initialized with a scene. If the scene is set to `None` it creates a scene window.

`close()`

GPy.util.warping_functions module

```
class GPy.util.warping_functions.TanhWarpingFunction(n_terms=3)
```

Bases: `GPy.util.warping_functions.WarpingFunction`

`f(y, psi)`

transform `y` with `f` using parameter vector `psi` `psi = [[a,b,c]]` ::math:: $f = \sum_{terms} a * \tanh(b*(y+c))$

f_inv (*y*, *psi*, *iterations=10*)
calculate the numerical inverse of *f*

Parameters *iterations* – number of N.R. iterations

fgrad_y (*y*, *psi*, *return_precalc=False*)
gradient of *f* w.r.t to *y* ([N x 1]) returns: Nx1 vector of derivatives, unless *return_precalc* is true, then it also returns the precomputed stuff

fgrad_y_psi (*y*, *psi*, *return_covar_chain=False*)
gradient of *f* w.r.t to *y* and *psi*
returns: NxIx3 tensor of partial derivatives

class GPy.util.warping_functions.**TanhWarpingFunction_d** (*n_terms=3*)
Bases: GPy.util.warping_functions.WarpingFunction

f (*y*, *psi*)
Transform *y* with *f* using parameter vector *psi* *psi* = [[a,b,c]]
$$f = \sum_{terms} a * \tanh(b * (y + c))$$

f_inv (*z*, *psi*, *max_iterations=1000*, *y=None*)
calculate the numerical inverse of *f*

Parameters *max_iterations* – maximum number of N.R. iterations

fgrad_y (*y*, *psi*, *return_precalc=False*)
gradient of *f* w.r.t to *y* ([N x 1])

Returns Nx1 vector of derivatives, unless *return_precalc* is true, then it also returns the precomputed stuff

fgrad_y_psi (*y*, *psi*, *return_covar_chain=False*)
gradient of *f* w.r.t to *y* and *psi*

Returns NxIx4 tensor of partial derivatives

class GPy.util.warping_functions.**WarpingFunction**
Bases: object

abstract function for warping $z = f(y)$

f (*y*, *psi*)
function transformation *y* is a list of values (GP training data) of shape [N,1]

f_inv (*z*, *psi*)
inverse function transformation

fgrad_y (*y*, *psi*)
gradient of *f* w.r.t to *y*

fgrad_y_psi (*y*, *psi*)
gradient of *f* w.r.t to *y*

plot (*psi*, *xmin*, *xmax*)

Module contents

1.1.2 Submodules

1.1.3 GPpy.models module

Implementations for common models used in GP regression and classification. The different models can be viewed in `GPpy.models_modules`, which holds detailed explanations for the different models.

Note: This module is a convenience module for endusers to use. For developers see `GPpy.models_modules`, which holds the implementations for each model.:

1.1.4 Module contents

`GPpy.read(fname)`

`GPpy.tests()`

Indices and tables

- *genindex*
- *modindex*
- *search*

g

- GPpy, 116
- GPpy.core, 19
- GPpy.core.domains, 3
- GPpy.core.fitc, 3
- GPpy.core.gp, 4
- GPpy.core.gp_base, 5
- GPpy.core.mapping, 7
- GPpy.core.model, 8
- GPpy.core.parameterized, 11
- GPpy.core.priors, 12
- GPpy.core.sparse_gp, 14
- GPpy.core.svigp, 16
- GPpy.core.transformations, 17
- GPpy.examples, 23
- GPpy.examples.classification, 19
- GPpy.examples.dimensionality_reduction, 20
- GPpy.examples.non_gaussian, 21
- GPpy.examples.regression, 21
- GPpy.examples.stochastic, 22
- GPpy.examples.tutorials, 22
- GPpy.inference, 26
- GPpy.inference.conjugate_gradient_descent, 23
- GPpy.inference.gradient_descent_update_rules, 24
- GPpy.inference.optimization, 24
- GPpy.inference.samplers, 25
- GPpy.inference.scg, 25
- GPpy.inference.sgd, 26
- GPpy.kern, 55
- GPpy.kern.constructors, 46
- GPpy.kern.kern, 53
- GPpy.kern.parts, 46
- GPpy.kern.parts.bias, 30
- GPpy.kern.parts.Brownian, 27
- GPpy.kern.parts.coregionalize, 30
- GPpy.kern.parts.eq_odel, 31
- GPpy.kern.parts.exponential, 32
- GPpy.kern.parts.finite_dimensional, 33
- GPpy.kern.parts.fixed, 33
- GPpy.kern.parts.gibbs, 33
- GPpy.kern.parts.hetero, 34
- GPpy.kern.parts.hierarchical, 35
- GPpy.kern.parts.independent_outputs, 35
- GPpy.kern.parts.kernpart, 35
- GPpy.kern.parts.linear, 36
- GPpy.kern.parts.Matern32, 27
- GPpy.kern.parts.Matern52, 28
- GPpy.kern.parts.mlp, 37
- GPpy.kern.parts.ODE_1, 29
- GPpy.kern.parts.ODE_UY, 29
- GPpy.kern.parts.periodic_exponential, 39
- GPpy.kern.parts.periodic_Matern32, 38
- GPpy.kern.parts.periodic_Matern52, 38
- GPpy.kern.parts.poly, 40
- GPpy.kern.parts.prod, 40
- GPpy.kern.parts.prod_orthogonal, 41
- GPpy.kern.parts.rational_quadratic, 41
- GPpy.kern.parts.rbf, 42
- GPpy.kern.parts.rbf_inv, 43
- GPpy.kern.parts.rbfcos, 43
- GPpy.kern.parts.spline, 44
- GPpy.kern.parts.symmetric, 44
- GPpy.kern.parts.sympy_helpers, 45
- GPpy.kern.parts.sympykern, 45
- GPpy.kern.parts.white, 45
- GPpy.likelihoods, 77
- GPpy.likelihoods.ep, 72
- GPpy.likelihoods.ep_mixed_noise, 73
- GPpy.likelihoods.gaussian, 73
- GPpy.likelihoods.gaussian_mixed_noise, 74
- GPpy.likelihoods.laplace, 74
- GPpy.likelihoods.likelihood, 75
- GPpy.likelihoods.noise_model_constructors, 76
- GPpy.likelihoods.noise_models, 72
- GPpy.likelihoods.noise_models.bernoulli_noise, 55

[GPY.likelihoods.noise_models.exponential_noise](#), 57
[GPY.likelihoods.noise_models.gamma_noise](#), 59
[GPY.likelihoods.noise_models.gaussian_noise](#), 61
[GPY.likelihoods.noise_models.gp_transformation](#), 63
[GPY.likelihoods.noise_models.noise_distribution](#), 65
[GPY.likelihoods.noise_models.poisson_noise](#), 67
[GPY.likelihoods.noise_models.student_t_noise](#), 69
[GPY.mappings](#), 78
[GPY.mappings.kernel](#), 77
[GPY.mappings.linear](#), 77
[GPY.mappings.mlp](#), 77
[GPY.models](#), 116
[GPY.models_modules](#), 89
[GPY.models_modules.bayesian_gplvm](#), 78
[GPY.models_modules.bcgplvm](#), 80
[GPY.models_modules.fitc_classification](#), 80
[GPY.models_modules.gp_classification](#), 81
[GPY.models_modules.gp_multioutput_regression](#), 81
[GPY.models_modules.gp_regression](#), 82
[GPY.models_modules.gplvm](#), 82
[GPY.models_modules.gradient_checker](#), 83
[GPY.models_modules.mrd](#), 83
[GPY.models_modules.sparse_gp_classification](#), 85
[GPY.models_modules.sparse_gp_multioutput_regression](#), 86
[GPY.models_modules.sparse_gp_regression](#), 87
[GPY.models_modules.sparse_gplvm](#), 87
[GPY.models_modules.svgp_regression](#), 88
[GPY.models_modules.warped_gp](#), 88
[GPY.testing](#), 95
[GPY.testing.bcgplvm_tests](#), 89
[GPY.testing.bgplvm_tests](#), 89
[GPY.testing.cgd_tests](#), 89
[GPY.testing.examples_tests](#), 89
[GPY.testing.gp_transformation_tests](#), 90
[GPY.testing.gplvm_tests](#), 90
[GPY.testing.kernel_tests](#), 90
[GPY.testing.likelihood_tests](#), 91
[GPY.testing.mapping_tests](#), 92
[GPY.testing.mrd_tests](#), 92
[GPY.testing.prior_tests](#), 92
[GPY.testing.psi_stat_expectation_tests](#), 93
[GPY.testing.psi_stat_gradient_tests](#), 93
[GPY.testing.sparse_gplvm_tests](#), 94
[GPY.testing.unit_tests](#), 94
[GPY.util](#), 116
[GPY.util.block_matrices](#), 98
[GPY.util.classification](#), 98
[GPY.util.config](#), 98
[GPY.util.datasets](#), 98
[GPY.util.decorators](#), 100
[GPY.util.diag](#), 100
[GPY.util.erfcx](#), 102
[GPY.util.latent_space_visualizations](#), 97
[GPY.util.latent_space_visualizations.controllers](#), 97
[GPY.util.latent_space_visualizations.controllers.async](#), 96
[GPY.util.latent_space_visualizations.controllers.interactive](#), 97
[GPY.util.linalg](#), 102
[GPY.util.ln_diff_erfs](#), 105
[GPY.util.misc](#), 105
[GPY.util.mocap](#), 106
[GPY.util.multioutput](#), 108
[GPY.util.netpbmfile](#), 108
[GPY.util.plot](#), 109
[GPY.util.plot_latent](#), 109
[GPY.util.squashers](#), 110
[GPY.util.subarray_and_sorting](#), 110
[GPY.util.symbolic](#), 110
[GPY.util.Tango](#), 97
[GPY.util.univariate_Gaussian](#), 111
[GPY.util.visualization](#), 112
[GPY.util.warping_functions](#), 114

A

acclaim_skeleton (class in GPy.util.mocap), 106
 activate() (GPy.util.latent_space_visualizations.controllers.axis_event_controller.AxisEventController method), 96
 adapt_learning_rate() (GPy.inference.sgd.opt_SGD method), 26
 add() (GPy.kern.kern.kern method), 53
 add() (in module GPy.util.diag), 100
 align_subplot_array() (in module GPy.util.plot), 109
 align_subplots() (in module GPy.util.plot), 109
 all_constrained_indices() (GPy.core.parameterized.Parameterized method), 11
 ard() (in module GPy.testing.psi_stat_expectation_tests), 93
 asarray() (GPy.util.netpbmfile.NetpbmFile method), 109
 async_callback_collect() (GPy.inference.conjugate_gradient_descent.AsyncOptimize method), 23
 AsyncOptimize (class in GPy.inference.conjugate_gradient_descent), 23
 at_least_one_element() (in module GPy.models_modules.gradient_checker), 83
 authorize_download() (in module GPy.util.datasets), 98
 auto_scale_factor (GPy.models_modules.mrd.MRD attribute), 84
 AxisChangedController (class in GPy.util.latent_space_visualizations.controllers.axis_event_controller), 96
 AxisEventController (class in GPy.util.latent_space_visualizations.controllers.axis_event_controller), 96

B

backsub_both_sides() (in module GPy.util.linalg), 102
 BayesianGPLVM (class in GPy.models_modules.bayesian_gplvm), 78
 BayesianGPLVMWithMissingData (class in GPy.models_modules.bayesian_gplvm), 79
 BCGPLVM (class in GPy.models_modules.bcgplvm), 80
 bcgplvm_linear_stick() (in module GPy.examples.dimensionality_reduction), 20
 bcgplvm_stick() (in module GPy.examples.dimensionality_reduction), 20
 BCGPLVMTests (class in GPy.testing.bcgplvm_tests), 89
 Bernoulli (class in GPy.likelihoods.noise_models.bernoulli_noise), 55
 bernoulli() (in module GPy.likelihoods.noise_model_constructors), 76
 bgplvm_oil() (in module GPy.examples.dimensionality_reduction), 20
 bgplvm_simulation() (in module GPy.examples.dimensionality_reduction), 20
 bgplvm_test_model() (in module GPy.examples.dimensionality_reduction), 20
 BGPLVMTests (class in GPy.testing.bgplvm_tests), 89
 Bias (class in GPy.kern.parts.bias), 30
 bias() (in module GPy.kern.constructors), 47
 boston_example() (in module GPy.examples.non_gaussian), 21
 boston_housing() (in module GPy.util.datasets), 98
 branch_str() (GPy.util.mocap.tree method), 107
 brendan_faces() (in module GPy.examples.dimensionality_reduction), 20
 brendan_faces() (in module GPy.util.datasets), 98
 Brownian (class in GPy.kern.parts.Brownian), 27
 Brownian() (in module GPy.kern.constructors), 46
 BufferedAxisChangedController (class in GPy.util.latent_space_visualizations.controllers.axis_event_controller), 96
 build_lcm() (in module GPy.kern.constructors), 47
 build_lcm() (in module GPy.util.multioutput), 108

C

callback() (GPy.inference.conjugate_gradient_descent.AsyncGP.optimize method), 23

