Package ‘gptk’

February 20, 2015

Version 1.08
Date 2014-03-07
Title Gaussian Processes Tool-Kit
Author Alfredo Kalaitzis <alkalait@gmail.com>,
      Antti Honkela <antti.honkela@tkk.fi>,
      Pei Gao <pg349@medschl.cam.ac.uk>,
      Neil D. Lawrence <N.Lawrence@dcs.shef.ac.uk>
Maintainer Alfredo Kalaitzis <alkalait@gmail.com>
Depends R (>= 2.8.0), Matrix, fields
Imports spam
Description The gptk package implements a general-purpose toolkit for Gaussian
       process regression with a variety of covariance functions (e.g. RBF, Mattern, polynomial, etc).
       Based on a MATLAB implementation by Neil D. Lawrence. See inst/doc/index.html for more details.
License BSD_2_clause + file LICENSE
URL
BugReports alkalait@gmail.com
NeedsCompilation no
Repository CRAN
Date/Publication 2014-03-07 17:44:33

R topics documented:

  basePlot ............................................................. 3
  cmpndKernParamInit .................................................. 3
  cmpndNoiseParamInit ............................................... 4
  demAutoOptimiseGp .................................................. 5
  demGpCov2D .......................................................... 5
  demGpSample ......................................................... 6
R topics documented:

demInterpolation ................................................. 7
demOptimiseGp .................................................. 7
demRegression .................................................... 8
exTransform ..................................................... 8
gaussianNoiseOut ............................................... 9
gaussianNoiseParamInit ....................................... 10
gaussSamp ........................................................ 11
gpBlockIndices ................................................. 11
gpComputeAlpha ............................................... 12
gpComputeM ....................................................... 13
gpCovGrads ...................................................... 13
gpCovGradsTest ................................................. 14
gpCreate .......................................................... 15
gpDataIndices ................................................... 15
gpExpandParam .................................................. 16
gpExtractParam .................................................. 17
gpGradient ......................................................... 17
gpLogLikeGradients .......................................... 18
gpLogLikelihood ............................................... 19
gpMeanFunctionGradient ..................................... 19
gpObjective ....................................................... 20
gpOptimise ....................................................... 21
gpOptions .......................................................... 21
gpOut .............................................................. 22
gpPlot .............................................................. 23
gpPosteriorMeanVar ........................................... 23
gpPosteriorSample ............................................. 24
gpSample .......................................................... 25
gpScaleBiasGradient ....................................... 25
gpTest ............................................................. 26
gpUpdateAD ...................................................... 27
gpUpdateKernels ............................................... 27
kernCompute ..................................................... 28
dernKernCreate .................................................. 29
dernKernDiagGradient ........................................ 30
dernKernDiagGradX ............................................ 31
dernKernGradient ............................................... 31
dernKernParamInit ............................................. 32
dernKernTest ..................................................... 33
dernModelDisplay .............................................. 34
dernModelExpandParam ....................................... 34
dernModelExtractParam ...................................... 35
dernModelGradient ............................................. 35
dernModelGradientCheck .................................... 36
dernModelOut .................................................... 37
dernModelOutputGrad ........................................ 37
dernMultiKernParamInit .................................... 38
dernNoiseCreate ............................................... 39
**basePlot**

*baseplot*

Plot a contour of the 2D Gaussian distribution with covariance matrix $K$.

**Description**

Creates the basic plot as an ellipse with major and minor radii as the square roots of the two eigenvalues of $K$.

**Usage**

```r
basePlot(K)
```

**Arguments**

- `K` the covariance matrix.

**See Also**

- `zeroAxes`

---

**cmpndKernParamInit**

*CMPND kernel parameter initialisation.*

**Description**

initialises the compound kernel structure with some default parameters.

**Usage**

```r
cmpndKernParamInit(kern)
```
Arguments

kern
the kernel structure which requires initialisation.

Value

kern
the kernel structure with the default parameters placed in.

See Also

kernCreate, kernParamInit.

Examples

## missing
**demAutoOptimiseGp**  
*Gaussian Process Optimisation Demo*

**Description**

Shows that by varying the length scale, an artificial data set has different likelihoods, yet there is an optimum for which the likelihood is maximised. This demo is similar to demOptimiseGp, only here, it is demonstrated how the length scale hyperparameter is optimised automatically through SCG (scaled conjugate gradients) numerical optimisation. Run multiple times to understand the effect of optimisation on randomly generated datasets.

**Usage**

```r
demAutoOptimiseGp(path=getwd(), filename='demAutoOptimiseGp', png=FALSE, gif=FALSE)
```

**Arguments**

- **path**: path where the plot images are saved.
- **filename**: name of saved images.
- **png**: save image as png.
- **gif**: save series of images as animated gif.

**See Also**

- gpOptions, kernCreate, gaussSamp, gpCreate, gpExpandParam, gpLogLikelihood, gpPosteriorMean

---

**demGpCov2D**  
*Gaussian Process 2D Covariance Demo*

**Description**

Gives the joint distribution for f1 and f2, two values of a function sampled from a Gaussian process. The plots show the joint distributions as well as the conditional for f2 given f1.

**Usage**

```r
demGpCov2D( ind=c(1,2), path = getwd(), filename = paste('demGpCov2D', ind[1],'_', ind[2], sep=''), png=FALSE, gif=FALSE )
```
Arguments

- `ind` indices of function values.
- `path` path where the plot images are saved.
- `filename` name of saved images.
- `png` save image as png.
- `gif` save series of images as animated gif.

See Also

demGpSample, basePlot, zeroAxes.

demGpSample

Gaussian Process Sampling Demo

Description

This example shows how points which look like they come from a function to be sampled from a Gaussian distribution. The sample is 25 dimensional and is from a Gaussian with a particular covariance.

Usage

demGpSample(bw=FALSE, path=getwd(), filename='gpSample', png=FALSE)

Arguments

- `bw` greyscale plots.
- `path` path where the plot images are saved.
- `filename` name of saved images.
- `png` save image as png.