CGD (class in GPy.inference.conjugate_gradient_descent), 23

chain_1() (in module GPy.util.misc), 105

chain_2() (in module GPy.util.misc), 105

chain_3() (in module GPy.util.misc), 105

check_for_missing() (GPy.inference.sgd.opt_SGD method), 26

check_model() (GPy.testing.unit_tests.GradientTests method), 94

checkgrad() (GPy.core.model.Model method), 8

chol_inv() (in module GPy.util.linalg), 102

cholupdate() (in module GPy.util.linalg), 102

close() (GPy.util.netpbmfile.NetpbmFile method), 109

close() (GPy.util.visualize.data_show method), 112

close() (GPy.util.visualize.matplotlib_show method), 113

close() (GPy.util.visualize.vpython_show method), 114

cmu_mocap() (in module GPy.util.datasets), 98

cmu_mocap_35_walk_jog() (in module GPy.util.datasets), 98

cmu_mocap_49_balance() (in module GPy.util.datasets), 98

cmu_urls_files() (in module GPy.util.datasets), 98

common_subarrays() (in module GPy.util.subarray_and_sorting), 110

compute_param_slices() (GPy.kern.kern.kern method), 53

compute_psi_stats() (GPy.kern.parts.sympykern.spkern method), 45

conf_matrix() (in module GPy.util.classification), 98

connection_matrix() (GPy.util.mocap.skeleton method), 107

constrain() (GPy.core.parameterized.Parameterized method), 11

constrain_bounded() (GPy.core.parameterized.Parameterized method), 11

constrain_fixed() (GPy.core.parameterized.Parameterized method), 11

constrain_negative() (GPy.core.parameterized.Parameterized method), 11

constrain_positive() (GPy.core.parameterized.Parameterized method), 11

copy() (GPy.core.parameterized.Parameterized method), 11

coregionalization_sparse() (in module GPy.examples.regression), 21

coregionalization_toy2() (in module GPy.examples.regression), 21

Coregionalize (class in GPy.kern.parts.coregionalize), 30

coregionalize() (in module GPy.kern.constructors), 47

crisp_data() (in module GPy.util.datasets), 99

crescent_data() (in module GPy.examples.classification), 19

crescent_data() (in module GPy.util.datasets), 99

currentDark() (in module GPy.util.Tango), 97

currentLight() (in module GPy.util.Tango), 97

currentMedium() (in module GPy.util.Tango), 97

D

d2logpdf_df2() (GPy.likelihoods.noise_models.noise_distributions.NoiseDistributions method), 65

d2logpdf_df2_dtheta() (GPy.likelihoods.noise_models.noise_distributions.NoiseDistributions method), 65

d2logpdf_dlink2() (GPy.likelihoods.noise_models.bernoulli_noise.BernoulliNoise method), 56

d2logpdf_dlink2() (GPy.likelihoods.noise_models.exponential_noise.ExponentialNoise method), 57

d2logpdf_dlink2() (GPy.likelihoods.noise_models.gamma_noise.GammaNoise method), 59

d2logpdf_dlink2() (GPy.likelihoods.noise_models.gaussian_noise.GaussianNoise method), 61

d2logpdf_dlink2() (GPy.likelihoods.noise_models.noise_distributions.NoiseDistributions method), 65

d2logpdf_dlink2() (GPy.likelihoods.noise_models.poisson_noise.PoissonNoise method), 67

d2logpdf_dlink2() (GPy.likelihoods.noise_models.student_t_noise.StudentT method), 69

d2logpdf_dlink2_dtheta() (GPy.likelihoods.noise_models.gaussian_noise.GaussianNoise method), 61

d2logpdf_dlink2_dtheta() (GPy.likelihoods.noise_models.noise_distributions.NoiseDistributions method), 65

d2logpdf_dlink2_dtheta() (GPy.likelihoods.noise_models.student_t_noise.StudentT method), 70

d2logpdf_dlink2_dvar() (GPy.likelihoods.noise_models.gaussian_noise.GaussianNoise method), 61

d2logpdf_dlink2_dvar() (GPy.likelihoods.noise_models.student_t_noise.StudentT method), 70

d2transf_df2() (GPy.likelihoods.noise_models.gp_transformations.GPTransformations method), 63

d2transf_df2() (GPy.likelihoods.noise_models.gp_transformations.Heaviside method), 64

d2transf_df2() (GPy.likelihoods.noise_models.gp_transformations.Identity method), 64

d2transf_df2() (GPy.likelihoods.noise_models.gp_transformations.Log method), 64

d2transf_df2() (GPy.likelihoods.noise_models.gp_transformations.Log_ex method), 64

d2transf_df2() (GPy.likelihoods.noise_models.gp_transformations.Probit method), 65

d2transf_df2()	(GPy.likelihoods.noise_models.gp_transformations.NoiseDistribution method), 65	dK_dtheta()	(GPy.kern.parts.Brownian.Brownian method), 30
d3logpdf_df3()	(GPy.likelihoods.noise_models.noise_distributions.NoiseDistribution method), 65	dK_dtheta()	(GPy.kern.parts.coregionalize.Coregionalize method), 31
d3logpdf_dlink3()	(GPy.likelihoods.noise_models.bernoulli_noise.Bernoulli method), 56	dK_dtheta()	(GPy.kern.parts.eq_ode1.Eq_ode1 method), 58
d3logpdf_dlink3()	(GPy.likelihoods.noise_models.exponential_noise.Exponential method), 58	dK_dtheta()	(GPy.kern.parts.exponential.Exponential method), 33
d3logpdf_dlink3()	(GPy.likelihoods.noise_models.gamma_noise.Gamma method), 59	dK_dtheta()	(GPy.kern.parts.finite_dimensional.FiniteDimensional method), 33
d3logpdf_dlink3()	(GPy.likelihoods.noise_models.gaussian_noise.Gaussian method), 61	dK_dtheta()	(GPy.kern.parts.fixed.Fixed method), 34
d3logpdf_dlink3()	(GPy.likelihoods.noise_models.noise_distributions.NoiseDistributions.gibbs.Gibbs method), 66	dK_dtheta()	(GPy.kern.parts.hetero.Hetero method), 35
d3logpdf_dlink3()	(GPy.likelihoods.noise_models.poisson_noise.Poisson method), 68	dK_dtheta()	(GPy.kern.parts.hierarchical.Hierarchical method), 35
d3logpdf_dlink3()	(GPy.likelihoods.noise_models.student_t_noise.StudentT method), 70	dK_dtheta()	(GPy.kern.parts.independent_outputs.IndependentOutputs method), 35
d3transf_df3()	(GPy.likelihoods.noise_models.gp_transformations.Log method), 63	dK_dtheta()	(GPy.kern.parts.kernpart.Kernpart method), 35
d3transf_df3()	(GPy.likelihoods.noise_models.gp_transformations.Log method), 64	dK_dtheta()	(GPy.kern.parts.linear.Linear method), 36
d3transf_df3()	(GPy.likelihoods.noise_models.gp_transformations.Log method), 64	dK_dtheta()	(GPy.kern.parts.Matern32.Matern32 method), 28
d3transf_df3()	(GPy.likelihoods.noise_models.gp_transformations.Log method), 64	dK_dtheta()	(GPy.kern.parts.Matern52.Matern52 method), 29
d3transf_df3()	(GPy.likelihoods.noise_models.gp_transformations.Log method), 65	dK_dtheta()	(GPy.kern.parts.mlp.MLP method), 37
d3transf_df3()	(GPy.likelihoods.noise_models.gp_transformations.Log method), 65	dK_dtheta()	(GPy.kern.parts.ODE_1.ODE_1 method), 29
d3transf_df3()	(GPy.likelihoods.noise_models.gp_transformations.Log method), 65	dK_dtheta()	(GPy.kern.parts.ODE_UY.ODE_UY method), 30
data_available()	(in module GPy.util.datasets), 99	dK_dtheta()	(GPy.kern.parts.periodic_exponential.PeriodicExponential method), 39
data_details_return()	(in module GPy.util.datasets), 99	dK_dtheta()	(GPy.kern.parts.periodic_Matern32.PeriodicMatern32 method), 38
data_play()	(in module GPy.util.visualize), 112	dK_dtheta()	(GPy.kern.parts.periodic_Matern52.PeriodicMatern52 method), 40
data_show	(class in GPy.util.visualize), 112	dK_dtheta()	(GPy.kern.parts.poly.POLY method), 41
deactivate()	(GPy.util.latent_space_visualizations.controllers.axis_event_controller.AxisEventController method), 96	dK_dtheta()	(GPy.kern.parts.prod.Prod method), 41
deepTest()	(in module GPy.testing), 95	dK_dtheta()	(GPy.kern.parts.prod_orthogonal.prod_orthogonal method), 41
della_gatta_TRP63_gene_expression()	(in module GPy.util.datasets), 99	dK_dtheta()	(GPy.kern.parts.rational_quadratic.RationalQuadratic method), 42
df_dtheta()	(GPy.core.mapping.Mapping method), 7	dK_dtheta()	(GPy.kern.parts.rbf.RBF method), 42
df_dtheta()	(GPy.mappings.kernel.Kernel method), 77	dK_dtheta()	(GPy.kern.parts.rbf_inv.RBFInv method), 43
df_dtheta()	(GPy.mappings.linear.Linear method), 77	dK_dtheta()	(GPy.kern.parts.rbfcos.RBFCos method), 44
df_dX()	(GPy.core.mapping.Mapping method), 7	dK_dtheta()	(GPy.kern.parts.spline.Spline method), 44
df_dX()	(GPy.mappings.kernel.Kernel method), 77	dK_dtheta()	(GPy.kern.parts.symmetric.Symmetric method), 44
df_dX()	(GPy.mappings.linear.Linear method), 77	dK_dtheta()	(GPy.kern.parts.symplekern.spkern method), 45
df_dX()	(GPy.mappings.mlp.MLP method), 78	dK_dtheta()	(GPy.kern.parts.white.White method), 45
dh_dd_i	(class in GPy.util.symbolic), 110	dK_dtheta_old()	(GPy.kern.parts.coregionalize.Coregionalize method), 31
dh_dd_j	(class in GPy.util.symbolic), 110	dK_dX()	(GPy.kern.kern.kern method), 54
dh_dl	(class in GPy.util.symbolic), 111	dK_dX()	(GPy.kern.parts.bias.Bias method), 30
dh_dt	(class in GPy.util.symbolic), 111		
dh_dtprime	(class in GPy.util.symbolic), 111		
divide()	(in module GPy.util.diag), 100		
dK_dtheta()	(GPy.kern.kern.kern method), 54		

`dK_dX()` (GPy.kern.parts.Brownian.Brownian method), 27
`dK_dX()` (GPy.kern.parts.coregionalize.Coregionalize method), 31
`dK_dX()` (GPy.kern.parts.eq_ode1.Eq_ode1 method), 32
`dK_dX()` (GPy.kern.parts.exponential.Exponential method), 32
`dK_dX()` (GPy.kern.parts.fixed.Fixed method), 33
`dK_dX()` (GPy.kern.parts.gibbs.Gibbs method), 34
`dK_dX()` (GPy.kern.parts.hetero.Hetero method), 34
`dK_dX()` (GPy.kern.parts.hierarchical.Hierarchical method), 35
`dK_dX()` (GPy.kern.parts.independent_outputs.IndependentOutputs method), 35
`dK_dX()` (GPy.kern.parts.kernpart.Kernpart method), 35
`dK_dX()` (GPy.kern.parts.linear.Linear method), 36
`dK_dX()` (GPy.kern.parts.Matern32.Matern32 method), 28
`dK_dX()` (GPy.kern.parts.Matern52.Matern52 method), 28
`dK_dX()` (GPy.kern.parts.mlp.MLP method), 37
`dK_dX()` (GPy.kern.parts.poly.POLY method), 40
`dK_dX()` (GPy.kern.parts.prod.Prod method), 41
`dK_dX()` (GPy.kern.parts.prod_orthogonal.prod_orthogonal method), 41
`dK_dX()` (GPy.kern.parts.rational_quadratic.RationalQuadratic method), 42
`dK_dX()` (GPy.kern.parts.rbf.RBF method), 42
`dK_dX()` (GPy.kern.parts.rbf_inv.RBFInv method), 43
`dK_dX()` (GPy.kern.parts.rbfcos.RBFCos method), 44
`dK_dX()` (GPy.kern.parts.symmetric.Symmetric method), 44
`dK_dX()` (GPy.kern.parts.sympykern.spkern method), 45
`dK_dX()` (GPy.kern.parts.white.White method), 45
`dKdiag_dtheta()` (GPy.kern.kern.kern method), 54
`dKdiag_dtheta()` (GPy.kern.parts.bias.Bias method), 30
`dKdiag_dtheta()` (GPy.kern.parts.Brownian.Brownian method), 27
`dKdiag_dtheta()` (GPy.kern.parts.coregionalize.Coregionalize method), 31
`dKdiag_dtheta()` (GPy.kern.parts.eq_ode1.Eq_ode1 method), 32
`dKdiag_dtheta()` (GPy.kern.parts.exponential.Exponential method), 33
`dKdiag_dtheta()` (GPy.kern.parts.finite_dimensional.FiniteDimensional method), 33
`dKdiag_dtheta()` (GPy.kern.parts.gibbs.Gibbs method), 34
`dKdiag_dtheta()` (GPy.kern.parts.hetero.Hetero method), 34
`dKdiag_dtheta()` (GPy.kern.parts.hierarchical.Hierarchical method), 35
`dKdiag_dtheta()` (GPy.kern.parts.independent_outputs.IndependentOutputs method), 35
`dKdiag_dtheta()` (GPy.kern.parts.kernpart.Kernpart method), 35
`dKdiag_dtheta()` (GPy.kern.parts.kernpart.Kernpart_stationary method), 36
`dKdiag_dtheta()` (GPy.kern.parts.linear.Linear method), 36
`dKdiag_dtheta()` (GPy.kern.parts.Matern32.Matern32 method), 28
`dKdiag_dtheta()` (GPy.kern.parts.Matern52.Matern52 method), 29
`dKdiag_dtheta()` (GPy.kern.parts.periodic_exponential.PeriodicExponential method), 39
`dKdiag_dtheta()` (GPy.kern.parts.periodic_Matern32.PeriodicMatern32 method), 38
`dKdiag_dtheta()` (GPy.kern.parts.periodic_Matern52.PeriodicMatern52 method), 39
`dKdiag_dtheta()` (GPy.kern.parts.prod.Prod method), 41
`dKdiag_dtheta()` (GPy.kern.parts.prod_orthogonal.prod_orthogonal method), 41
`dKdiag_dtheta()` (GPy.kern.parts.rational_quadratic.RationalQuadratic method), 42
`dKdiag_dtheta()` (GPy.kern.parts.rbf.RBF method), 42
`dKdiag_dtheta()` (GPy.kern.parts.rbfcos.RBFCos method), 44
`dKdiag_dtheta()` (GPy.kern.parts.spline.Spline method), 44
`dKdiag_dtheta()` (GPy.kern.parts.symmetric.Symmetric method), 44
`dKdiag_dtheta()` (GPy.kern.parts.sympykern.spkern method), 45
`dKdiag_dtheta()` (GPy.kern.parts.white.White method), 45
`dKdiag_dX()` (GPy.kern.kern.kern method), 54
`dKdiag_dX()` (GPy.kern.parts.bias.Bias method), 30
`dKdiag_dX()` (GPy.kern.parts.Brownian.Brownian method), 27
`dKdiag_dX()` (GPy.kern.parts.exponential.Exponential method), 33
`dKdiag_dX()` (GPy.kern.parts.fixed.Fixed method), 33
`dKdiag_dX()` (GPy.kern.parts.gibbs.Gibbs method), 34
`dKdiag_dX()` (GPy.kern.parts.hetero.Hetero method), 34
`dKdiag_dX()` (GPy.kern.parts.hierarchical.Hierarchical method), 35
`dKdiag_dX()` (GPy.kern.parts.independent_outputs.IndependentOutputs method), 35
`dKdiag_dX()` (GPy.kern.parts.kernpart.Kernpart method), 35
`dKdiag_dX()` (GPy.kern.parts.kernpart.Kernpart_stationary method), 36
`dKdiag_dX()` (GPy.kern.parts.linear.Linear method), 36
`dKdiag_dX()` (GPy.kern.parts.Matern32.Matern32 method), 28
`dKdiag_dX()` (GPy.kern.parts.Matern52.Matern52 method), 29