See Also

kernCreate, kernCompute, gaussSamp, demGpSample, basePlot, zeroAxes.
**demInterpolation**

### Gaussian Process Interpolation Demo

#### Description

Plots, consecutively, an increasing number of data points, followed by an interpolated fit through the data points using a Gaussian process. This is a noiseless system, and the data is sampled from a GP with a known covariance function. The curve is then recovered with minimal uncertainty after only nine data points are included.

#### Usage

```r
demInterpolation(path=getwd(), filename='demInterpolation', png=FALSE, gif=FALSE)
```

#### Arguments

- **path**: path where the plot images are saved.
- **filename**: name of saved images.
- **png**: save image as png.
- **gif**: save series of images as animated gif.

#### See Also

- `gpOptions`, `kernCreate`, `kernCompute`, `gaussSamp`, `kernDiagCompute`, `gpCreate`, `gpPlot`

---

**demOptimiseGp**

### Gaussian Process Optimisation Demo

#### Description

Shows a series of plots of a Gaussian process with different length scales fitted to six data points. For each plot there is a corresponding plot of the log likelihood. The log likelihood peaks for a length scale close to 1. This was the length scale used to generate the data.

#### Usage

```r
demOptimiseGp(path=getwd(), filename='demOptimiseGp', png=FALSE, gif=FALSE)
```

#### Arguments

- **path**: path where the plot images are saved.
- **filename**: name of saved images.
- **png**: save image as png.
- **gif**: save series of images as animated gif.
expTransform

See Also

gpOptions, kernCreate, kernCompute, gaussSamp, kernDiagCompute, gpCreate, gpExpandParam, gpPlot, demInterpolation

demRegression  Gaussian Process Regression Demo

Description

The regression demo very much follows the format of the interpolation demo. Here the difference is that the data is sampled with noise. Fitting a model with noise means that the regression will not necessarily pass right through each data point.

Usage

demRegression(path=getwd(), filename='demRegression', png=FALSE, gif=FALSE)

Arguments

path  path where the plot images are saved.
filename  name of saved images.
png  save image as png.
gif  save series of images as animated gif.

See Also

gpOptions, kernCreate, kernCompute, gaussSamp, kernDiagCompute, gpCreate, gpPlot, demInterpolation

expTransform  Constrains a parameter.

Description

contains commands to constrain parameters to be positive via exponentiation or within a fixed interval via the sigmoid function.

Usage

expTransform(x, transform)
sigmoidTransform(x, transform)
boundedTransform(x, transform, bounds)
gaussianNoiseOut

Arguments

- **x**: input argument.
- **transform**: type of transform, 'atox' maps a value into the transformed space (i.e. makes it positive). 'xtoa' maps the parameter back from transformed space to the original space. 'gradfact' gives the factor needed to correct gradients with respect to the transformed parameter.
- **bounds**: a 2-vector of bounds of allowed values in boundedTransform

Value

Return value as selected by transform

See Also

- modelOptimise

Examples

```
# Transform unconstrained parameter -4 to a positive value
expTransform(-4, 'atox')

# Transform a bounded parameter in (1,3) to an unconstrained one
boundedTransform(2, 'xtoa', c(1,3))
```

---

**gaussianNoiseOut**: Compute the output of the GAUSSIAN noise given the input mean and variance.

Description

computes the output for the Gaussian noise given input mean and variances.

Usage

gaussianNoiseOut(noise, mu, varSigma)

Arguments

- **noise**: the noise structure for which the output is computed.
- **mu**: the input mean values.
- **varSigma**: the input variance values.

Value

- **y**: the output from the noise model.
gaussianNoiseParamInit

**GAUSSIAN noise parameter initialisation.**

**Description**

initialises the Gaussian noise structure with some default parameters.

**Usage**

`gaussianNoiseParamInit(noise, y)`

**Arguments**

- `noise` the noise structure which requires initialisation.
- `y` the data design matrix.

**Value**

- `noise` the noise structure with the default parameters placed in.

**See Also**

`noiseCreate, noiseParamInit`.

**Examples**

```r
## missing
```
**gaussSamp**

Sample from a Gaussian with a given covariance.

**Description**

samples a given number of samples from a Gaussian with a given covariance matrix.

**Usage**

```r
gaussSamp(mu=matrix(0,nrow=dim(Sigma)[1]), Sigma, numSamps)
```

**Arguments**

- `mu`: the mean of the Gaussian to sample from.
- `Sigma`: the covariance of the Gaussian to sample from.
- `numSamps`: the number of samples to take from the Gaussian.

**Value**

- `y`: the samples from the Gaussian

**See Also**

`rnorm`, `eigen`.

**Examples**

```r
## missing
```

---

**gpBlockIndices**

Return indices of given block.

**Description**

returns the indices of a given block for the PITC approximation.

**Usage**

```r
gpBlockIndices(model, blockNo)
```

**Arguments**

- `model`: the model for which the indices are being computed.
- `blockNo`: the block number for which the indices are required.
Value
indices the data indices associated with given block.

See Also
gpComputeAlpha, gpCovGrads, gpLogLikeGradients, gpLogLikelihood, gpUpdateAD.

Examples
## missing

---

gpComputeAlpha Update the vector ‘alpha’ for computing posterior mean quickly.

Description
updates the vectors that are known as ‘alpha’ in the support vector machine, in other words invK*y, where y is the target values.

Usage
gpComputeAlpha(model, m)

Arguments
model the model for which the alphas are going to be updated.
m the values of m for which the updates will be made.

Value
model the model with the updated alphas.

See Also
gpCreate, gpUpdateAD, gpUpdateKernels.

Examples
## missing
gpComputeM

Compute the matrix m given the model.

Description
computes the matrix m (the scaled, bias and mean function removed matrix of the targets), given
the model.

Usage
gpComputeM(model)

Arguments
model the model for which the values are to be computed.

Value
m the scaled, bias and mean function removed values.

See Also
gpCreate, gpComputeAlpha, gpUpdateAD.

Examples
## missing

gpCovGrads

Sparse objective function gradients wrt Covariance functions for in-
ducing variables.

Description
gives the gradients of the log likelihood with respect to the components of the sparse covariance (or
the full covariance for the ftc case).