[dKdiag_dX\(\)](#) (GPpy.kern.parts.mlp.MLP method), 38
[dKdiag_dX\(\)](#) (GPpy.kern.parts.poly.POLY method), 40
[dKdiag_dX\(\)](#) (GPpy.kern.parts.prod.Prod method), 41
[dKdiag_dX\(\)](#) (GPpy.kern.parts.prod_orthogonal.prod_orthogonal method), 41
[dKdiag_dX\(\)](#) (GPpy.kern.parts.rational_quadratic.RationalQuadratic method), 42
[dKdiag_dX\(\)](#) (GPpy.kern.parts.rbf.RBF method), 42
[dKdiag_dX\(\)](#) (GPpy.kern.parts.rbf_inv.RBFInv method), 43
[dKdiag_dX\(\)](#) (GPpy.kern.parts.rbfcos.RBFCos method), 44
[dKdiag_dX\(\)](#) (GPpy.kern.parts.spline.Spline method), 44
[dKdiag_dX\(\)](#) (GPpy.kern.parts.symmetric.Symmetric method), 44
[dKdiag_dX\(\)](#) (GPpy.kern.parts.sympykern.spkern method), 45
[dKdiag_dX\(\)](#) (GPpy.kern.parts.white.White method), 45
[dKL_dmuS\(\)](#) (GPpy.models_modules.bayesian_gplvm.BayesianGPLVM method), 78
[dL_dmuS\(\)](#) (GPpy.models_modules.bayesian_gplvm.BayesianGPLVM method), 78
[dL_dtheta\(\)](#) (GPpy.core.fitc.FITC method), 4
[dL_dtheta\(\)](#) (GPpy.core.sparse_gp.SparseGP method), 14
[dL_dtheta\(\)](#) (GPpy.core.svgp.SVIGP method), 17
[dL_dX\(\)](#) (GPpy.models_modules.sparse_gplvm.SparseGPLVM method), 87
[dL_dZ\(\)](#) (GPpy.core.fitc.FITC method), 4
[dL_dZ\(\)](#) (GPpy.core.sparse_gp.SparseGP method), 14
[dlogpdf_df\(\)](#) (GPpy.likelihoods.noise_models.noise_distributions.NoiseDistributions method), 66
[dlogpdf_df_dtheta\(\)](#) (GPpy.likelihoods.noise_models.noise_distributions.NoiseDistributions method), 66
[dlogpdf_dlink\(\)](#) (GPpy.likelihoods.noise_models.bernoulli_noise.BernoulliNoise method), 56
[dlogpdf_dlink\(\)](#) (GPpy.likelihoods.noise_models.exponential_noise.ExponentialNoise method), 58
[dlogpdf_dlink\(\)](#) (GPpy.likelihoods.noise_models.gamma_noise.GammaNoise method), 60
[dlogpdf_dlink\(\)](#) (GPpy.likelihoods.noise_models.gaussian_noise.GaussianNoise method), 62
[dlogpdf_dlink\(\)](#) (GPpy.likelihoods.noise_models.noise_distributions.NoiseDistributions method), 66
[dlogpdf_dlink\(\)](#) (GPpy.likelihoods.noise_models.poisson_noise.PoissonNoise method), 68
[dlogpdf_dlink\(\)](#) (GPpy.likelihoods.noise_models.student_t_noise.StudentTNoise method), 70
[dlogpdf_dlink_dtheta\(\)](#) (GPpy.likelihoods.noise_models.gaussian_noise.GaussianNoise method), 62
[dlogpdf_dlink_dtheta\(\)](#) (GPpy.likelihoods.noise_models.noise_distributions.NoiseDistributions method), 66
[dlogpdf_dlink_dtheta\(\)](#) (GPpy.likelihoods.noise_models.student_t_noise.StudentTNoise method), 70
[dlogpdf_dlink_dvar\(\)](#) (GPpy.likelihoods.noise_models.gaussian_noise.GaussianNoise method), 62
[dlogpdf_dlink_dvar\(\)](#) (GPpy.likelihoods.noise_models.student_t_noise.StudentTNoise method), 71
[dlogpdf_dlink_dvar\(\)](#) (GPpy.likelihoods.noise_models.gaussian_noise.GaussianNoise method), 62
[dlogpdf_dlink_dvar\(\)](#) (GPpy.likelihoods.noise_models.student_t_noise.StudentTNoise method), 71
[dmu_dX\(\)](#) (GPpy.models_modules.bayesian_gplvm.BayesianGPLVM method), 78
[dmu_dX\(\)](#) (GPpy.models_modules.bayesian_gplvm.BayesianGPLVM method), 78
[dmu_dX\(\)](#) (GPpy.models_modules.bayesian_gplvm.BayesianGPLVM method), 78
[domain](#) (GPpy.core.priors.Gamma attribute), 12
[domain](#) (GPpy.core.priors.Gaussian attribute), 13
[domain](#) (GPpy.core.priors.inverse_gamma attribute), 14
[domain](#) (GPpy.core.priors.LogGaussian attribute), 13
[domain](#) (GPpy.core.priors.MultivariateGaussian attribute), 13
[domain](#) (GPpy.core.priors.Prior attribute), 13
[domain](#) (GPpy.core.transformations.exponent attribute), 17
[domain](#) (GPpy.core.transformations.logexp attribute), 17
[domain](#) (GPpy.core.transformations.logexp_clipped attribute), 18
[domain](#) (GPpy.core.transformations.logistic attribute), 18
[domain](#) (GPpy.core.transformations.negative_exponent attribute), 18
[domain](#) (GPpy.core.transformations.negative_logexp attribute), 18
[domain](#) (GPpy.core.transformations.square attribute), 18
[domain](#) (GPpy.core.transformations.transformation attribute), 19
[download_data\(\)](#) (in module GPpy.util.datasets), 99
[download_rogers_girolami_data\(\)](#) (in module GPpy.util.datasets), 99
[download_url\(\)](#) (in module GPpy.util.datasets), 99
[dpsi0_dmuS\(\)](#) (GPpy.kern.kern.kern method), 54
[dpsi0_dmuS\(\)](#) (GPpy.kern.parts.bias.Bias method), 30

- dpsi0_dmuS() (GPy.kern.parts.kernpart.Kernpart method), 35
- dpsi0_dmuS() (GPy.kern.parts.linear.Linear method), 36
- dpsi0_dmuS() (GPy.kern.parts.rbf.RBF method), 42
- dpsi0_dmuS() (GPy.kern.parts.white.White method), 46
- dpsi0_dtheta() (GPy.kern.kern.kern method), 54
- dpsi0_dtheta() (GPy.kern.parts.bias.Bias method), 30
- dpsi0_dtheta() (GPy.kern.parts.kernpart.Kernpart method), 36
- dpsi0_dtheta() (GPy.kern.parts.linear.Linear method), 36
- dpsi0_dtheta() (GPy.kern.parts.rbf.RBF method), 42
- dpsi0_dtheta() (GPy.kern.parts.white.White method), 46
- dpsi0_dZ() (GPy.kern.kern.kern method), 54
- dpsi0_dZ() (GPy.kern.parts.bias.Bias method), 30
- dpsi1_dmuS() (GPy.kern.kern.kern method), 54
- dpsi1_dmuS() (GPy.kern.parts.bias.Bias method), 30
- dpsi1_dmuS() (GPy.kern.parts.kernpart.Kernpart method), 36
- dpsi1_dmuS() (GPy.kern.parts.linear.Linear method), 37
- dpsi1_dmuS() (GPy.kern.parts.rbf.RBF method), 42
- dpsi1_dmuS() (GPy.kern.parts.rbf_inv.RBFInv method), 43
- dpsi1_dmuS() (GPy.kern.parts.white.White method), 46
- dpsi1_dtheta() (GPy.kern.kern.kern method), 54
- dpsi1_dtheta() (GPy.kern.parts.bias.Bias method), 30
- dpsi1_dtheta() (GPy.kern.parts.kernpart.Kernpart method), 36
- dpsi1_dtheta() (GPy.kern.parts.linear.Linear method), 37
- dpsi1_dtheta() (GPy.kern.parts.rbf.RBF method), 42
- dpsi1_dtheta() (GPy.kern.parts.rbf_inv.RBFInv method), 43
- dpsi1_dtheta() (GPy.kern.parts.white.White method), 46
- dpsi1_dZ() (GPy.kern.kern.kern method), 54
- dpsi1_dZ() (GPy.kern.parts.bias.Bias method), 30
- dpsi1_dZ() (GPy.kern.parts.kernpart.Kernpart method), 36
- dpsi1_dZ() (GPy.kern.parts.linear.Linear method), 36
- dpsi1_dZ() (GPy.kern.parts.rbf.RBF method), 42
- dpsi1_dZ() (GPy.kern.parts.rbf_inv.RBFInv method), 43
- dpsi1_dZ() (GPy.kern.parts.white.White method), 46
- dpsi2_dmuS() (GPy.kern.kern.kern method), 54
- dpsi2_dmuS() (GPy.kern.parts.bias.Bias method), 30
- dpsi2_dmuS() (GPy.kern.parts.kernpart.Kernpart method), 36
- dpsi2_dmuS() (GPy.kern.parts.linear.Linear method), 37
- dpsi2_dmuS() (GPy.kern.parts.rbf.RBF method), 42
- dpsi2_dmuS() (GPy.kern.parts.rbf_inv.RBFInv method), 43
- dpsi2_dmuS() (GPy.kern.parts.white.White method), 46
- dpsi2_dmuS_new() (GPy.kern.parts.linear.Linear method), 37
- dpsi2_dtheta() (GPy.kern.kern.kern method), 54
- dpsi2_dtheta() (GPy.kern.parts.bias.Bias method), 30
- dpsi2_dtheta() (GPy.kern.parts.kernpart.Kernpart method), 36
- dpsi2_dtheta() (GPy.kern.parts.linear.Linear method), 37
- dpsi2_dtheta() (GPy.kern.parts.rbf.RBF method), 42
- dpsi2_dtheta() (GPy.kern.parts.rbf_inv.RBFInv method), 43
- dpsi2_dtheta() (GPy.kern.parts.white.White method), 46
- dpsi2_dtheta_new() (GPy.kern.parts.linear.Linear method), 37
- dpsi2_dZ() (GPy.kern.kern.kern method), 54
- dpsi2_dZ() (GPy.kern.parts.bias.Bias method), 30
- dpsi2_dZ() (GPy.kern.parts.kernpart.Kernpart method), 36
- dpsi2_dZ() (GPy.kern.parts.linear.Linear method), 37
- dpsi2_dZ() (GPy.kern.parts.rbf.RBF method), 42
- dpsi2_dZ() (GPy.kern.parts.rbf_inv.RBFInv method), 43
- dpsi2_dZ() (GPy.kern.parts.white.White method), 46
- DPSiStatTest (class in GPy.testing.psi_stat_gradient_tests), 93
- draw_edges() (GPy.util.visualize.mocap_data_show method), 113
- draw_edges() (GPy.util.visualize.mocap_data_show_vpython method), 114
- draw_vertices() (GPy.util.visualize.mocap_data_show method), 113
- draw_vertices() (GPy.util.visualize.mocap_data_show_vpython method), 114
- DSYR() (in module GPy.util.linalg), 102
- DSYR_blas() (in module GPy.util.linalg), 102
- DSYR_numpy() (in module GPy.util.linalg), 102
- dtransf_df() (GPy.likelihoods.noise_models.gp_transformations.GPTransformations method), 63
- dtransf_df() (GPy.likelihoods.noise_models.gp_transformations.Heaviside method), 64
- dtransf_df() (GPy.likelihoods.noise_models.gp_transformations.Identity method), 64
- dtransf_df() (GPy.likelihoods.noise_models.gp_transformations.Log method), 64
- dtransf_df() (GPy.likelihoods.noise_models.gp_transformations.Log_ex_1 method), 64
- dtransf_df() (GPy.likelihoods.noise_models.gp_transformations.Probit method), 65
- dtransf_df() (GPy.likelihoods.noise_models.gp_transformations.Reciprocal method), 65
- dtrtrs() (in module GPy.util.linalg), 103
- ## E
- ensure_default_constraints() (GPy.core.model.Model method), 9
- EP (class in GPy.likelihoods.ep), 72
- EP_Mixed_Noise (class in GPy.likelihoods.ep_mixed_noise), 73
- epomeo_gpx() (in module GPy.examples.regression), 21
- Eq_ode1 (class in GPy.kern.parts.eq_ode1), 31

- eq_sympy() (in module GPy.kern.constructors), 48
- erfc (class in GPy.util.symbolic), 111
- erfcx (class in GPy.util.symbolic), 111
- erfcx() (in module GPy.kern.parts.sympy_helpers), 45
- erfcx() (in module GPy.util.erfcx), 102
- eval() (GPy.util.symbolic.dh_dd_i class method), 110
- eval() (GPy.util.symbolic.dh_dd_j class method), 111
- eval() (GPy.util.symbolic.dh_dl class method), 111
- eval() (GPy.util.symbolic.dh_dt class method), 111
- eval() (GPy.util.symbolic.dh_dtprime class method), 111
- eval() (GPy.util.symbolic.erfc class method), 111
- eval() (GPy.util.symbolic.erfcx class method), 111
- eval() (GPy.util.symbolic.h class method), 111
- eval() (GPy.util.symbolic.ln_diff_erf class method), 111
- ExamplesTests (class in GPy.testing.examples_tests), 89
- exponent (class in GPy.core.transformations), 17
- Exponential (class in GPy.kern.parts.exponential), 32
- Exponential (class in GPy.likelihoods.noise_models.exponential_noise), 57
- exponential() (in module GPy.kern.constructors), 48
- exponential() (in module GPy.likelihoods.noise_model_constructors), 76
- exponents() (in module GPy.inference.scg), 25
- extent() (GPy.util.latent_space_visualizations.controllers.axis_controller method), 96
- ## F
- f() (GPy.core.mapping.Mapping method), 7
- f() (GPy.core.transformations.exponent method), 17
- f() (GPy.core.transformations.logexp method), 17
- f() (GPy.core.transformations.logexp_clipped method), 18
- f() (GPy.core.transformations.logistic method), 18
- f() (GPy.core.transformations.negative_exponent method), 18
- f() (GPy.core.transformations.negative_logexp method), 18
- f() (GPy.core.transformations.square method), 18
- f() (GPy.core.transformations.transformation method), 19
- f() (GPy.mappings.kernel.Kernel method), 77
- f() (GPy.mappings.linear.Linear method), 77
- f() (GPy.mappings.mlp.MLP method), 78
- f() (GPy.util.warping_functions.TanhWarpingFunction method), 114
- f() (GPy.util.warping_functions.TanhWarpingFunction_d method), 115
- f() (GPy.util.warping_functions.WarpingFunction method), 115
- f_inv() (GPy.util.warping_functions.TanhWarpingFunction method), 114
- f_inv() (GPy.util.warping_functions.TanhWarpingFunction_d method), 115
- f_inv() (GPy.util.warping_functions.WarpingFunction method), 115
- fast_array_equal() (in module GPy.util.misc), 105
- fast_array_equal2() (in module GPy.util.misc), 105
- fdiff() (GPy.util.symbolic.h method), 111
- fdiff() (GPy.util.symbolic.ln_diff_erf method), 111
- fewerXticks() (in module GPy.util.plot), 109
- fewerXticks() (in module GPy.util.Tango), 97
- fgrad_y() (GPy.util.warping_functions.TanhWarpingFunction method), 115
- fgrad_y() (GPy.util.warping_functions.TanhWarpingFunction_d method), 115
- fgrad_y() (GPy.util.warping_functions.WarpingFunction method), 115
- fgrad_y_psi() (GPy.util.warping_functions.TanhWarpingFunction method), 115
- fgrad_y_psi() (GPy.util.warping_functions.TanhWarpingFunction_d method), 115
- fgrad_y_psi() (GPy.util.warping_functions.WarpingFunction method), 115
- finalize() (GPy.util.mocap.skeleton method), 107
- finalize_axes() (GPy.util.visualize.mocap_data_show method), 113
- finalize_axes_modify() (GPy.util.visualize.mocap_data_show method), 113
- find_children() (GPy.util.mocap.tree method), 107
- find_parents() (GPy.util.mocap.tree method), 107
- find_root() (GPy.util.mocap.tree method), 107
- finite_dimensional() (in module GPy.kern.constructors), 48
- FiniteDimensional (class in GPy.kern.parts.finite_dimensional), 33
- finv() (GPy.core.transformations.exponent method), 17
- finv() (GPy.core.transformations.logexp method), 17
- finv() (GPy.core.transformations.logexp_clipped method), 18
- finv() (GPy.core.transformations.logistic method), 18
- finv() (GPy.core.transformations.negative_exponent method), 18
- finv() (GPy.core.transformations.negative_logexp method), 18
- finv() (GPy.core.transformations.square method), 18
- finv() (GPy.core.transformations.transformation method), 19
- fit_DTC() (GPy.likelihoods.ep.EP method), 72
- fit_DTC() (GPy.likelihoods.ep_mixed_noise.EP_Mixed_Noise method), 73
- fit_FITC() (GPy.likelihoods.ep.EP method), 72
- fit_FITC() (GPy.likelihoods.ep_mixed_noise.EP_Mixed_Noise method), 73
- fit_full() (GPy.likelihoods.ep.EP method), 72
- fit_full() (GPy.likelihoods.ep_mixed_noise.EP_Mixed_Noise method), 73
- fit_full() (GPy.likelihoods.laplace.Laplace method), 74
- fit_full() (GPy.likelihoods.likelihood.likelihood method), 75

- FITC (class in GPy.core.fitc), 3
- FITCClassification (class in GPy.models_modules.fitc_classification), 80
- Fixed (class in GPy.kern.parts.fixed), 33
- fixed() (in module GPy.kern.constructors), 48
- flatten_if_needed() (in module GPy.models_modules.gradient_checker), 83
- flatten_nested() (in module GPy.testing.examples_tests), 89
- FletcherReeves (class in GPy.inference.gradient_descent_update_rules), 24
- from_EV() (GPy.core.priors.Gamma static method), 12
- ## G
- Gamma (class in GPy.core.priors), 12
- Gamma (class in GPy.likelihoods.noise_models.gamma_noise), 59
- gamma() (in module GPy.likelihoods.noise_model_constructors), 76
- gamma_from_EV() (in module GPy.core.priors), 13
- Gaussian (class in GPy.core.priors), 12
- Gaussian (class in GPy.likelihoods.gaussian), 73
- Gaussian (class in GPy.likelihoods.noise_models.gaussian_noise), 61
- gaussian() (in module GPy.likelihoods.noise_model_constructors), 76
- gaussian_ep() (in module GPy.likelihoods.noise_model_constructors), 76
- Gaussian_Mixed_Noise (class in GPy.likelihoods.gaussian_mixed_noise), 74
- GDUpdateRule (class in GPy.inference.gradient_descent_update_rules), 24
- get_blocks() (in module GPy.util.block_matrices), 98
- get_child_xyz() (GPy.util.mocap.acclaim_skeleton method), 106
- get_gradient() (GPy.core.model.Model method), 9
- get_grid() (GPy.util.latent_space_visualizations.controllers.axis_event_controller.BufferedAxisChangedController method), 96
- get_index_by_id() (GPy.util.mocap.tree method), 107
- get_index_by_name() (GPy.util.mocap.tree method), 107
- get_optimizer() (in module GPy.inference.optimization), 25
- get_param_shapes() (GPy.inference.sgd.opt_SGD method), 26
- get_shape() (in module GPy.models_modules.gradient_checker), 83
- get_vb_param() (GPy.core.svgp.SVIGP method), 17
- getstate() (GPy.core.gp.GP method), 4
- getstate() (GPy.core.gp_base.GPBase method), 5
- getstate() (GPy.core.model.Model method), 9
- getstate() (GPy.core.parameterized.Parameterized method), 11
- getstate() (GPy.core.sparse_gp.SparseGP method), 14
- getstate() (GPy.core.svgp.SVIGP method), 17
- getstate() (GPy.kern.kern.kern method), 54
- getstate() (GPy.models_modules.bayesian_gplvm.BayesianGPLVM method), 79
- getstate() (GPy.models_modules.bayesian_gplvm.BayesianGPLVMWithMI method), 79
- getstate() (GPy.models_modules.gp_regression.GPRegression method), 82
- getstate() (GPy.models_modules.gplvm.GPLVM method), 83
- getstate() (GPy.models_modules.mrd.MRD method), 84
- getstate() (GPy.models_modules.sparse_gp_classification.SparseGPClassification method), 85
- getstate() (GPy.models_modules.sparse_gp_regression.SparseGPRegression method), 87
- getstate() (GPy.models_modules.sparse_gplvm.SparseGPLVM method), 87
- getstate() (GPy.models_modules.svgp_regression.SVIGPRegression method), 88
- getstate() (GPy.models_modules.warped_gp.WarpedGP method), 88
- Gibbs (class in GPy.kern.parts.gibbs), 33
- gibbs() (in module GPy.kern.constructors), 49
- GP (class in GPy.core.gp), 4
- GPBase (class in GPy.core.gp_base), 5
- GPClassification (class in GPy.models_modules.gp_classification), 81
- GPLVM (class in GPy.models_modules.gplvm), 82
- gplvm_oil_100() (in module GPy.examples.dimensionality_reduction), 20
- GPLVMTests (class in GPy.testing.gplvm_tests), 90
- GPMultioutputRegression (class in GPy.models_modules.gp_multioutput_regression), 81
- gplot() (in module GPy.util.plot), 109
- GPRegression (class in GPy.models_modules.gp_regression), 82
- GPTransformation (class in GPy.likelihoods.noise_models.gp_transformations), 63
- GPy (module), 116
- GPy.core (module), 19
- GPy.core.domains (module), 3
- GPy.core.fitc (module), 3
- GPy.core.gp (module), 4
- GPy.core.gp_base (module), 5
- GPy.core.mapping (module), 7

GPy.core.model (module), 8
 GPy.core.parameterized (module), 11
 GPy.core.priors (module), 12
 GPy.core.sparse_gp (module), 14
 GPy.core.svgp (module), 16
 GPy.core.transformations (module), 17
 GPy.examples (module), 23
 GPy.examples.classification (module), 19
 GPy.examples.dimensionality_reduction (module), 20
 GPy.examples.non_gaussian (module), 21
 GPy.examples.regression (module), 21
 GPy.examples.stochastic (module), 22
 GPy.examples.tutorials (module), 22
 GPy.inference (module), 26
 GPy.inference.conjugate_gradient_descent (module), 23
 GPy.inference.gradient_descent_update_rules (module), 24
 GPy.inference.optimization (module), 24
 GPy.inference.samplers (module), 25
 GPy.inference.scg (module), 25
 GPy.inference.sgd (module), 26
 GPy.kern (module), 55
 GPy.kern.constructors (module), 46
 GPy.kern.kern (module), 53
 GPy.kern.parts (module), 46
 GPy.kern.parts.bias (module), 30
 GPy.kern.parts.Brownian (module), 27
 GPy.kern.parts.coregionalize (module), 30
 GPy.kern.parts.eq_ode1 (module), 31
 GPy.kern.parts.exponential (module), 32
 GPy.kern.parts.finite_dimensional (module), 33
 GPy.kern.parts.fixed (module), 33
 GPy.kern.parts.gibbs (module), 33
 GPy.kern.parts.hetero (module), 34
 GPy.kern.parts.hierarchical (module), 35
 GPy.kern.parts.independent_outputs (module), 35
 GPy.kern.parts.kernpart (module), 35
 GPy.kern.parts.linear (module), 36
 GPy.kern.parts.Matern32 (module), 27
 GPy.kern.parts.Matern52 (module), 28
 GPy.kern.parts.mlp (module), 37
 GPy.kern.parts.ODE_1 (module), 29
 GPy.kern.parts.ODE_UY (module), 29
 GPy.kern.parts.periodic_exponential (module), 39
 GPy.kern.parts.periodic_Matern32 (module), 38
 GPy.kern.parts.periodic_Matern52 (module), 38
 GPy.kern.parts.poly (module), 40
 GPy.kern.parts.prod (module), 40
 GPy.kern.parts.prod_orthogonal (module), 41
 GPy.kern.parts.rational_quadratic (module), 41
 GPy.kern.parts.rbf (module), 42
 GPy.kern.parts.rbf_inv (module), 43
 GPy.kern.parts.rbfcos (module), 43
 GPy.kern.parts.spline (module), 44
 GPy.kern.parts.symmetric (module), 44
 GPy.kern.parts.sympy_helpers (module), 45
 GPy.kern.parts.sympykern (module), 45
 GPy.kern.parts.white (module), 45
 GPy.likelihoods (module), 77
 GPy.likelihoods.ep (module), 72
 GPy.likelihoods.ep_mixed_noise (module), 73
 GPy.likelihoods.gaussian (module), 73
 GPy.likelihoods.gaussian_mixed_noise (module), 74
 GPy.likelihoods.laplace (module), 74
 GPy.likelihoods.likelihood (module), 75
 GPy.likelihoods.noise_model_constructors (module), 76
 GPy.likelihoods.noise_models (module), 72
 GPy.likelihoods.noise_models.bernoulli_noise (module), 55
 GPy.likelihoods.noise_models.exponential_noise (module), 57
 GPy.likelihoods.noise_models.gamma_noise (module), 59
 GPy.likelihoods.noise_models.gaussian_noise (module), 61
 GPy.likelihoods.noise_models.gp_transformations (module), 63
 GPy.likelihoods.noise_models.noise_distributions (module), 65
 GPy.likelihoods.noise_models.poisson_noise (module), 67
 GPy.likelihoods.noise_models.student_t_noise (module), 69
 GPy.mappings (module), 78
 GPy.mappings.kernel (module), 77
 GPy.mappings.linear (module), 77
 GPy.mappings.mlp (module), 77
 GPy.models (module), 116
 GPy.models_modules (module), 89
 GPy.models_modules.bayesian_gplvm (module), 78
 GPy.models_modules.bcgplvm (module), 80
 GPy.models_modules.ftc_classification (module), 80
 GPy.models_modules.gp_classification (module), 81
 GPy.models_modules.gp_multioutput_regression (module), 81
 GPy.models_modules.gp_regression (module), 82
 GPy.models_modules.gplvm (module), 82
 GPy.models_modules.gradient_checker (module), 83
 GPy.models_modules.mrd (module), 83
 GPy.models_modules.sparse_gp_classification (module), 85
 GPy.models_modules.sparse_gp_multioutput_regression (module), 86
 GPy.models_modules.sparse_gp_regression (module), 87
 GPy.models_modules.sparse_gplvm (module), 87
 GPy.models_modules.svgp_regression (module), 88
 GPy.models_modules.warped_gp (module), 88
 GPy.testing (module), 95

- GPy.testing.bcgplvm_tests (module), 89
 GPy.testing.bgplvm_tests (module), 89
 GPy.testing.cgd_tests (module), 89
 GPy.testing.examples_tests (module), 89
 GPy.testing.gp_transformation_tests (module), 90
 GPy.testing.gplvm_tests (module), 90
 GPy.testing.kernel_tests (module), 90
 GPy.testing.likelihood_tests (module), 91
 GPy.testing.mapping_tests (module), 92
 GPy.testing.mrd_tests (module), 92
 GPy.testing.prior_tests (module), 92
 GPy.testing.psi_stat_expectation_tests (module), 93
 GPy.testing.psi_stat_gradient_tests (module), 93
 GPy.testing.sparse_gplvm_tests (module), 94
 GPy.testing.unit_tests (module), 94
 GPy.util (module), 116
 GPy.util.block_matrices (module), 98
 GPy.util.classification (module), 98
 GPy.util.config (module), 98
 GPy.util.datasets (module), 98
 GPy.util.decorators (module), 100
 GPy.util.diag (module), 100
 GPy.util.erfcx (module), 102
 GPy.util.latent_space_visualizations (module), 97
 GPy.util.latent_space_visualizations.controllers (module), 97
 GPy.util.latent_space_visualizations.controllers.axis_event_controller (module), 96
 GPy.util.latent_space_visualizations.controllers.imshow_controller (module), 97
 GPy.util.linalg (module), 102
 GPy.util.ln_diff_erfs (module), 105
 GPy.util.misc (module), 105
 GPy.util.mocap (module), 106
 GPy.util.multioutput (module), 108
 GPy.util.netpbmfile (module), 108
 GPy.util.plot (module), 109
 GPy.util.plot_latent (module), 109
 GPy.util.squashers (module), 110
 GPy.util.subarray_and_sorting (module), 110
 GPy.util.symbolic (module), 110
 GPy.util.Tango (module), 97
 GPy.util.univariate_Gaussian (module), 111
 GPy.util.visualize (module), 112
 GPy.util.warping_functions (module), 114
 gradfactor() (GPy.core.transformations.exponent method), 17
 gradfactor() (GPy.core.transformations.logexp method), 17
 gradfactor() (GPy.core.transformations.logexp_clipped method), 18
 gradfactor() (GPy.core.transformations.logistic method), 18
 gradfactor() (GPy.core.transformations.negative_exponent method), 18
 gradfactor() (GPy.core.transformations.negative_logexp method), 18
 gradfactor() (GPy.core.transformations.square method), 19
 gradfactor() (GPy.core.transformations.transformation method), 19
 GradientChecker (class in GPy.models_modules.gradient_checker), 83
 GradientTests (class in GPy.testing.unit_tests), 94
 Gram_matrix() (GPy.kern.parts.exponential.Exponential method), 32
 Gram_matrix() (GPy.kern.parts.Matern32.Matern32 method), 27
 Gram_matrix() (GPy.kern.parts.Matern52.Matern52 method), 28
 Gram_matrix() (GPy.kern.parts.periodic_exponential.PeriodicExponential method), 39
 Gram_matrix() (GPy.kern.parts.periodic_Matern32.PeriodicMatern32 method), 38
 Gram_matrix() (GPy.kern.parts.periodic_Matern52.PeriodicMatern52 method), 39
 grep_model() (GPy.core.parameterized.Parameterized method), 11
 group_program_names() (GPy.core.parameterized.Parameterized method), 11
- ## H
- h (class in GPy.util.symbolic), 111
 h() (in module GPy.kern.parts.sympy_helpers), 45
 hapmap3() (in module GPy.util.datasets), 99
 Heaviside (class in GPy.likelihoods.noise_models.gp_transformations), 64
 Hetero (class in GPy.kern.parts.hetero), 34
 hetero() (in module GPy.kern.constructors), 49
 hex2rgb() (in module GPy.util.Tango), 97
 Hierarchical (class in GPy.kern.parts.hierarchical), 35
 hierarchical() (in module GPy.kern.constructors), 49
- ## I
- Identity (class in GPy.likelihoods.noise_models.gp_transformations), 64
 image_show (class in GPy.util.visualize), 112
 ImAnnotateController (class in GPy.util.latent_space_visualizations.controllers.imshow_controller), 97
 imread() (in module GPy.util.netpbmfile), 108
 imsave() (in module GPy.util.netpbmfile), 108
 ImshowController (class in GPy.util.latent_space_visualizations.controllers.imshow_controller), 97