Usage
gpCovGrads(model, M)

Arguments
model the model for which the gradients are to be computed.
M The training data for which the computation is to be made
Value

- **gK_uu**: the gradient of the likelihood with respect to the elements of \( K_{uu} \) (or in the case of the 'ftc' criterion the gradients with respect to the kernel).
- **gK_uf**: the gradient of the likelihood with respect to the elements of \( K_{uf} \).
- **gLambda**: the gradient of the likelihood with respect to the diagonal term in the fitc approximation and the blocks of the pitc approximation.
- **gBeta**: the gradient with respect to the beta term in the covariance structure.

See Also

- `gpCreate`, `gpLogLikeGradients`.

Examples

```r
## missing
```

---

**Description**

tests the gradients of the covariance to ensure they are correct.

**Usage**

gpCovGradsTest(model)

**Arguments**

- **model**: the model to be tested.

**Value**

- **model**: the model that was tested.

See Also

- `gpCreate`, `gpCovGrads`.

Examples

```r
## missing
```
**gpCreate**  
*Create a GP model with inducing variables/pseudo-inputs.*

**Description**

Creates a Gaussian process model structure with default parameter settings as specified by the options vector.

**Usage**

```
gpCreate(q, d, X, y, options)
```

**Arguments**

- `q`: input data dimension.
- `d`: the number of processes (i.e. output data dimension).
- `X`: the input data matrix.
- `y`: the target (output) data.
- `options`: options structure as defined by `gpOptions.R`.

**Value**

- `model`: model structure containing the Gaussian process.

**See Also**

`gpOptions`.

**Examples**

```
## missing
```

---

**gpDataIndices**  
*Return indices of present data.*

**Description**

returns the indices of data which is not missing for a given dimension in the GP-LVM and a block number in the PITC approximation.

**Usage**

```
gpDataIndices(model, dimNo, blockNo)
```
**Arguments**

- `model`: the model for which the indices are being returned.
- `dimNo`: the dimension for which the presence of missing data is being looked at.
- `blockNo`: the block number in the PITC approximation for which the indices are required.

**Value**

- `ind`: indices of training data along that dimension which isn’t missing.

**See Also**

`gpCreate`.

**Examples**

```r
## missing
```

---

**gpExpandParam**

*Expand a parameter vector into a GP model.*

**Description**

 takes the given vector of parameters and places them in the model structure, it then updates any stored representations that are dependent on those parameters, for example kernel matrices etc..

**Usage**

```r
gpExpandParam(model, params)
```

**Arguments**

- `model`: the model structure for which parameters are to be updated.
- `params`: a vector of parameters for placing in the model structure.

**Value**

- `model`: a returned model structure containing the updated parameters.

**See Also**

`gpCreate, gpExtractParam, modelExtractParam, gpUpdateKernels`.

**Examples**

```r
## missing
```
gpExtractParam

Extract a parameter vector from a GP model.

Description

does the same as above, but also returns parameter names.

Usage

gpExtractParam(model, only.values=TRUE, ...)

Arguments

model      the model structure containing the information about the model.
only.values (logical) do not return parameter names.
...         optional additional arguments.

Value

params     a vector of parameters from the model.
names      cell array of parameter names.

See Also

gpCreate, gpExpandParam, modelExtractParam.

Examples

## missing

gpGradient

Gradient wrapper for a GP model.

Description

wraps the log likelihood gradient function to return the gradient of the negative of the log likelihood. This can then be used in, for example, NETLAB, minimisation tools.

Usage

gpGradient(params, model)

Arguments

params  the parameters of the model.
model   the model for which gradients will be computed.
gpLogLikeGradients

Value

\( g \)

the returned gradient of the negative log likelihood for the given parameters.

See Also

\( \text{gpCreate}, \text{gpGradient}, \text{gpLogLikeGradients}, \text{gpOptimise}. \)

Examples

```r
## missing
```

gpLogLikeGradients  \textit{Compute the gradients for the parameters and X.}

Description

computes the gradients of the Gaussian process log likelihood with respect to the model parameters (and optionally, as above with respect to inducing variables and input data) given the target data, input data and inducing variable locations.

Usage

\[
gpLogLikeGradients( \text{model}, X=\text{model}$X, M, X_u, gX_u.return=FALSE,\ 
gX.return=FALSE, g\_beta.return=FALSE )
\]

Arguments

- **model**: the model structure for which gradients are computed.
- **X**: the input data locations for which gradients are computed.
- **M**: the scaled and bias removed target data for which the gradients are computed.
- **X_u**: the inducing variable locations for which gradients are computed.
- **gX_u.return**: (logical) return the gradient of the log likelihood with respect to the inducing variables. If inducing variables aren’t being used this returns zero.
- **gX.return**: (logical) return the gradient of the log likelihood with respect to the input data locations.
- **g\_beta.return**: (logical) to return the gradient of the log likelihood with respect to beta.

Value

\( gParam \)

contains the gradient of the log likelihood with respect to the model parameters (including any gradients with respect to beta).

See Also

\( \text{gpLogLikelihood}. \)
gpLogLikelihood

Examples
## missing

---

**gpLogLikelihood**  
*Compute the log likelihood of a GP.*

**Description**
computes the log likelihood of a data set given a GP model.

**Usage**
gpLogLikelihood(model)

**Arguments**
- **model**  
  the GP model for which log likelihood is to be computed.

**Value**
- **11**  
  the log likelihood of the data in the GP model.

**See Also**
gpCreate, gpLogLikeGradients, modelLogLikelihood.

---

**gpMeanFunctionGradient**  
*Compute the log likelihood gradient wrt the scales.*

**Description**
computes the gradient of the log likelihood with respect to the scales. In the future the gradients with respect to the biases may also be included.

**Usage**
gpMeanFunctionGradient(model)

**Arguments**
- **model**  
  the model for which the gradients are to be computed.
gpObjective

Value

the gradients of the likelihood with respect to the mean function's parameters.

See Also

gpCreate, gpScaleBiasGradient, gpLogLikeGradients, gpLogLikelihood.

Examples

## missing

---

gpObjective   Wrapper function for GP objective.

Description

returns the negative log likelihood of a Gaussian process model given the model structure and a vector of parameters. This allows the use of NETLAB minimisation functions to find the model parameters.

Usage

gpObjective(params, model)

Arguments

params   the parameters of the model for which the objective will be evaluated.
model    the model structure for which the objective will be evaluated.

Value

f    the negative log likelihood of the GP model.

See Also

gpCreate, gpGradient, gpLogLikelihood, gpOptimise.

Examples

## missing
gpOptimise

Optimise the inducing variable based kernel.

Description
optimises the Gaussian process model for a given number of iterations.

Usage
gpOptimise(model, display, iters, gradcheck)

Arguments
model the model to be optimised.
display whether or not to display while optimisation proceeds, set to 2 for the most verbose and 0 for the least verbose.
iters number of iterations for the optimisation.
graddateh (logical) do a gradient check.

Value
model the optimised model.

See Also
gpCreate, gpGradient, gpObjective.

Examples
## missing

---

gpOptions

Return default options for GP model.

Description
returns the default options in a structure for a GP model.

Usage
gpOptions(approx)
gpOut

Arguments

approx approximation type, either 'ftc' (no approximation), 'dtcvar' (variational sparse approximation) 'dtc' (deterministic training conditional), 'fitc' (fully independent training conditional) or 'pitc' (partially independent training conditional).

Value

options structure containing the default options for the given approximation type.

See Also

gpCreate.

Examples

## missing

---

\textit{gpOut} \hspace{1cm} \textit{Evaluate the output of an Gaussian process model.}

Description

evaluates the output of a given Gaussian process model.

Usage

gpOut(model, x)

Arguments

model the model for which the output is being evaluated.

x the input position for which the output is required.

Value

y the output of the GP model. The function checks if there is a noise model associated with the GP, if there is, it is used, otherwise the mean of gpPosteriorMeanVar is returned.

See Also

gpCreate, gpPosteriorMeanVar.

Examples

## missing
gpPlot

Gaussian Process Plotter

Description

Plots the GP mean and variance.

Usage

```r
gpPlot( modelL xstarL muL sL simposeL xlimL ylimL xlabL ylabL colL titleL )
```

Arguments

- `model`: the model structure for which GP mean and variance are to be plotted.
- `xstar`: the input positions for which the mean and variance will be plotted.
- `mu`: the precomputed GP posterior mean vector.
- `s`: the precomputed GP posterior variance vector.
- `simpose`: vector of datapoints to be superimposed on the plot with added white noise.
- `xlim`: x-axis plotting limits.
- `ylim`: y-axis plotting limits.
- `xlab`: x-axis label.
- `ylab`: y-axis label.
- `col`: color for plotting the GP mean and variance.
- `title`: plot title.

See Also

`gpPosteriorMeanVar, polygon, zeroAxes`.

---

gpPosteriorMeanVar

Mean and variances of the posterior at points given by X.

Description

returns the posterior mean and variance for a given set of points.

Usage

```r
gpPosteriorMeanVar(modelL X, varsigma.return=FALSE)
```
Arguments

- **model**: the model for which the posterior will be computed.
- **X**: the input positions for which the posterior will be computed.
- **varsigma.return**: (logical) compute variances.

Value

- **mu**: the mean of the posterior distribution.
- **sigma**: the variances of the posterior distributions.

See Also

- `gpCreate`.

Examples

```r
## missing
```

---

**gpposteriorsample**  
*Plot Samples from a GP Posterior.*

Description

Gaussian processes are non-parametric models. They are specified by their covariance function and a mean function. When combined with data observations a posterior Gaussian process is induced. This function samples from that posterior.

Usage

```r
gpPosteriorSample( kernType, numSamps=10, params=NULL, 
                  lims=c(-3,3), path=getwd(), png=FALSE )
```

Arguments

- **kernType**: the type of kernel to sample from.
- **numSamps**: the number of samples to take.
- **params**: parameter vector for the kernel.
- **lims**: limits of the x axis.
- **path**: path where the plot images are saved.
- **png**: save image as png.

See Also

- `gpOptions`, `kernCreate`, `kernCompute`, `gaussSamp`, `zeroAxes`.  

**gpSample**

*Plot Samples from a GP.*

**Description**

creates a plot of samples from a kernel with the given parameters and variance.

**Usage**

```r
gpSample(kernType, numSamps=10, params=NULL, lims=c(-3,3), path=getwd(), png=FALSE)
```

**Arguments**

- `kernType`: the type of kernel to sample from.
- `numSamps`: the number of samples to take.
- `params`: parameter vector for the kernel.
- `lims`: limits of the x axis.
- `path`: path where the plot images are saved.
- `png`: save image as png.

**See Also**

`gpOptions`.

---

**gpScaleBiasGradient**

*Compute the log likelihood gradient wrt the scales.*

**Description**

computes the gradient of the log likelihood with respect to the scales. In the future the gradients with respect to the biases may also be included.

**Usage**

```r
gpScaleBiasGradient(model)
```

**Arguments**

- `model`: the model for which the gradients are to be computed.

**Value**

- `g`: the gradients of the likelihood with respect to the scales.
See Also

`gpCreate, gpLogLikeGradients, gpLogLikelihood`.

Examples

```# missing```
**gpUpdateAD**

*Update the representations of A and D associated with the model.*

**Description**

updates the representations of A and D in the model when called by gpUpdateKernels.

**Usage**

```
gpUpdateAD(model, X)
```

**Arguments**

- `model`: the model for which the representations are being updated.
- `X`: the X values for which the representations are being computed.

**Value**

- `model`: the model with the A and D representations updated.

**See Also**

`gpUpdateKernels`, `gpExpandParam`.

**Examples**

```r
test <- array(sample(1:500, 10, replace = T), dim = c(2, 5))
gpUpdateAD(model, test)
```

---

**gpUpdateKernels**

*Update the kernels that are needed.*

**Description**

updates any representations of the kernel in the model structure, such as invK, logDetK or K.

**Usage**