- independent_outputs() (in module GPy.kern.constructors), 49
- IndependentOutputs (class in GPy.kern.parts.independent_outputs), 35
- index_to_slices() (in module GPy.kern.parts.independent_outputs), 35
- index_to_slices() (in module GPy.kern.parts.ODE_UY), 30
- initialise_latent() (in module GPy.models_modules.gplvm), 83
- initialize() (GPy.core.transformations.exponent method), 17
- initialize() (GPy.core.transformations.logexp method), 18
- initialize() (GPy.core.transformations.logexp_clipped method), 18
- initialize() (GPy.core.transformations.logistic method), 18
- initialize() (GPy.core.transformations.negative_exponent method), 18
- initialize() (GPy.core.transformations.negative_logexp method), 18
- initialize() (GPy.core.transformations.square method), 19
- initialize() (GPy.core.transformations.transformation method), 19
- initialize_axes() (GPy.util.visualize.mocap_data_show method), 113
- initialize_axes_modify() (GPy.util.visualize.mocap_data_show method), 113
- input_dim (GPy.testing.psi_stat_expectation_tests.Test attribute), 93
- input_dim (GPy.testing.psi_stat_gradient_tests.DPsiStatTest attribute), 93
- input_sensitivity() (GPy.core.model.Model method), 9
- inv_std_norm_cdf() (in module GPy.util.univariate_Gaussian), 111
- inverse_gamma (class in GPy.core.priors), 14
- is_positive_definite() (GPy.kern.kern.Kern_check_model method), 53
- isomap_faces() (in module GPy.util.datasets), 99
- ## J
- jacobian() (GPy.models_modules.gplvm.GPLVM method), 83
- jitchol() (in module GPy.util.linalg), 103
- jitchol_old() (in module GPy.util.linalg), 103
- ## K
- K() (GPy.kern.kern.kern method), 53
- K() (GPy.kern.parts.bias.Bias method), 30
- K() (GPy.kern.parts.Brownian.Brownian method), 27
- K() (GPy.kern.parts.coreionalize.Coreionalize method), 31
- K() (GPy.kern.parts.eq_ode1.Eq_ode1 method), 32
- K() (GPy.kern.parts.exponential.Exponential method), 32
- K() (GPy.kern.parts.finite_dimensional.FiniteDimensional method), 33
- K() (GPy.kern.parts.fixed.Fixed method), 33
- K() (GPy.kern.parts.gibbs.Gibbs method), 33
- K() (GPy.kern.parts.hetero.Hetero method), 34
- K() (GPy.kern.parts.hierarchical.Hierarchical method), 35
- K() (GPy.kern.parts.independent_outputs.IndependentOutputs method), 35
- K() (GPy.kern.parts.kernpart.Kernpart method), 35
- K() (GPy.kern.parts.linear.Linear method), 36
- K() (GPy.kern.parts.Matern32.Matern32 method), 27
- K() (GPy.kern.parts.Matern52.Matern52 method), 28
- K() (GPy.kern.parts.mlp.MLP method), 37
- K() (GPy.kern.parts.ODE_1.ODE_1 method), 29
- K() (GPy.kern.parts.ODE_UY.ODE_UY method), 29
- K() (GPy.kern.parts.periodic_exponential.PeriodicExponential method), 39
- K() (GPy.kern.parts.periodic_Matern32.PeriodicMatern32 method), 38
- K() (GPy.kern.parts.periodic_Matern52.PeriodicMatern52 method), 39
- K() (GPy.kern.parts.poly.POLY method), 40
- K() (GPy.kern.parts.prod.Prod method), 40
- K() (GPy.kern.parts.prod_orthogonal.prod_orthogonal method), 41
- K() (GPy.kern.parts.rational_quadratic.RationalQuadratic method), 41
- K() (GPy.kern.parts.rbf.RBF method), 42
- K() (GPy.kern.parts.rbfcos.RBFCos method), 43
- K() (GPy.kern.parts.spline.Spline method), 44
- K() (GPy.kern.parts.symmetric.Symmetric method), 44
- K() (GPy.kern.parts.sympykern.spkern method), 45
- K() (GPy.kern.parts.white.White method), 45
- K1() (GPy.kern.parts.prod.Prod method), 40
- K2() (GPy.kern.parts.prod.Prod method), 40
- Kdiag() (GPy.kern.kern.kern method), 53
- Kdiag() (GPy.kern.parts.bias.Bias method), 30
- Kdiag() (GPy.kern.parts.Brownian.Brownian method), 27
- Kdiag() (GPy.kern.parts.coreionalize.Coreionalize method), 31
- Kdiag() (GPy.kern.parts.eq_ode1.Eq_ode1 method), 32
- Kdiag() (GPy.kern.parts.exponential.Exponential method), 32
- Kdiag() (GPy.kern.parts.finite_dimensional.FiniteDimensional method), 33
- Kdiag() (GPy.kern.parts.gibbs.Gibbs method), 34
- Kdiag() (GPy.kern.parts.hetero.Hetero method), 34
- Kdiag() (GPy.kern.parts.hierarchical.Hierarchical method), 35
- Kdiag() (GPy.kern.parts.independent_outputs.IndependentOutputs method), 35
- Kdiag() (GPy.kern.parts.kernpart.Kernpart method), 35
- Kdiag() (GPy.kern.parts.linear.Linear method), 36
- Kdiag() (GPy.kern.parts.Matern32.Matern32 method), 28

Kdiag() (GPpy.kern.parts.Matern52.Matern52 method), 28
Kdiag() (GPpy.kern.parts.mlp.MLP method), 37
Kdiag() (GPpy.kern.parts.ODE_1.ODE_1 method), 29
Kdiag() (GPpy.kern.parts.ODE_UY.ODE_UY method), 30
Kdiag() (GPpy.kern.parts.periodic_exponential.PeriodicExponential method), 39
Kdiag() (GPpy.kern.parts.periodic_Matern32.PeriodicMatern32 method), 38
Kdiag() (GPpy.kern.parts.periodic_Matern52.PeriodicMatern52 method), 39
Kdiag() (GPpy.kern.parts.poly.POLY method), 40
Kdiag() (GPpy.kern.parts.prod.Prod method), 40
Kdiag() (GPpy.kern.parts.prod_orthogonal.prod_orthogonal method), 41
Kdiag() (GPpy.kern.parts.rational_quadratic.RationalQuadratic method), 41
Kdiag() (GPpy.kern.parts.rbf.RBF method), 42
Kdiag() (GPpy.kern.parts.rbfcos.RBFCos method), 43
Kdiag() (GPpy.kern.parts.spline.Spline method), 44
Kdiag() (GPpy.kern.parts.symmetric.Symmetric method), 44
Kdiag() (GPpy.kern.parts.sympykern.spkern method), 45
Kdiag() (GPpy.kern.parts.white.White method), 45
kern (class in GPpy.kern.kern), 53
Kern_check_dK_dtheta (class in GPpy.kern.kern), 53
Kern_check_dK_dX (class in GPpy.kern.kern), 53
Kern_check_dKdiag_dtheta (class in GPpy.kern.kern), 53
Kern_check_dKdiag_dX (class in GPpy.kern.kern), 53
Kern_check_model (class in GPpy.kern.kern), 53
kern_test() (in module GPpy.kern.kern), 55
Kernel (class in GPpy.mappings.kernel), 77
kernels (GPpy.testing.psi_stat_gradient_tests.DPSiStatTest attribute), 93
KernelTests (class in GPpy.testing.kernel_tests), 90
Kernpart (class in GPpy.kern.parts.kernpart), 35
Kernpart_inner (class in GPpy.kern.parts.kernpart), 36
Kernpart_stationary (class in GPpy.kern.parts.kernpart), 36
KL_divergence() (GPpy.models_modules.bayesian_gplvm.BayesianGPLVM method), 78
kmm_init() (in module GPpy.util.misc), 105

L

Laplace (class in GPpy.likelihoods.laplace), 74
Laplace_covariance() (GPpy.core.model.Model method), 8
Laplace_evidence() (GPpy.core.model.Model method), 8
LaplaceTests (class in GPpy.testing.likelihood_tests), 91
latent_cost() (in module GPpy.models_modules.bayesian_gplvm), 80
latent_cost_and_grad() (in module GPpy.models_modules.bayesian_gplvm), 80
latent_grad() (in module GPpy.models_modules.bayesian_gplvm), 80
likelihood (class in GPpy.likelihoods.likelihood), 75
likelihood_list (GPpy.models_modules.mrd.MRD attribute), 84
latent_changed() (GPpy.util.latent_space_visualizations.controllers.axis_event method), 96
Linear (class in GPpy.kern.parts.linear), 36
Linear (class in GPpy.mappings.linear), 77
linear() (in module GPpy.kern.constructors), 49
linear_grid() (in module GPpy.util.misc), 105
ln_diff_erf (class in GPpy.util.symbolic), 111
ln_diff_erf() (in module GPpy.kern.parts.sympy_helpers), 45
ln_diff_erfs() (in module GPpy.util.ln_diff_erfs), 105
lnpdf() (GPpy.core.priors.Gamma method), 12
lnpdf() (GPpy.core.priors.Gaussian method), 13
lnpdf() (GPpy.core.priors.inverse_gamma method), 14
lnpdf() (GPpy.core.priors.LogGaussian method), 13
lnpdf() (GPpy.core.priors.MultivariateGaussian method), 13
lnpdf_grad() (GPpy.core.priors.Gamma method), 12
lnpdf_grad() (GPpy.core.priors.Gaussian method), 13
lnpdf_grad() (GPpy.core.priors.inverse_gamma method), 14
lnpdf_grad() (GPpy.core.priors.LogGaussian method), 13
lnpdf_grad() (GPpy.core.priors.MultivariateGaussian method), 13
load_batch() (GPpy.core.svgp.SVIGP method), 17
load_channels() (GPpy.util.mocap.acclaim_skeleton method), 106
load_skel() (GPpy.util.mocap.acclaim_skeleton method), 106
load_text_data() (in module GPpy.util.mocap), 106
Log (class in GPpy.likelihoods.noise_models.gp_transformations), 64
Log_ex_1 (class in GPpy.likelihoods.noise_models.gp_transformations), 64
log_likelihood() (GPpy.core.fitc.FITC method), 4
log_likelihood() (GPpy.core.gp.GP method), 4
log_likelihood() (GPpy.core.mapping.Mapping_check_model method), 8
log_likelihood() (GPpy.core.model.Model method), 9
log_likelihood() (GPpy.core.sparse_gp.SparseGP method), 14
log_likelihood() (GPpy.core.svgp.SVIGP method), 17
log_likelihood() (GPpy.kern.kern.Kern_check_dKdiag_dtheta method), 53
log_likelihood() (GPpy.kern.kern.Kern_check_dKdiag_dX method), 53
log_likelihood() (GPpy.kern.kern.Kern_check_model method), 53
log_likelihood() (GPpy.models_modules.bayesian_gplvm.BayesianGPLVM method), 79

[log_likelihood\(\) \(GPy.models_modules.bayesian_gplvm.BayesianGPLVMWithMissingData method\), 79](#)
[log_likelihood\(\) \(GPy.models_modules.gradient_checker.GradientChecker method\), 83](#)
[log_likelihood\(\) \(GPy.models_modules.mrd.MRD method\), 84](#)
[log_likelihood\(\) \(GPy.models_modules.sparse_gplvm.SparseGPLVM method\), 88](#)
[log_likelihood\(\) \(GPy.models_modules.warped_gp.WarpedGP method\), 88](#)
[log_likelihood\(\) \(GPy.testing.psi_stat_gradient_tests.PsiStatModel method\), 94](#)
[log_max_bound \(GPy.core.transformations.logexp_clipped attribute\), 18](#)
[log_min_bound \(GPy.core.transformations.logexp_clipped attribute\), 18](#)
[log_predictive_density\(\) \(GPy.core.gp_base.GPBase method\), 5](#)
[log_predictive_density\(\) \(GPy.likelihoods.ep.EP method\), 72](#)
[log_predictive_density\(\) \(GPy.likelihoods.gaussian.Gaussian method\), 73](#)
[log_predictive_density\(\) \(GPy.likelihoods.laplace.Laplace method\), 74](#)
[log_predictive_density\(\) \(GPy.likelihoods.likelihood.likelihood method\), 75](#)
[log_predictive_density\(\) \(GPy.likelihoods.noise_models.noise_distributions.NoiseDistribution method\), 66](#)
[log_prior\(\) \(GPy.core.model.Model method\), 9](#)
[logexp \(class in GPy.core.transformations\), 17](#)
[logexp_clipped \(class in GPy.core.transformations\), 18](#)
[LogGaussian \(class in GPy.core.priors\), 13](#)
[logistic \(class in GPy.core.transformations\), 18](#)
[logpdf\(\) \(GPy.likelihoods.noise_models.noise_distributions.NoiseDistribution method\), 66](#)
[logpdf_link\(\) \(GPy.likelihoods.noise_models.bernoulli_noise.Bernoulli method\), 56](#)
[logpdf_link\(\) \(GPy.likelihoods.noise_models.exponential_noise.Exponential method\), 58](#)
[logpdf_link\(\) \(GPy.likelihoods.noise_models.gamma_noise.Gamma method\), 60](#)
[logpdf_link\(\) \(GPy.likelihoods.noise_models.gaussian_noise.Gaussian method\), 63](#)
[logpdf_link\(\) \(GPy.likelihoods.noise_models.noise_distributions.NoiseDistribution method\), 67](#)
[logpdf_link\(\) \(GPy.likelihoods.noise_models.poisson_noise.Poisson method\), 68](#)
[logpdf_link\(\) \(GPy.likelihoods.noise_models.student_t_noise.StudentT method\), 71](#)
[lvm \(class in GPy.util.visualize\), 113](#)
[lvm_dimselect \(class in GPy.util.visualize\), 113](#)
[lvm_subplots \(class in GPy.util.visualize\), 113](#)
[magnification\(\) \(GPy.models_modules.gplvm.GPLVM method\), 83](#)
[Mapping \(class in GPy.core.mapping\), 7](#)
[Mapping_check_df_dtheta \(class in GPy.core.mapping\), 8](#)
[Mapping_check_df_dX \(class in GPy.core.mapping\), 8](#)
[Mapping_check_model \(class in GPy.core.mapping\), 8](#)
[MappingTests \(class in GPy.testing.mapping_tests\), 92](#)
[Matern32 \(class in GPy.kern.parts.Matern32\), 27](#)
[Matern32\(\) \(in module GPy.kern.constructors\), 46](#)
[Matern52 \(class in GPy.kern.parts.Matern52\), 28](#)
[Matern52\(\) \(in module GPy.kern.constructors\), 46](#)
[matplotlib_show \(class in GPy.util.visualize\), 113](#)
[max_bound \(GPy.core.transformations.logexp_clipped attribute\), 18](#)
[mdot\(\) \(in module GPy.util.linalg\), 103](#)
[Metropolis_Hastings \(class in GPy.inference.samplers\), 25](#)
[min_bound \(GPy.core.transformations.logexp_clipped attribute\), 18](#)
[MLP \(class in GPy.kern.parts.mlp\), 37](#)
[MLP \(class in GPy.mappings.mlp\), 77](#)
[mlp\(\) \(in module GPy.kern.constructors\), 49](#)
[mocap_data_show \(class in GPy.util.visualize\), 113](#)
[mocap_data_show_vpython \(class in GPy.util.visualize\), 113](#)
[Model \(class in GPy.core.model\), 8](#)
[model_checkgrads\(\) \(in module GPy.testing.examples_tests\), 89](#)
[model_instance\(\) \(in module GPy.testing.examples_tests\), 90](#)
[model_interaction\(\) \(in module GPy.examples.tutorials\), 113](#)
[modify\(\) \(GPy.util.visualize.data_show method\), 112](#)
[modify\(\) \(GPy.util.visualize.image_show method\), 112](#)
[modify\(\) \(GPy.util.visualize.lvm method\), 113](#)
[modify\(\) \(GPy.util.visualize.mocap_data_show method\), 113](#)
[modify\(\) \(GPy.util.visualize.mocap_data_show_vpython method\), 114](#)
[modify\(\) \(GPy.util.visualize.vector_show method\), 114](#)
[modify_edges\(\) \(GPy.util.visualize.mocap_data_show_vpython method\), 114](#)
[modify_vertices\(\) \(GPy.util.visualize.mocap_data_show_vpython method\), 114](#)
[most_significant_input_dimensions\(\) \(in module GPy.util.plot_latent\), 109](#)
[MRD \(class in GPy.models_modules.mrd\), 83](#)
[mrd_simulation\(\) \(in module GPy.examples.dimensionality_reduction\), 20](#)
[MRDTests \(class in GPy.testing.mrd_tests\), 92](#)

multioutput_regression_1D()
(GPy.testing.unit_tests.GradientTests method),
94

multioutput_sparse_regression_1D()
(GPy.testing.unit_tests.GradientTests method),
94

multiple_optima() (in module GPy.examples.regression),
21

multiple_pdiv() (in module GPy.util.linalg), 103

multiply() (in module GPy.util.diag), 101

MultivariateGaussian (class in GPy.core.priors), 13

N

N (GPy.testing.psi_stat_expectation_tests.Test attribute),
93

N (GPy.testing.psi_stat_gradient_tests.DPsiStatTest attribute), 93

nargs (GPy.util.symbolic.dh_dd_i attribute), 110

nargs (GPy.util.symbolic.dh_dd_j attribute), 111

nargs (GPy.util.symbolic.dh_dl attribute), 111

nargs (GPy.util.symbolic.dh_dt attribute), 111

nargs (GPy.util.symbolic.dh_dtprime attribute), 111

nargs (GPy.util.symbolic.erfc attribute), 111

nargs (GPy.util.symbolic.erfcx attribute), 111

nargs (GPy.util.symbolic.h attribute), 111

nargs (GPy.util.symbolic.ln_diff_erf attribute), 111

negative_exponent (class in GPy.core.transformations),
18

negative_logexp (class in GPy.core.transformations), 18

NetpbmFile (class in GPy.util.netpbmfile), 109

new_chain() (GPy.inference.samplers.Metropolis_Hastings method), 25

nextDark() (in module GPy.util.Tango), 97

nextLight() (in module GPy.util.Tango), 98

nextMedium() (in module GPy.util.Tango), 98

NoiseDistribution (class in GPy.likelihoods.noise_models.noise_distributions),
65

non_null_samples() (GPy.inference.sgd.opt_SGD method), 26

Nsamples (GPy.testing.psi_stat_expectation_tests.Test attribute), 93

num_inducing (GPy.testing.psi_stat_expectation_tests.Test attribute), 93

num_inducing (GPy.testing.psi_stat_gradient_tests.DPsiStat attribute), 93

num_params_transformed()
(GPy.core.parameterized.Parameterized method), 12

objective_and_gradients() (GPy.core.model.Model method), 9

objective_function() (GPy.core.model.Model method), 9

objective_function_gradients() (GPy.core.model.Model method), 9

ode1_eq() (in module GPy.kern.constructors), 50

ODE_1 (class in GPy.kern.parts.ODE_1), 29

ODE_1() (in module GPy.kern.constructors), 47

ODE_UY (class in GPy.kern.parts.ODE_UY), 29

ODE_UY() (in module GPy.kern.constructors), 47

oil() (in module GPy.examples.classification), 19

oil() (in module GPy.util.datasets), 99

oil_100() (in module GPy.util.datasets), 99

olivetti_faces() (in module GPy.examples.dimensionality_reduction),
20

olivetti_faces() (in module GPy.util.datasets), 99

olympic_100m_men() (in module GPy.examples.regression), 21

olympic_100m_men() (in module GPy.util.datasets), 99

olympic_100m_women() (in module GPy.util.datasets),
99

olympic_200m_men() (in module GPy.util.datasets), 99

olympic_200m_women() (in module GPy.util.datasets),
99

olympic_400m_men() (in module GPy.util.datasets), 99

olympic_400m_women() (in module GPy.util.datasets),
99

olympic_marathon_men() (in module GPy.examples.regression), 21

olympic_marathon_men() (in module GPy.util.datasets),
99

olympic_sprints() (in module GPy.util.datasets), 99

on_click() (GPy.util.visualize.lvm method), 113

on_click() (GPy.util.visualize.lvm_dimselect method),
113

on_enter() (GPy.util.visualize.lvm method), 113

on_leave() (GPy.util.visualize.lvm method), 113

on_leave() (GPy.util.visualize.lvm_dimselect method),
113

on_move() (GPy.util.visualize.lvm method), 113

opt() (GPy.inference.conjugate_gradient_descent.Async_Optimize method), 23

opt() (GPy.inference.conjugate_gradient_descent.CGD method), 23

opt() (GPy.inference.optimization.opt_lbfgsb method), 25

opt() (GPy.inference.optimization.opt_rasm method), 25

opt() (GPy.inference.optimization.opt_SCG method), 25

opt() (GPy.inference.optimization.opt_simplex method),
25

opt() (GPy.inference.optimization.opt_tnc method), 25

opt() (GPy.inference.optimization.Optimizer method), 25

opt() (GPy.inference.sgd.opt_SGD method), 26

opt_async() (GPy.inference.conjugate_gradient_descent.Async_Optimize method), 23

opt_async() (GPy.inference.conjugate_gradient_descent.CGD method), 23

- [opt_lbfgsb](#) (class in `GPy.inference.optimization`), 25
[opt_name](#) (`GPy.inference.conjugate_gradient_descent.CGD` attribute), 24
[opt_rasm](#) (class in `GPy.inference.optimization`), 25
[opt_SCG](#) (class in `GPy.inference.optimization`), 25
[opt_SGD](#) (class in `GPy.inference.sgd`), 26
[opt_simplex](#) (class in `GPy.inference.optimization`), 25
[opt_tnc](#) (class in `GPy.inference.optimization`), 25
[opt_wrapper\(\)](#) (in module `GPy.util.misc`), 105
[optimize\(\)](#) (`GPy.core.model.Model` method), 10
[optimize\(\)](#) (`GPy.core.svgp.SVIGP` method), 17
[optimize_restarts\(\)](#) (`GPy.core.model.Model` method), 10
[optimize_SGD\(\)](#) (`GPy.core.model.Model` method), 10
[Optimizer](#) (class in `GPy.inference.optimization`), 24
[order_vertices\(\)](#) (`GPy.util.mocap.tree` method), 107
[osu_run1\(\)](#) (in module `GPy.util.datasets`), 99
- ## P
- [Parameterized](#) (class in `GPy.core.parameterized`), 11
[parse_text\(\)](#) (in module `GPy.util.mocap`), 106
[pca\(\)](#) (in module `GPy.util.linalg`), 103
[pddet\(\)](#) (in module `GPy.util.linalg`), 104
[pdf\(\)](#) (`GPy.core.priors.MultivariateGaussian` method), 13
[pdf\(\)](#) (`GPy.core.priors.Prior` method), 13
[pdf\(\)](#) (`GPy.likelihoods.noise_models.noise_distributions.NoiseDistribution` method), 67
[pdf_link\(\)](#) (`GPy.likelihoods.noise_models.bernoulli_noise.Bernoulli` method), 57
[pdf_link\(\)](#) (`GPy.likelihoods.noise_models.exponential_noise.Exponential` method), 58
[pdf_link\(\)](#) (`GPy.likelihoods.noise_models.gamma_noise.Gamma` method), 60
[pdf_link\(\)](#) (`GPy.likelihoods.noise_models.gaussian_noise.Gaussian` method), 63
[pdf_link\(\)](#) (`GPy.likelihoods.noise_models.noise_distributions.NoiseDistribution` method), 67
[pdf_link\(\)](#) (`GPy.likelihoods.noise_models.poisson_noise.Poisson` method), 69
[pdf_link\(\)](#) (`GPy.likelihoods.noise_models.student_t_noise.StudentT` method), 71
[pdinv\(\)](#) (in module `GPy.util.linalg`), 104
[periodic_exponential\(\)](#) (in module `GPy.kern.constructors`), 50
[periodic_Matern32\(\)](#) (in module `GPy.kern.constructors`), 50
[periodic_Matern52\(\)](#) (in module `GPy.kern.constructors`), 50
[PeriodicExponential](#) (class in `GPy.kern.parts.periodic_exponential`), 39
[PeriodicMatern32](#) (class in `GPy.kern.parts.periodic_Matern32`), 38
[PeriodicMatern52](#) (class in `GPy.kern.parts.periodic_Matern52`), 38
[pickle\(\)](#) (`GPy.core.parameterized.Parameterized` method), 12
[plot\(\)](#) (`GPy.core.gp_base.GPBase` method), 5
[plot\(\)](#) (`GPy.core.mapping.Mapping` method), 7
[plot\(\)](#) (`GPy.core.priors.MultivariateGaussian` method), 13
[plot\(\)](#) (`GPy.core.priors.Prior` method), 13
[plot\(\)](#) (`GPy.core.sparse_gp.SparseGP` method), 14
[plot\(\)](#) (`GPy.core.svgp.SVIGP` method), 17
[plot\(\)](#) (`GPy.inference.optimization.Optimizer` method), 25
[plot\(\)](#) (`GPy.kern.kern.kern` method), 54
[plot\(\)](#) (`GPy.models_modules.gplvm.GPLVM` method), 83
[plot\(\)](#) (`GPy.models_modules.sparse_gplvm.SparseGPLVM` method), 88
[plot\(\)](#) (`GPy.util.warping_functions.WarpingFunction` method), 115
[plot_ARD\(\)](#) (`GPy.kern.kern.kern` method), 54
[plot_f\(\)](#) (`GPy.core.gp_base.GPBase` method), 6
[plot_f\(\)](#) (`GPy.core.sparse_gp.SparseGP` method), 15
[plot_latent\(\)](#) (`GPy.models_modules.bayesian_gplvm.BayesianGPLVM` method), 79
[plot_latent\(\)](#) (`GPy.models_modules.gplvm.GPLVM` method), 83
[plot_latent\(\)](#) (`GPy.models_modules.mrd.MRD` method), 84
[plot_latent\(\)](#) (`GPy.models_modules.sparse_gplvm.SparseGPLVM` method), 88
[plot_latent\(\)](#) (in module `GPy.util.plot_latent`), 109
[plot_magnification\(\)](#) (`GPy.models_modules.gplvm.GPLVM` method), 83
[plot_magnification\(\)](#) (in module `GPy.util.plot_latent`), 109
[plot_predict\(\)](#) (`GPy.models_modules.mrd.MRD` method), 84
[plot_scalars\(\)](#) (`GPy.models_modules.mrd.MRD` method), 84
[plot_steepest_gradient_map\(\)](#) (`GPy.models_modules.bayesian_gplvm.BayesianGPLVM` method), 79
[plot_traces\(\)](#) (`GPy.core.svgp.SVIGP` method), 17
[plot_traces\(\)](#) (`GPy.inference.sgd.opt_SGD` method), 26
[plot_warping\(\)](#) (`GPy.models_modules.warped_gp.WarpedGP` method), 88
[plot_X\(\)](#) (`GPy.models_modules.mrd.MRD` method), 84
[plot_X_1d\(\)](#) (`GPy.models_modules.bayesian_gplvm.BayesianGPLVM` method), 79
[plot_X_1d\(\)](#) (`GPy.models_modules.mrd.MRD` method), 84
[Poisson](#) (class in `GPy.likelihoods.noise_models.poisson_noise`), 67
[poisson\(\)](#) (in module `GPy.likelihoods.noise_model_constructors`), 76
[PolakRibiere](#) (class in `GPy.inference.gradient_descent_update_rules`),