```
gpUpdateKernels(model, X, X_u)
```

**Arguments**

- `model`: the model structure for which kernels are being updated.
- `X`: the input locations for update of kernels.
- `X_u`: the inducing input locations.
kernCompute

Value

model the model structure with the kernels updated.

See Also

gpExpandParam, gpCreate.

Examples

## missing

### kernCompute

Compute the kernel given the parameters and X.

Usage

kernCompute(kern, x, x2)
kernDiagCompute(kern, x)

Arguments

kern kernel structure to be computed.
x depending on the number of inputs, x can be the input data matrix (rows are data points) to the kernel computation, or the first input matrix to the kernel computation (forms the rows of the kernel).
x2 second input matrix to the kernel computation (forms the columns of the kernel).

Details

K <- kernCompute(kern, x) computes a kernel matrix for the given kernel type given an input data matrix.
K <- kernCompute(kern, x1, x2) computes a kernel matrix for the given kernel type given two input data matrices, one for the rows and one for the columns.
K <- kernDiagCompute(kern, x) computes the diagonal of a kernel matrix for the given kernel.
K <- *X*kernCompute(kern1, kern2, x) K <- *X*kernCompute(kern1, kern2, x1, x2) same as above, but for cross combinations of two kernels, kern1 and kern2.

Value

K computed elements of the kernel structure.
Kd vector containing computed diagonal elements of the kernel structure.
**kernCreate**

**See Also**

- `kernCreate`

**Examples**

```r
kern <- kernCreate(1, 'rbf')
K <- kernCompute(kern, as.matrix(3:8))
```

---

**Description**

Initialise a kernel structure.

**Usage**

```r
kernCreate(x, kernType, kernOptions=NULL)
```

**Arguments**

- `x` (Input data values (from which kernel will later be computed)).
- `kernType` (Type of kernel to be created, some standard types are 'lin', 'rbf', 'white', 'bias' and 'rbfard'. If a cell of the form 'cmpnd', 'rbf', 'lin', 'white' is used a compound kernel based on the sum of the individual kernels will be created. The 'cmpnd' element at the start of the sequence is optional. Furthermore, 'tensor', 'rbf', 'lin' can be used to give a tensor product kernel, whose elements are the formed from the products of the two individual kernel's elements and 'multi', 'rbf', ... can be used to create a block structured kernel for use with multiple outputs. Finally the form 'parametric', struct('opt1', val1), 'rbf' can be used to pass options to other kernels).
- `kernOptions` (the kernel options).

**Details**

- `kern <- kernCreate(x, kernType)` input points and a kernel type.
- `kern <- kernCreate(dim, kernType)` creates a kernel matrix structure given the dimensions of the design matrix and the kernel type.

**Value**

- `kern` (The kernel structure).

**See Also**

- `kernParamInit`
Examples

## missing
kernDiagGradX

Compute the gradient of the kernel wrt X.

Description

computes the gradient of the (diagonal of the) kernel matrix with respect to the elements of the
design matrix given in X.

Usage

kernDiagGradX(kern, x)
kernGradX(kern, x1, x2)

Arguments

kern the kernel structure for which gradients are being computed.
x the input data in the form of a design matrix.
x1 row locations against which gradients are being computed.
x2 column locations against which gradients are being computed.

Value

gX the gradients of the diagonal with respect to each element of X. The returned
matrix has the same dimensions as X.
gX2 the returned gradients. The gradients are returned in a matrix which is numData
x numInputs x numData. Where numData is the number of data points and
numInputs is the number of input dimensions in X.

See Also

kernGradient

kernGradient

Compute the gradient wrt the kernel parameters.

Description

Compute the gradient wrt the kernel parameters.

Usage

kernGradient(kern, x, ...)
  ## kernGradient(kern, x, partial)
  ## kernGradient(kern, x, x1, x2, partial_...
Arguments

- **kern**: the kernel structure for which the gradients are being computed.
- **x**: the input locations for which the gradients are being computed.
- **...**: other arguments such as: 'partial', a matrix of partial derivatives of the function of interest with respect to the kernel matrix. The argument takes the form of a square matrix of dimension `numData`, where `numData` is the number of rows in `X`, 'x1', the input locations associated with the rows of the kernel matrix, 'x2', the input locations associated with the columns of the kernel matrix, 'partial_', matrix of partial derivatives of the function of interest with respect to the kernel matrix. The matrix should have the same number of rows as `x1` and the same number of columns as `x2` has rows.

Details

\[ g \leftarrow \text{kernGradient}(kern, x, \partial) \]
\[ g \leftarrow \text{kernGradient}(kern, x, \partial) \]
\[ g \leftarrow \text{kernGradient}(kern, x, x', \partial) \]
\[ g \leftarrow \text{kernGradient}(kern, x, x', \partial) \]
\[ g \leftarrow \text{kernGradient}(kern, x, x', \partial) \]
\[ g \leftarrow \text{kernGradient}(kern, x, x', \partial) \]

- **Value**: gradients of the function of interest with respect to the kernel parameters. The ordering of the vector should match that provided by the function `kernExtractParam`.

See Also

- `kernCompute`, `kernExtractParam`.

Examples

```r
kern <- kernCreate(1, 'rbf')
g <- kernGradient(kern, as.matrix(c(1, 4)), array(1, c(2, 2)))
```

Description

Initialises the parameters of a kernel.
### kernTest

**Usage**

```
  kernParamInit(kern)
```

**Arguments**

- **kern**
  - the kernel structure for which the parameters will be initialised.

**Value**

```
  kern
```
  - the kernel structure with the parameters initialised.

**See Also**

- `kernCreate`

**Examples**

```
## missing
```

---

### kernTest

*Run some tests on the specified kernel.*

**Description**

runs some tests on a given kernel structure to ensure it is correctly implemented.

**Usage**

```
  kernTest(kernTypeL numInL tieParamNamesL listL testindexL NULL)
```

**Arguments**

- **kernType**
  - type of kernel to test. For example, 'rbf' or 'cmpnd', 'rbf', 'lin', 'white'.
- **numIn**
  - the number of input dimensions (default is 4).
- **tieParamNames**
  - list of regular expressions for parameter names that should be tied (default is none).
- **testindex**
  - indices of the covariance gradient to test for.

**Value**

```
  kern
```
  - the kernel as it was used in the tests.

**See Also**

- `kernCreate`

**Examples**

```
## missing
```
**modelDisplay**  
*Display a model.*

**Description**

displays the parameters of the model/kernel and the model/kernel type to the console.

**Usage**

```r
modelDisplay(model, ...)```

**Arguments**

- `model` the model/kernel structure to be displayed.
- `...` optional additional arguments.