- 24
- POLY (class in GPy.kern.parts.poly), 40
- poly() (in module GPy.kern.constructors), 50
- pos_axis() (GPy.util.visualize.mocap_data_show_vpython method), 114
- posterior_samples() (GPy.core.gp_base.GPBase method), 6
- posterior_samples_f() (GPy.core.gp_base.GPBase method), 7
- ppca() (in module GPy.util.linalg), 104
- ppca_missing_data_at_random() (in module GPy.util.linalg), 104
- predict() (GPy.core.gp.GP method), 4
- predict() (GPy.core.sparse_gp.SparseGP method), 16
- predict() (GPy.core.svignp.SVIGNP method), 17
- predict() (GPy.inference.samplers.Metropolis_Hastings method), 25
- predict() (GPy.models_modules.warped_gp.WarpedGP method), 88
- predict_single_output() (GPy.core.gp.GP method), 5
- predictive_values() (GPy.likelihoods.ep.EP method), 73
- predictive_values() (GPy.likelihoods.ep_mixed_noise.EP_Mixed_Noise method), 73
- predictive_values() (GPy.likelihoods.gaussian.Gaussian method), 73
- predictive_values() (GPy.likelihoods.gaussian_mixed_noise.Gaussian_Mixed_Noise method), 74
- predictive_values() (GPy.likelihoods.laplace.Laplace method), 75
- predictive_values() (GPy.likelihoods.likelihood.likelihood method), 75
- predictive_values() (GPy.likelihoods.noise_models.noise_distributions.NoiseDistribution method), 67
- print_out() (in module GPy.inference.scg), 26
- Prior (class in GPy.core.priors), 13
- PriorTests (class in GPy.testing.prior_tests), 92
- Probit (class in GPy.likelihoods.noise_models.gp_transformations), 64
- process_values() (GPy.util.visualize.mocap_data_show method), 113
- process_values() (GPy.util.visualize.mocap_data_show_vpython method), 114
- process_values() (GPy.util.visualize.skeleton_show method), 114
- process_values() (GPy.util.visualize.stick_show method), 114
- Prod (class in GPy.kern.parts.prod), 40
- prod() (GPy.kern.kern.kern method), 55
- prod() (in module GPy.kern.constructors), 51
- prod_orthogonal (class in GPy.kern.parts.prod_orthogonal), 41
- prompt_user() (in module GPy.util.datasets), 99
- propagate_param() (GPy.models_modules.mrd.MRD method), 84
- pseudo_EM() (GPy.core.model.Model method), 10
- psi0() (GPy.kern.kern.kern method), 55
- psi0() (GPy.kern.parts.bias.Bias method), 30
- psi0() (GPy.kern.parts.kernpart.Kernpart method), 36
- psi0() (GPy.kern.parts.linear.Linear method), 37
- psi0() (GPy.kern.parts.rbf.RBF method), 42
- psi0() (GPy.kern.parts.white.White method), 46
- psi1() (GPy.kern.kern.kern method), 55
- psi1() (GPy.kern.parts.bias.Bias method), 30
- psi1() (GPy.kern.parts.kernpart.Kernpart method), 36
- psi1() (GPy.kern.parts.linear.Linear method), 37
- psi1() (GPy.kern.parts.rbf.RBF method), 43
- psi1() (GPy.kern.parts.white.White method), 46
- psi2() (GPy.kern.kern.kern method), 55
- psi2() (GPy.kern.parts.bias.Bias method), 30
- psi2() (GPy.kern.parts.kernpart.Kernpart method), 36
- psi2() (GPy.kern.parts.linear.Linear method), 37
- psi2() (GPy.kern.parts.rbf.RBF method), 43
- psi2() (GPy.kern.parts.white.White method), 46
- psi2_new() (GPy.kern.parts.linear.Linear method), 37
- PsiStatModel (class in GPy.testing.psi_stat_gradient_tests), 93
- pumadyn() (in module GPy.util.datasets), 100
- ## R
- randomize() (GPy.core.model.Model method), 10
- randomize() (GPy.mappings.kernel.Kernel method), 77
- randomize() (GPy.mappings.linear.Linear method), 77
- randomize() (GPy.mappings.mlp.MLP method), 78
- randomize() (GPy.models_modules.mrd.MRD method), 84
- randomize() (GPy.likelihoods.laplace.Laplace method), 75
- rational_quadratic() (in module GPy.kern.constructors), 51
- RationalQuadratic (class in GPy.kern.parts.rational_quadratic), 41
- RBF (class in GPy.kern.parts.rbf), 42
- rbf() (in module GPy.kern.constructors), 51
- rbf_inv() (in module GPy.kern.constructors), 51
- rbf_symple() (in module GPy.kern.constructors), 52
- RBFCos (class in GPy.kern.parts.rbfcos), 43
- rbfcos() (in module GPy.kern.constructors), 52
- RBFInv (class in GPy.kern.parts.rbf_inv), 43
- read() (in module GPy), 116
- read_bonedata() (GPy.util.mocap.acclaim_skeleton method), 106
- read_channels() (GPy.util.mocap.acclaim_skeleton method), 106
- read_connections() (in module GPy.util.mocap), 106
- read_documentation() (GPy.util.mocap.acclaim_skeleton method), 106
- read_hierarchy() (GPy.util.mocap.acclaim_skeleton method), 106

- read_line() (GPy.util.mocap.acclaim_skeleton method), 106
 read_root() (GPy.util.mocap.acclaim_skeleton method), 106
 read_skel() (GPy.util.mocap.acclaim_skeleton method), 106
 read_units() (GPy.util.mocap.acclaim_skeleton method), 106
 Reciprocal (class in GPy.likelihoods.noise_models.gp_transfusions module GPy.inference.scg), 25
 recompute_X() (GPy.util.latent_space_visualizations.controllers.axis_attribute_controller.BufferedAxisChangedController method), 96
 removeRightTicks() (in module GPy.util.plot), 109
 removeRightTicks() (in module GPy.util.Tango), 98
 removeUpperTicks() (in module GPy.util.plot), 109
 removeUpperTicks() (in module GPy.util.Tango), 98
 reporthook() (in module GPy.util.datasets), 100
 reset() (in module GPy.util.Tango), 98
 resolve_indices() (GPy.util.mocap.acclaim_skeleton method), 106
 restart() (GPy.likelihoods.ep.EP method), 73
 restart() (GPy.likelihoods.ep_mixed_noise.EP_Mixed_Noise method), 73
 restart() (GPy.likelihoods.laplace.Laplace method), 75
 restart() (GPy.likelihoods.likelihood.likelihood method), 75
 restore_constraints() (GPy.inference.sgd.opt_SGD method), 26
 ripley_synth() (in module GPy.util.datasets), 100
 robot_wireless() (in module GPy.examples.dimensionality_reduction), 20
 robot_wireless() (in module GPy.examples.regression), 21
 robot_wireless() (in module GPy.util.datasets), 100
 rotation_matrix() (in module GPy.util.mocap), 106
 run() (GPy.inference.optimization.Optimizer method), 25
 runsignal (GPy.inference.conjugate_gradient_descent.AsyncOptimize attribute), 23
 rvs() (GPy.core.priors.Gamma method), 12
 rvs() (GPy.core.priors.Gaussian method), 13
 rvs() (GPy.core.priors.inverse_gamma method), 14
 rvs() (GPy.core.priors.LogGaussian method), 13
 rvs() (GPy.core.priors.MultivariateGaussian method), 13
- ## S
- sample() (GPy.inference.samplers.Metropolis_Hastings method), 25
 sample_class() (in module GPy.util.datasets), 100
 samples() (GPy.likelihoods.noise_models.bernoulli_noise.Bernoulli method), 57
 samples() (GPy.likelihoods.noise_models.exponential_noise.Exponential method), 59
 samples() (GPy.likelihoods.noise_models.gaussian_noise.Gaussian method), 63
 samples() (GPy.likelihoods.noise_models.noise_distributions.NoiseDistributions method), 67
 samples() (GPy.likelihoods.noise_models.poisson_noise.Poisson method), 69
 samples() (GPy.likelihoods.noise_models.student_t_noise.StudentT method), 72
 SCG (in module GPy.inference.scg), 25
 SENTINEL (GPy.inference.conjugate_gradient_descent.AsyncOptimize attribute), 23
 set_data() (GPy.likelihoods.gaussian.Gaussian method), 74
 set_data() (GPy.likelihoods.gaussian_mixed_noise.Gaussian_Mixed_Noise method), 74
 set_image() (GPy.util.visualize.image_show method), 112
 set_prior() (GPy.core.model.Model method), 10
 set_rotation_matrices() (GPy.util.mocap.acclaim_skeleton method), 106
 set_vb_param() (GPy.core.svgp.SVIGP method), 17
 setDarkFigures() (in module GPy.util.Tango), 98
 setLightFigures() (in module GPy.util.Tango), 98
 setstate() (GPy.core.gp.GP method), 5
 setstate() (GPy.core.gp_base.GPBase method), 7
 setstate() (GPy.core.model.Model method), 11
 setstate() (GPy.core.parameterized.Parameterized method), 12
 setstate() (GPy.core.sparse_gp.SparseGP method), 16
 setstate() (GPy.core.svgp.SVIGP method), 17
 setstate() (GPy.kern.kern.kern method), 55
 setstate() (GPy.models_modules.bayesian_gplvm.BayesianGPLVM method), 79
 setstate() (GPy.models_modules.bayesian_gplvm.BayesianGPLVMWithMixtures method), 79
 setstate() (GPy.models_modules.gp_regression.GPRegression method), 82
 setstate() (GPy.models_modules.gplvm.GPLVM method), 83
 setstate() (GPy.models_modules.mrd.MRD method), 84
 setstate() (GPy.models_modules.sparse_gp_classification.SparseGPClassification method), 85
 setstate() (GPy.models_modules.sparse_gp_regression.SparseGPRegression method), 87
 setstate() (GPy.models_modules.sparse_gplvm.SparseGPLVM method), 88
 setstate() (GPy.models_modules.svgp_regression.SVIGPRegression method), 88
 setstate() (GPy.models_modules.warped_gp.WarpedGP method), 88
 setUp() (GPy.testing.gp_transformation_tests.TestTransformations method), 90
 setUp() (GPy.testing.likelihood_tests.LaplaceTests method), 91

- setUp() (GPy.testing.likelihood_tests.TestNoiseModels method), 91
 setUp() (GPy.testing.psi_stat_expectation_tests.Test method), 93
 setUp() (GPy.testing.unit_tests.GradientTests method), 94
 shift_constraints() (GPy.inference.sgd.opt_SGD method), 26
 show_sensitivities() (GPy.util.visualize.lvm method), 113
 sigmoid() (in module GPy.util.squashers), 110
 silence_errors() (in module GPy.util.decorators), 100
 silhouette() (in module GPy.examples.regression), 21
 silhouette() (in module GPy.util.datasets), 100
 simulation_BGPLVM() (in module GPy.util.datasets), 100
 single_softmax() (in module GPy.util.squashers), 110
 skeleton (class in GPy.util.mocap), 107
 skeleton_show (class in GPy.util.visualize), 114
 smooth_angle_channels() (GPy.util.mocap.skeleton method), 107
 softmax() (in module GPy.util.squashers), 110
 sparse_GP_regression_1D() (in module GPy.examples.regression), 22
 sparse_GP_regression_2D() (in module GPy.examples.regression), 22
 sparse_gplvm_oil() (in module GPy.examples.dimensionality_reduction), 21
 sparse_GPLVMTests (class in GPy.testing.sparse_gplvm_tests), 94
 sparse_toy_linear_1d_classification() (in module GPy.examples.classification), 19
 SparseGP (class in GPy.core.sparse_gp), 14
 SparseGPClassification (class in GPy.models_modules.sparse_gp_classification), 85
 SparseGPLVM (class in GPy.models_modules.sparse_gplvm), 87
 SparseGPMultioutputRegression (class in GPy.models_modules.sparse_gp_multioutput_regression), 86
 SparseGPRegression (class in GPy.models_modules.sparse_gp_regression), 87
 spkern (class in GPy.kern.parts.sympykern), 45
 Spline (class in GPy.kern.parts.spline), 44
 spline() (in module GPy.kern.constructors), 52
 square (class in GPy.core.transformations), 18
 std_norm_cdf() (in module GPy.util.univariate_Gaussian), 111
 std_norm_pdf() (in module GPy.util.univariate_Gaussian), 112
 step_with_missing_data() (GPy.inference.sgd.opt_SGD method), 26
 stick() (in module GPy.examples.dimensionality_reduction), 21
 stick_bgplvm() (in module GPy.examples.dimensionality_reduction), 21
 stick_play() (in module GPy.examples.dimensionality_reduction), 21
 stick_show (class in GPy.util.visualize), 114
 student_t() (in module GPy.likelihoods.noise_model_constructors), 76
 student_t_approx() (in module GPy.examples.non_gaussian), 21
 StudentT (class in GPy.likelihoods.noise_models.student_t_noise), 69
 subset_parameter_vector() (GPy.inference.sgd.opt_SGD method), 26
 subtract() (in module GPy.util.diag), 101
 summary() (GPy.core.priors.Gamma method), 12
 summary() (GPy.core.priors.MultivariateGaussian method), 13
 SVIGP (class in GPy.core.svigp), 16
 SVIGPRegression (class in GPy.models_modules.svigp_regression), 88
 swap_vertices() (GPy.util.mocap.tree method), 107
 swiss_roll() (in module GPy.examples.dimensionality_reduction), 21
 swiss_roll() (in module GPy.util.datasets), 100
 swiss_roll_1000() (in module GPy.util.datasets), 100
 swiss_roll_generated() (in module GPy.util.datasets), 100
 Symmetric (class in GPy.kern.parts.symmetric), 44
 symmetric() (in module GPy.kern.constructors), 52
 symmetrify() (in module GPy.util.linalg), 104
 symmetrify_murray() (in module GPy.util.linalg), 104
 sympykern() (in module GPy.kern.constructors), 52
- ## T
- t_d2logpdf2_df2_dparams() (GPy.testing.likelihood_tests.TestNoiseModels method), 91
 t_d2logpdf2_dlink2_dparams() (GPy.testing.likelihood_tests.TestNoiseModels method), 91
 t_d2logpdf_df2() (GPy.testing.likelihood_tests.TestNoiseModels method), 91
 t_d2logpdf_dlink2() (GPy.testing.likelihood_tests.TestNoiseModels method), 91
 t_d2transf_df2() (GPy.testing.gp_transformation_tests.TestTransformations method), 90
 t_d3logpdf_df3() (GPy.testing.likelihood_tests.TestNoiseModels method), 91