**See Also**

- `modelextractparam`
- `modelexpandparam`

---

**modelExpandParam**  
*Update a model structure with new parameters or update the posterior processes.*

**Description**

Update a model structure or component with new parameters, or update the posterior processes.

**Usage**

```r
modelExpandParam(model, params)
modelUpdateProcesses(model, predt=NULL)```

**Arguments**

- `model` the model structure to be updated.
- `params` vector of parameters.
- `predt` .

**Details**

- `model <- modelExpandParam(model, params)` returns a model structure filled with the parameters in the given vector. This is used as a helper function to enable parameters to be optimised in, for example, the optimisation functions.
- `model <- modelUpdateProcesses(model)` updates posterior processes of the given model.
**modelExtractParam**

Value
- model: updated model structure.

See Also
- modelExtractParam

---

**Model Extract Parameters**

Extract parameters from the model into a vector of parameters for optimisation.

**Usage**

```r
modelExtractParam(model, only.values=TRUE, untransformed.values=FALSE)
```

**Arguments**
- model: the model structure containing the parameters to be extracted.
- only.values: include parameter names in the returned vector.
- untransformed.values: return actual values, not transformed values used by the optimisers.

**Value**
- param: vector of parameters extracted from the model.

**See Also**
- modelExpandParam

---

**Model Gradient**

Model log-likelihood/objective error function and its gradient.

**Description**

`modelGradient` gives the gradient of the objective function for a model. By default the objective function (`modelObjective`) is a negative log likelihood (`modelLogLikelihood`).

**Usage**

```r
modelObjective(params, model, ...)
modelLogLikelihood(model)
modelGradient(params, model, ...)
```
modelGradientCheck

Arguments

params parameter vector to evaluate at.
model model structure.
... optional additional arguments.

Value

g the gradient of the error function to be minimised.
v the objective function value (lower is better).
ll the log-likelihood value.

See Also

modelOptimise.

modelGradientCheck Check gradients of given model.

Description

checks the supplied gradient function and the supplied objective function to ensure that the numerical gradients (as computed with the objective function) match the analytically computed gradients.

Usage

modelGradientCheck(model, ...)

Arguments

model the model for which gradients are to be checked.
... additional arguments that are passed to the objective and gradient functions (after the parameter vector which is always assumed to be the first argument passed).

See Also

modelObjective, modelGradient.
modelOut  

Give the output of a model for given X.

Description
Give the output of a model for given X.

Usage
modelOut(model, X, Phi.return=FALSE, ...)

Arguments
- model: structure specifying the model.
- X: input location(s) for which output is to be computed.
- Phi.return: (logical) return the basis function(s) as well.
- ...: optional additional arguments.

Details
Y <- modelOut(model, X) gives the output of the model for a given input X. For latent variable models it gives a position in data space given a position in latent space.

Value
- Y: output location(s) corresponding to given input locations.
- Phi: output basis function(s) corresponding to given input

Examples
## missing

modelOutputGrad  

Compute derivatives with respect to params of model outputs.

Description
Compute derivatives with respect to params of model outputs.

Usage
modelOutputGrad(model, X, dim)
Arguments

- **model**: the model structure for which gradients are computed.
- **X**: input locations where gradients are to be computed.
- **dim**: the dimension of the model for which gradients are required.

Details

\[ g \leftarrow \text{modelOutputGrad}(\text{model}, X) \]
gives the gradients of the outputs from the model with respect to the parameters for a given set of inputs.

\[ g \leftarrow \text{modelOutputGrad}(\text{model}, X, \text{dim}) \]
gives the gradients of the outputs from the model with respect to the parameters for a given set of inputs.

Value

\[ g \]
gradients of the model output with respect to the model parameters for the given input locations. The size of the returned matrix is of dimension number of data x number of parameters x number of model outputs (which maintains compatibility with NETLAB).

See Also

- `modelLogLikelihood`

Examples

```r
## missing
```

---

**multiKernParamInit**  
*MULTI* kernel parameter initialisation.

Description

initialises the multiple output block kernel structure with some default parameters.

Usage

`multiKernParamInit(kern)`

Arguments

- **kern**: the kernel structure which requires initialisation.

Value

- **kern**: the kernel structure with the default parameters placed in.
noiseCreate

See Also

kernCreate, kernParamInit.

Examples

## missing

---

### noiseCreate

**Initialise a noise structure.**

**Description**

takes a noise type and a target vector and initialises a noise structure from it. The parameters of the noise structure are the set by calling noiseParamInit.

**Usage**

noiseCreate(noiseType, y)

**Arguments**

- **noiseType**: the type of noise to be created (e.g. 'gaussian', 'probit', 'ncnm').
- **y**: the target vector.

**See Also**

noiseParamInit.

**Examples**

## missing

---

### noiseOut

**Give the output of the noise model given the mean and variance.**

**Description**

computes the output for the given noise given input mean and variances.

**Usage**

noiseOut(noise, mu, varsigma)
Arguments

- **noise**: the noise structure for which the output is computed.
- **mu**: the input mean values.
- **varsigma**: the input variance values.

Value

- **y**: the output from the noise model.

See Also

- `noiseParamInit`, `noisecreate`.

Examples

```r
### missing
```

---

**noiseParamInit**

*Noise model's parameter initialisation.*

Description

Initialises the noise structure with some default parameters.

Usage

```r
noiseParamInit(noise, y)
```

Arguments

- **noise**: the noise structure which requires initialisation.
- **y**: the data design matrix.

Value

- **noise**: the noise structure with the default parameters placed in.

See Also

- `noisecreate`.

Examples

```r
### missing
```
optimiDefaultConstraint

Returns function for parameter constraint.

Description
returns the current default function for constraining a parameter.

Usage
optimiDefaultConstraint(constraint)

Arguments
constraint the type of constraint you want to place on the parameter, options include 'positive' (gives an 'exp' constraint) and 'zeroone' (gives a 'sigmoid' constraint).

Value
val a list with two components: 'func' for the name of function used to apply the constraint, and 'hasArgs' for a boolean flag if the function requires additional arguments.

See Also
expTransform, sigmoidTransform

Examples
optimiDefaultConstraint('positive')
optimiDefaultConstraint('bounded')

rbfKernDiagGradX Gradient of RBF kernel’s diagonal with respect to X.

Description
computes the gradient of the diagonal of the radial basis function kernel matrix with respect to the elements of the design matrix given in X.

Usage
rbfKernDiagGradX(kern, X)
The kernel function for which gradients are being computed.

x
the input data in the form of a design matrix.

Value

gx
the gradients of the diagonal with respect to each element of X. The returned matrix has the same dimensions as X.

See Also

rbfKernParamInit, kernDiagGradX, rbfKernGradX.

Examples

## missing

### Description

computes the gradient of the radial basis function kernel with respect to the input positions where both the row positions and column positions are provided separately.

#### Usage

rbfKernGradX(kern, x1, x2)

#### Arguments

kern
kernel structure for which gradients are being computed.

x1
row locations against which gradients are being computed.

x2
column locations against which gradients are being computed.

#### Value

g
the returned gradients. The gradients are returned in a matrix which is numData2 x numInputs x numData1. Where numData1 is the number of data points in X1, numData2 is the number of data points in X2 and numInputs is the number of input dimensions in X.

See Also

rbfKernParamInit, kernGradX, rbfKernDiagGradX.

Examples

## missing
**rbbKernParamInit**

**rbbKernParamInit**  
*RBF kernel parameter initialisation.*

**Description**

initialises the radial basis function kernel structure with some default parameters.

**Usage**

```r
rbbKernParamInit(kern)
```

**Arguments**

- `kern`  
  the kernel structure which requires initialisation.

**Value**

- `kern`  
  the kernel structure with the default parameters placed in.

**See Also**

- `kernCreate`, `kernParamInit`.

**Examples**

```r
## missing
```

**SCGOptim**

**SCGOptim**  
*Optimise the given function using (scaled) conjugate gradients.*

**Description**

Optimise the given function using (scaled) conjugate gradients.

**Usage**

```r
## options <- optimiDefaultOptions()
SCGOptim(x, fn, grad, options, ...)
CGOptim(x, fn, grad, options, ...)
modelOptimise(model, options, ...)
```
Arguments

- `model` the model to be optimised.
- `x` initial parameter values.
- `fn` objective function to minimise
- `grad` gradient function of the objective
- `options` options structure like one returned by `optimidefaultoptions`. The fields are interpreted as:
  - `option[1]` : number of iterations
  - `option[2]` : interval for the line search
  - `option[3]` : tolerance for `x` to terminate the loop
  - `option[4]` : tolerance for `fn` to terminate the loop
  - `option$display` : option of showing the details of optimisation
  - ... extra arguments to pass to `fn` and `grad`

Value

- `options` an options structure
- `newParams` optimised parameter values
- `model` the optimised model.

See Also

- `modelObjective`, `modelGradient`

Examples

```r
# Not run to speed up package checks
# model <- GPlearn(..., dontOptimise=TRUE)
# options <- optimidefaultoptions()
# model <- modelOptimise(model, options)
```

---

**whiteKernDiagGradX**  
*Gradient of WHITE kernel’s diagonal with respect to X.*

Description

computes the gradient of the diagonal of the white noise kernel matrix with respect to the elements of the design matrix given in `X`.

Usage