[t_d3logpdf_dlink3\(\)](#) (GPy.testing.likelihood_tests.TestNoiseModels method), 91
[t_d3transf_df3\(\)](#) (GPy.testing.gp_transformation_tests.TestTransformations method), 91
[t_dlogpdf_df\(\)](#) (GPy.testing.likelihood_tests.TestNoiseModels method), 95
[t_dlogpdf_df_dparams\(\)](#) (GPy.testing.likelihood_tests.TestNoiseModels method), 90
[t_dlogpdf_dlink\(\)](#) (GPy.testing.likelihood_tests.TestNoiseModels method), 95
[t_dlogpdf_dlink_dparams\(\)](#) (GPy.testing.likelihood_tests.TestNoiseModels method), 94
[t_dlogpdf_dparams\(\)](#) (GPy.testing.likelihood_tests.TestNoiseModels method), 94
[t_dlogpdf_link_dparams\(\)](#) (GPy.testing.likelihood_tests.TestNoiseModels method), 94
[t_dtransf_df\(\)](#) (GPy.testing.gp_transformation_tests.TestTransformations method), 90
[t_ep_fit_rbf_white\(\)](#) (GPy.testing.likelihood_tests.TestNoiseModels method), 94
[t_laplace_fit_rbf_white\(\)](#) (GPy.testing.likelihood_tests.TestNoiseModels method), 94
[t_logpdf\(\)](#) (GPy.testing.likelihood_tests.TestNoiseModels method), 94
[TanhWarpingFunction](#) (class in GPy.util.warping_functions), 114
[TanhWarpingFunction_d](#) (class in GPy.util.warping_functions), 115
[tdot\(\)](#) (in module GPy.util.linalg), 104
[tdot_blas\(\)](#) (in module GPy.util.linalg), 104
[tdot_numpy\(\)](#) (in module GPy.util.linalg), 104
[tearDown\(\)](#) (GPy.testing.gp_transformation_tests.TestTransformations method), 90
[tearDown\(\)](#) (GPy.testing.likelihood_tests.LaplaceTests method), 91
[tearDown\(\)](#) (GPy.testing.likelihood_tests.TestNoiseModels method), 91
[Test](#) (class in GPy.testing.cgd_tests), 89
[Test](#) (class in GPy.testing.psi_stat_expectation_tests), 93
[test_bias_kern\(\)](#) (GPy.testing.bgplvm_tests.BGPLVMTTests method), 89
[test_bias_kern\(\)](#) (GPy.testing.gplvm_tests.GPLVMTTests method), 90
[test_bias_kern\(\)](#) (GPy.testing.sparse_gplvm_tests.sparse_GPLVMTTests method), 94
[test_eq_sympykernel\(\)](#) (GPy.testing.kernel_tests.KernelTests method), 90
[test_fixedkernel\(\)](#) (GPy.testing.kernel_tests.KernelTests method), 90
[test_Gamma\(\)](#) (GPy.testing.prior_tests.PriorTests method), 92
[test_Gaussian_d2logpdf_df2_2\(\)](#) (GPy.testing.likelihood_tests.LaplaceTests method), 91
[test_generalized_FITC\(\)](#) (GPy.testing.unit_tests.GradientTests method), 95
[test_gibbskernel\(\)](#) (GPy.testing.kernel_tests.KernelTests method), 90
[test_GP_EP_probit\(\)](#) (GPy.testing.unit_tests.GradientTests method), 94
[test_GPLVM_rbf_bias_white_kern_2D\(\)](#) (GPy.testing.unit_tests.GradientTests method), 94
[test_GPLVM_rbf_linear_white_kern_2D\(\)](#) (GPy.testing.unit_tests.GradientTests method), 94
[test_GPRegression_bias_kern_1D\(\)](#) (GPy.testing.unit_tests.GradientTests method), 94
[test_GPRegression_bias_kern_2D\(\)](#) (GPy.testing.unit_tests.GradientTests method), 94
[test_GPRegression_exponential_1D\(\)](#) (GPy.testing.unit_tests.GradientTests method), 94
[test_GPRegression_exponential_2D\(\)](#) (GPy.testing.unit_tests.GradientTests method), 94
[test_GPRegression_exponential_ARD_2D\(\)](#) (GPy.testing.unit_tests.GradientTests method), 94
[test_GPRegression_linear_kern_1D\(\)](#) (GPy.testing.unit_tests.GradientTests method), 94
[test_GPRegression_linear_kern_1D_ARD\(\)](#) (GPy.testing.unit_tests.GradientTests method), 94
[test_GPRegression_linear_kern_2D\(\)](#) (GPy.testing.unit_tests.GradientTests method), 94
[test_GPRegression_linear_kern_2D_ARD\(\)](#) (GPy.testing.unit_tests.GradientTests method), 94
[test_GPRegression_matern32_1D\(\)](#) (GPy.testing.unit_tests.GradientTests method), 94
[test_GPRegression_matern32_2D\(\)](#) (GPy.testing.unit_tests.GradientTests method), 95
[test_GPRegression_matern32_ARD_2D\(\)](#) (GPy.testing.unit_tests.GradientTests method), 95
[test_GPRegression_matern52_1D\(\)](#) (GPy.testing.unit_tests.GradientTests method), 95

```

test_GPRegression_matern52_2D()
    (GPy.testing.unit_tests.GradientTests method),
    95
test_GPRegression_matern52_ARD_2D()
    (GPy.testing.unit_tests.GradientTests method),
    95
test_GPRegression_mlp_1d()
    (GPy.testing.unit_tests.GradientTests method),
    95
test_GPRegression_poly_1d()
    (GPy.testing.unit_tests.GradientTests method),
    95
test_GPRegression_rbf_1d()
    (GPy.testing.unit_tests.GradientTests method),
    95
test_GPRegression_rbf_2D()
    (GPy.testing.unit_tests.GradientTests method),
    95
test_GPRegression_rbf_ARD_2D()
    (GPy.testing.unit_tests.GradientTests method),
    95
test_gradients() (GPy.testing.mrd_tests.MRDTests
    method), 92
test_heterokernel() (GPy.testing.kernel_tests.KernelTests
    method), 90
test_incompatibility() (GPy.testing.prior_tests.PriorTests
    method), 92
test_kernel_backconstraint()
    (GPy.testing.bcgplvm_tests.BCGPLVMTests
    method), 89
test_kernelmapping() (GPy.testing.mapping_tests.MappingTests
    method), 92
test_kerneltie() (GPy.testing.kernel_tests.KernelTests
    method), 90
test_laplace_log_likelihood()
    (GPy.testing.likelihood_tests.LaplaceTests
    method), 91
test_linear_backconstraint()
    (GPy.testing.bcgplvm_tests.BCGPLVMTests
    method), 89
test_linear_bias_kern() (GPy.testing.bgplvm_tests.BGPLVMTests
    method), 89
test_linear_kern() (GPy.testing.bgplvm_tests.BGPLVMTests
    method), 89
test_linear_kern() (GPy.testing.gplvm_tests.GPLVMTests
    method), 90
test_linear_kern() (GPy.testing.sparse_gplvm_tests.sparse_GPLVMTests
    method), 94
test_linearkernel() (GPy.testing.kernel_tests.KernelTests
    method), 90
test_linearmapping() (GPy.testing.mapping_tests.MappingTests
    method), 92
test_lognormal() (GPy.testing.prior_tests.PriorTests
    method), 92
test_Matern32kernel() (GPy.testing.kernel_tests.KernelTests
    method), 90
test_Matern52kernel() (GPy.testing.kernel_tests.KernelTests
    method), 90
test_mlp_backconstraint()
    (GPy.testing.bcgplvm_tests.BCGPLVMTests
    method), 89
test_mlpkernel() (GPy.testing.kernel_tests.KernelTests
    method), 90
test_mplmapping() (GPy.testing.mapping_tests.MappingTests
    method), 92
test_models() (in module GPy.testing.examples_tests), 90
test_noise_models() (GPy.testing.likelihood_tests.TestNoiseModels
    method), 91
test_ode1_eqkernel() (GPy.testing.kernel_tests.KernelTests
    method), 90
test_periodic_exponentialkernel()
    (GPy.testing.kernel_tests.KernelTests method),
    91
test_periodic_Matern32kernel()
    (GPy.testing.kernel_tests.KernelTests method),
    90
test_periodic_Matern52kernel()
    (GPy.testing.kernel_tests.KernelTests method),
    90
test_polykernel() (GPy.testing.kernel_tests.KernelTests
    method), 91
test_psi0() (GPy.testing.psi_stat_expectation_tests.Test
    method), 93
test_psi1() (GPy.testing.psi_stat_expectation_tests.Test
    method), 93
test_psi2() (GPy.testing.psi_stat_expectation_tests.Test
    method), 93
test_rational_quadratickernel()
    (GPy.testing.kernel_tests.KernelTests method),
    91
test_rbf_bias_kern() (GPy.testing.bgplvm_tests.BGPLVMTests
    method), 89
test_rbf_invkernel() (GPy.testing.kernel_tests.KernelTests
    method), 91
test_rbf_kern() (GPy.testing.bgplvm_tests.BGPLVMTests
    method), 89
test_rbf_kern() (GPy.testing.gplvm_tests.GPLVMTests
    method), 90
test_rbf_kern() (GPy.testing.sparse_gplvm_tests.sparse_GPLVMTests
    method), 94
test_rbf_kern() (GPy.testing.bcgplvm_tests.BCGPLVMTests
    method), 89
test_rbf_sympykernel() (GPy.testing.kernel_tests.KernelTests
    method), 91
test_rbfkernel() (GPy.testing.kernel_tests.KernelTests
    method), 91
test_sparse_EP_DTC_probit()
    (GPy.testing.unit_tests.GradientTests method),

```


- 95
 test_SparseGPRegression_rbf_linear_white_kern_1D()
 (GPy.testing.unit_tests.GradientTests method), 95
 test_SparseGPRegression_rbf_linear_white_kern_1D_uncertain_inputs()
 (GPy.testing.unit_tests.GradientTests method), 95
 test_SparseGPRegression_rbf_linear_white_kern_2D()
 (GPy.testing.unit_tests.GradientTests method), 95
 test_SparseGPRegression_rbf_linear_white_kern_2D_uncertain_inputs()
 (GPy.testing.unit_tests.GradientTests method), 95
 test_SparseGPRegression_rbf_white_kern_1d()
 (GPy.testing.unit_tests.GradientTests method), 95
 test_SparseGPRegression_rbf_white_kern_2D()
 (GPy.testing.unit_tests.GradientTests method), 95
 test_transformations() (GPy.testing.gp_transformation_tests.TestTransformations method), 90
 testMinimizeSquare() (GPy.testing.cgd_tests.Test method), 89
 TestNoiseModels (class in GPy.testing.likelihood_tests), 91
 testPsi0() (GPy.testing.psi_stat_gradient_tests.DPsiStatTest method), 93
 testPsi1() (GPy.testing.psi_stat_gradient_tests.DPsiStatTest method), 93
 testPsi2_bia() (GPy.testing.psi_stat_gradient_tests.DPsiStatTest method), 93
 testPsi2_lin() (GPy.testing.psi_stat_gradient_tests.DPsiStatTest method), 93
 testPsi2_lin_bia() (GPy.testing.psi_stat_gradient_tests.DPsiStatTest method), 93
 testPsi2_rbf() (GPy.testing.psi_stat_gradient_tests.DPsiStatTest method), 93
 testPsi2_rbf_bia() (GPy.testing.psi_stat_gradient_tests.DPsiStatTest method), 93
 testRosen() (GPy.testing.cgd_tests.Test method), 89
 tests() (in module GPy), 116
 TestTransformations (class in GPy.testing.gp_transformation_tests), 90
 theta() (in module GPy.kern.parts.Brownian), 27
 theta() (in module GPy.kern.parts.spline), 44
 tie_params() (GPy.core.parameterized.Parameterized method), 12
 times() (in module GPy.util.diag), 101
 to_xyz() (GPy.util.mocap.acclaim_skeleton method), 106
 to_xyz() (GPy.util.mocap.skeleton method), 107
 toy_1d() (in module GPy.examples.stochastic), 22
 toy_ARD() (in module GPy.examples.regression), 22
 toy_ARD_sparse() (in module GPy.examples.regression), 22
 toy_heaviside() (in module GPy.examples.classification), 19
 toy_linear_1d_classification() (in module GPy.examples.classification), 20
 toy_linear_1d_classification_laplace() (in module GPy.util.datasets), 100
 toy_linear_1d_classification_laplace() (in module GPy.examples.classification), 20
 toy_poisson_rbf_1d_laplace() (in module GPy.examples.regression), 22
 toy_rbf_1d() (in module GPy.util.datasets), 100
 toy_rbf_1d_50() (in module GPy.examples.regression), 22
 toy_rbf_1d_50() (in module GPy.util.datasets), 100
 trace_dot() (in module GPy.util.linalg), 104
 transf() (GPy.likelihoods.noise_models.gp_transformations.GPTransformations method), 64
 transf() (GPy.likelihoods.noise_models.gp_transformations.Heaviside method), 64
 transf() (GPy.likelihoods.noise_models.gp_transformations.Identity method), 64
 transf() (GPy.likelihoods.noise_models.gp_transformations.Log method), 64
 transf() (GPy.likelihoods.noise_models.gp_transformations.Log_ex_1 method), 64
 transf() (GPy.likelihoods.noise_models.gp_transformations.Probit method), 65
 transf() (GPy.likelihoods.noise_models.gp_transformations.Reciprocal method), 65
 transform_data() (GPy.models_modules.warped_gp.WarpedGP method), 88
 transformation (class in GPy.core.transformations), 19
 StratTest (class in GPy.util.mocap), 107
 tuto_GP_regression() (in module GPy.examples.tutorials), 22
 tuto_kernel_overview() (in module GPy.examples.tutorials), 22
- ## U
- uncertain_inputs_sparse_regression() (in module GPy.examples.regression), 22
 unconstrain() (GPy.core.parameterized.Parameterized method), 12
 untie_everything() (GPy.core.parameterized.Parameterized method), 12
 update() (GPy.util.latent_space_visualizations.controllers.axis_event_controller method), 96
 update() (GPy.util.latent_space_visualizations.controllers.axis_event_controller method), 97
 update_likelihood_approximation() (GPy.core.fitc.FITC method), 4
 update_likelihood_approximation() (GPy.core.gp.GP method), 5

[update_likelihood_approximation\(\)](#)
 (GPy.core.sparse_gp.SparseGP method), [16](#)
[update_likelihood_approximation\(\)](#)
 (GPy.models_modules.mrd.MRD method), [84](#)
[update_view\(\)](#) (GPy.util.latent_space_visualizations.controllers.axis_event_controller.BufferedAxisChangedController method), [97](#)
[update_view\(\)](#) (GPy.util.latent_space_visualizations.controllers.imshow_controller.ImAnnotateController method), [97](#)
[update_view\(\)](#) (GPy.util.latent_space_visualizations.controllers.imshow_controller.ImshowController method), [97](#)

Y

[Y](#) (GPy.testing.psi_stat_gradient_tests.DPsiStatTest attribute), [93](#)
[ylim_changed\(\)](#) (GPy.util.latent_space_visualizations.controllers.axis_event_controller.BufferedAxisChangedController method), [96](#)
[ylim_changed\(\)](#) (GPy.util.latent_space_visualizations.controllers.axis_event_controller.ImAnnotateController method), [96](#)
[Z](#) (GPy.models_modules.mrd.MRD attribute), [84](#)
[Z](#) (GPy.testing.psi_stat_gradient_tests.DPsiStatTest attribute), [93](#)

V

[variance](#) (GPy.likelihoods.noise_models.student_t_noise.StudentT attribute), [72](#)
[vb_grad_natgrad\(\)](#) (GPy.core.svigp.SVIGP method), [17](#)
[vector_show](#) (class in GPy.util.visualize), [114](#)
[vertex](#) (class in GPy.util.mocap), [107](#)
[view\(\)](#) (in module GPy.util.diag), [101](#)
[vpython_show](#) (class in GPy.util.visualize), [114](#)

W

[WarpedGP](#) (class in GPy.models_modules.warped_gp), [88](#)
[warping_function_gradients\(\)](#)
 (GPy.models_modules.warped_gp.WarpedGP method), [89](#)
[WarpingFunction](#) (class in GPy.util.warping_functions), [115](#)
[weave_psi2\(\)](#) (GPy.kern.parts.rbf.RBF method), [43](#)
[weave_psi2\(\)](#) (GPy.kern.parts.rbf_inv.RBFInv method), [43](#)
[White](#) (class in GPy.kern.parts.white), [45](#)
[white\(\)](#) (in module GPy.kern.constructors), [52](#)
[wrap_around\(\)](#) (GPy.util.visualize.skeleton_show method), [114](#)
[write\(\)](#) (GPy.util.netpbmfile.NetpbmFile method), [109](#)

X

[X](#) (GPy.models_modules.mrd.MRD attribute), [84](#)
[X](#) (GPy.testing.psi_stat_gradient_tests.DPsiStatTest attribute), [93](#)
[x_frame1D\(\)](#) (in module GPy.util.plot), [109](#)
[x_frame2D\(\)](#) (in module GPy.util.plot), [109](#)
[X_var](#) (GPy.testing.psi_stat_gradient_tests.DPsiStatTest attribute), [93](#)
[X_variance](#) (GPy.models_modules.mrd.MRD attribute), [84](#)
[xlim_changed\(\)](#) (GPy.util.latent_space_visualizations.controllers.axis_event_controller.AxisChangedController method), [96](#)
[xlim_changed\(\)](#) (GPy.util.latent_space_visualizations.controllers.axis_event_controller.AxisEventController method), [96](#)
[xw_pen\(\)](#) (in module GPy.util.datasets), [100](#)