```r
whiteKernDiagGradX(kern, X)
```

Arguments

- `kern` the kernel structure for which gradients are being computed.
- `X` the input data in the form of a design matrix.
whiteKernGradX

Value

gX the gradients of the diagonal with respect to each element of X. The returned matrix has the same dimensions as X.

See Also

whiteKernParamInit, kernDiagGradX, whiteKernGradX.

Examples

## missing

whiteKernGradX Gradient of WHITE kernel with respect to input locations.

Description

computes the gradient of the white noise kernel with respect to the input positions where both the row positions and column positions are provided separately.

Usage

whiteKernGradX(kern, x1, x2)

Arguments

kern kernel structure for which gradients are being computed.
x1 row locations against which gradients are being computed.
x2 column locations against which gradients are being computed.

Value

g the returned gradients. The gradients are returned in a matrix which is numData2 x numInputs x numData1. Where numData1 is the number of data points in X1, numData2 is the number of data points in X2 and numInputs is the number of input dimensions in X.

Examples

## missing
whiteKernParamInit  \textit{WHITE kernel parameter initialisation.}

\textbf{Description}
initialises the white noise kernel structure with some default parameters.

\textbf{Usage}
\texttt{whiteKernParamInit(kern)}

\textbf{Arguments}
- \texttt{kern} the kernel structure which requires initialisation.

\textbf{Value}
- \texttt{kern} the kernel structure with the default parameters placed in.

\textbf{See Also}
- \texttt{kernCreate}, \texttt{kernParamInit}.

\textbf{Examples}
\texttt{## missing}

\textbf{zeroAxes  } \textit{A function to move the axes crossing point to the origin.}

\textbf{Description}
moves the crossing point of the axes to the origin.

\textbf{Usage}
\texttt{zeroAxes(col='blue')}

\textbf{Arguments}
- \texttt{col} color of the axes.

\textbf{See Also}
- \texttt{plot}.

\textbf{Examples}
\texttt{## missing}
Index

*Topic model
  basePlot, 3
  cmpndKernParamInit, 3
  cmpndNoiseParamInit, 4
  demAutoOptimiseGp, 5
  demGpCov2D, 5
  demGpSample, 6
  demInterpolation, 7
  demOptimiseGp, 7
  demRegression, 8
  expTransform, 8
  gaussianNoiseOut, 9
  gaussianNoiseParamInit, 10
  gaussSamp, 11
  gpBlockIndices, 11
  gpComputeAlpha, 12
  gpComputeM, 13
  gpCovGrads, 13
  gpCovGradsTest, 14
  gpCreate, 15
  gpDataIndices, 15
  gpExpandParam, 16
  gpExtractParam, 17
  gpGradient, 17
  gpLogLikeGradients, 18
  gpLogLikelihood, 19
  gpMeanFunctionGradient, 19
  gpObjective, 20
  gpOptimise, 21
  gpOptions, 21
  gpOut, 22
  gpPlot, 23
  gpPosteriorMeanVar, 23
  gpScaleBiasGradient, 25
  gpTest, 26
  gpUpdateAD, 27
  gpUpdateKernels, 27
  kernCompute, 28
  kernCreate, 29
  kernDiagGradient, 30
  kernDiagGradX, 31
  kernGrad, 31
  kernParamInit, 32
  kernTest, 33
  modelDisplay, 34
  modelExpandParam, 34
  modelExtractParam, 35
  modelGradient, 35
  modelGradientCheck, 36
  modelOut, 37
  modelOutputGrad, 37
  multiKernParamInit, 38
  noiseCreate, 39
  noiseOut, 39
  noiseParamInit, 40
  optimiDefaultConstraint, 41
  rbfKernDiagGradX, 41
  rbfKernGradX, 42
  rbfKernParamInit, 43
  SCGoptim, 43
  whiteKernDiagGradX, 44
  whiteKernGradX, 45
  whiteKernParamInit, 46
  zeroAxes, 46

basePlot, 3, 6
boundedTransform (expTransform), 8
CGoptim (SCGoptim), 43
cgpdisimExpandParam (modelExpandParam), 34
cgpdisimExtractParam (modelExtractParam), 35
cgpdisimGrad (modelGradient), 35
cgpdisimLogLikeGradients (modelGradient), 35
cgpdisimLogLikelihood (modelGradient), 35
cgpdisimObjective (modelGradient), 35
INDEX

gsimUpdateProcesses
(modelExpandParam), 34
 gpTest, 26
gpUpdateAD, 12, 13, 27
gpUpdateKernels, 12, 16, 27, 27

ekernCompute, 6–8, 24, 28, 32
kernCreate, 4–8, 24, 29, 33, 39, 43, 46
kernDiagCompute, 7, 8
kernDiagCompute (kernCompute), 28
kernDiagGrad, 30, 30
kernDiagGradX, 31, 42, 45
kernDisplay (modelDisplay), 34
kernExpandParam (modelExpandParam), 34
kernExpandParam, 30, 32
kernExtractParam (modelExtractParam), 35
kernGradient, 30, 31, 31
kernGradX, 42
kernGradX (kernDiagGradX), 31
kernParamInit, 4, 29, 32, 39, 43, 46
kernTest, 33

localCovarianceGradients
(gpLogLikeGradients), 18
localSCovarianceGradients
(gpLogLikeGradients), 18

mlpKernCompute (kernCompute), 28
mlpKernDiagGradX (kernDiagGradX), 31
mlpKernExpandParam (modelExpandParam), 34
mlpKernExtractParam
(modelExtractParam), 35
mlpKernGradient (kernGradient), 31
mlpKernGradX (kernDiagGradX), 31
mlpOptions (gpOptions), 21
modelDisplay, 34
modelExpandParam, 34, 35
modelExtractParam, 16, 17, 34, 35, 35
modelGradient, 35, 36, 44
modelGradientCheck, 36
modelLogLikelihood, 19, 38
modelLogLikelihood (modelGradient), 35
modelObjective, 36, 44
modelObjective (modelGradient), 35
modelOptimise, 9, 36
modelOptimise (SCGoptim), 43
modelOut, 37
modelOutputGrad, 37

modelUpdateProcesses
(modelExpandParam), 34
multiKernCompute (kernCompute), 28
multiKernDiagCompute (kernCompute), 28
multiKernDisplay (modelDisplay), 34
multiKernExpandParam
(modelExpandParam), 34
multiKernExtractParam
(modelExtractParam), 35
multiKernGradient (kernGradient), 31
multiKernParamInit, 38
noiseCreate, 4, 10, 39, 40
noiseOut, 10, 39
noiseParamInit, 4, 10, 39, 40, 40
optimiDefaultConstraint, 41
optimiDefaultOptions (SCGoptim), 43
plot, 46
polygon, 23

rbfKernCompute (kernCompute), 28
rbfKernDiagCompute (kernCompute), 28
rbfKernDiagGradX, 41, 42
rbfKernDisplay (modelDisplay), 34
rbfKernExpandParam (modelExpandParam), 34
rbfKernExtractParam
(modelExtractParam), 35
rbfKernGradient (kernGradient), 31
rbfKernGradX, 42, 42
rbfKernGradXpoint (rbfKernGradX), 42
rbfKernParamInit, 42, 43
rnorm, 11

SCGoptim, 43
sigmoidTransform, 41
sigmoidTransform (expTransform), 8
simKernCompute (kernCompute), 28
simKernDiagCompute (kernCompute), 28
simKernDisplay (modelDisplay), 34
simKernExpandParam (modelExpandParam), 34
simKernExtractParam
(modelExtractParam), 35
simKernGradient (kernGradient), 31
simXrbfKernCompute (kernCompute), 28
simXrbfKernGradient (kernGradient), 31
simXsimKernCompute (kernCompute), 28
simXsimKernGradient (kernGradient), 31

translateKernCompute (kernCompute), 28
translateKernDiagCompute (kernCompute), 28
translateKernExpandParam (modelExpandParam), 34
translateKernExtractParam (modelExtractParam), 35
translateKernGradient (kernGradient), 31

whiteKernCompute (kernCompute), 28
whiteKernDiagCompute (kernCompute), 28
whiteKernDiagGradX, 44
whiteKernDisplay (modelDisplay), 34
whiteKernExpandParam (modelExpandParam), 34
whiteKernExtractParam (modelExtractParam), 35
whiteKernGradient (kernGradient), 31
whiteKernGradX, 45, 45
whiteKernParamInit, 45, 46
whiteXwhiteKernCompute (kernCompute), 28
whiteXwhiteKernGradient (kernGradient), 31

zeroAxes, 3, 6, 23, 24, 46