Package 'cocons'

September 6, 2024

Type Package

Title Covariate-Based Covariance Functions for Nonstationary Spatial Modeling

Version 0.1.2

Author Federico Blasi [aut, cre] (<https://orcid.org/0000-0001-9337-7154>), Reinhard Furrer [ctb] (<https://orcid.org/0000-0002-6319-2332>)

Maintainer Federico Blasi <federico.blasi@math.uzh.ch>

Description Estimation and prediction of nonstationary Gaussian process with modular covariate-based covariance functions. Routines for handling large datasets are also provided.

Encoding UTF-8

LazyData true

License GPL (>= 3)

Depends R (>= 3.5.0)

Imports Rcpp (>= 1.0.10), spam (>= 2.9.1), fields, optimParallel, methods, knitr

LinkingTo Rcpp, BH

BugReports https://github.com/blasif/cocons/issues

RoxygenNote 7.3.2

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2024-09-06 14:20:02 UTC

Contents

`cocons-package` *Covariate-based Covariance Functions for Nonstationary Gaussian Processes*

Description

Provides routines and methods for estimating and predicting nonstationary Gaussian process models with modular covariate-based covariance functions. Several sources of nonstationarity can be modeled based on spatial information, including a trend, marginal standard deviation, local geometric anisotropy, local nugget, and spatially varying smoothness. Each of these components is modeled separately. A sparse-induced version of the nonstationary covariance function is provided for large datasets to speed up computations. Models are estimated via maximum likelihood (and flavours of it such as penalized and profile maximum likelihood). A variety of functions are also included to compute prediction metrics and to visualize, simulate, and summarize these types of models. Details of the models can be found in the vignette.

Disclaimer

This package is provided "as is" without warranty of any kind, either express or implied. Backwards compatibility will not be offered until later versions.

Author(s)

Federico Blasi [aut, cre], <federico.blasi@uzh.ch>

Examples

```
## Not run:
    vignette("cocons", package = "cocons")
    methods(class = "coco")

## End(Not run)
```

---

coco *Creates a coco S4 object*

Description

Creates an S4 object of class coco, which is the centerpiece of the **cocons** package. The function provides a set of consistency checks for ensuring the suitability of the different objects involved.

Usage

```
coco(type, data, locs, z, model.list, info, output = list())
```

Arguments

type	(character) One of two available types "dense" or "sparse". See description.
data	(data.frame) A data.frame with covariates information, where colnames(data) matches model.list specification.
locs	(matrix) A matrix with spatial locations.

`z`	(`vector` or `matrix`) A matrix of $n \times r$ response realizations, one realization per column. When considering only one realization, a vector can also be provided.
`model.list`	(`list`) A `list` specifying a model for each aspect of the spatial structure.
`info`	(`list` or `NULL`) A `list` specifying characteristics of the coco object.
`output`	(`list` or `NULL`) Empty or the resulting object from running optimParallel, adding to this a list with boundary information (check getBoundaries to check the expected structure).

Details

Two types of `coco` objects are available, each assuming a different type of covariance matrix for the Gaussian process. Type `"dense"` builds dense covariance matrices (non zero elements), while type `"sparse"` builds sparse covariance matrices by tapering the dense covariance matrix with a compact isotropic compact-supported correlation matrix [1]. Type `"sparse"` allows a set of efficient algorithms, thus making it more suitable for large sample sizes.

An important component of the `coco` S4 class is the `model.list` specification, involving individual formulas provided as a list, where each of them specifies a covariate-based parametric model for a specific source of nonstationarity. It involves `"trend"` for the spatial trend, the `"std.dev"` for the marginal standard deviation, `"scale"`, `"aniso"` and `"tilt"`, each of them shaping specific aspects of the local spatial geometrically anisotropy structure, `"smooth"` handling local smoothness, and `"nugget"` handling the local nugget effect. The models are defined as:

Source	Related to	Description	Model
mean	μ	Mean function	$\boldsymbol{X}_1\boldsymbol{\beta}$
std.dev	σ^X	Marginal standard deviation	$\exp(0.5\boldsymbol{X}_2\boldsymbol{\alpha})$
scale	$\boldsymbol{\Sigma}^X$	Local scale	$\exp(\boldsymbol{X}_3\boldsymbol{\theta}_1)$
aniso	$\boldsymbol{\Sigma}^X$	Local geometric anisotropy	$\exp(\boldsymbol{X}_4\boldsymbol{\theta}_2)$
tilt	$\boldsymbol{\Sigma}^X$	(Restricted) local tilt	$\cos(\text{logit}^{-1}(\boldsymbol{X}_5\boldsymbol{\theta}_3))$
smooth	ν^X	Local smoothness	$(\nu_u - \nu_l)/(1 + \exp(-\boldsymbol{X}_6\boldsymbol{\phi})) + \nu_l$
nugget	σ^X_ϵ	Local micro-scale variability	$\exp(\boldsymbol{X}_7\boldsymbol{\zeta})$

where $\boldsymbol{\beta}$, $\boldsymbol{\alpha}$, $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$, $\boldsymbol{\theta}_3$, $\boldsymbol{\phi}$, and $\boldsymbol{\zeta}$ are the parameter vectors of each model, ν_l, and ν_u are the lower and upper bounds limiting the range of variation of the spatially-varying smoothness, and where $\boldsymbol{X}_\ell$ relates to a specific design matrix defined by the specific models for each of the source of nonstationarity.

Lastly, arguments for the `"info"` list argument involve:

- `"lambda"`: (`numeric`) a positive scalar specifying the regularization parameter.
- `"smooth.limits"`: (`numeric vector`) specifying the allowed range of variation for the spatially varying smoothness.
- `"taper"`: (`numeric`) specifying the desired taper function from the spam package (only for "sparse" coco objects).
- `"delta"`: (`numeric`) specifying the taper range/scale (only for "sparse" coco objects).
- `"cat.vars"`: (`integer vector`) index of those variables in `data` that should not be scaled during the optimization (e.g., categorical).

Value

(S4) An S4 object of class coco.

Author(s)

Federico Blasi

References

[1] Furrer, Reinhard, Marc G. Genton, and Douglas Nychka. *"Covariance tapering for interpolation of large spatial datasets."* Journal of Computational and Graphical Statistics 15.3 (2006): 502-523.

See Also

spam::cov.wend1()

Examples

```
## Not run:
locs <- expand.grid(seq(0,1,length.out = 10),
seq(0,1,length.out = 10))

toydata <- data.frame('x' = locs[,1])

set.seed(1)
z <- rnorm(100)

model.list <- list('mean' = 0,
                   'std.dev' = formula( ~ 1),
                   'scale' = formula( ~ 1 + x),
                   'aniso' = 0,
                   'tilt' = 0,
                   'smooth' = 3/2,
                   'nugget' = -Inf)

coco_object <- coco(type = 'dense',
                    data = toydata,
                    locs = as.matrix(locs),
                    z = z,
                    model.list = model.list)

coco_object

## End(Not run)
```

coco-class *An S4 class to store information*

Description

An S4 class to store information

Slots

`type` One of two available types "dense" or "sparse". See description.

`data` A data.frame with covariates information, where colnames(data) matches model.list specification

`locs` a matrix with locs matching data

`z` A vector with response values

`model.list` A list specyfing a model for each aspect of the spatial structure.

`info` a list with information about the coco object

`output` an output from optimparallel output, including as well boundaries information as another element of the list

Author(s)

Federico Blasi

cocoOptim *Optimizer of nonstationary spatial models*

Description

Estimation of the spatial model parameters based on the L-BFGS-B optimizer [1].

Usage

```
cocoOptim(coco.object, boundaries = list(),
ncores = parallel::detectCores(), optim.control, optim.type)
```

Arguments

`coco.object`	(S4) a coco object.
`boundaries`	(`list`) if provided, a list with lower, init, and upper values, as the one provided by getBoundaries. Otherwise, it is computed based on getBoundaries with global lower and upper values -2 and 2.
`ncores`	(`integer`) number of threads for the optimization routine.
`optim.control`	(`list`) list with settings to be passed to the optimParallel function [2].

`optim.type` (`character`) Optimization approach: whether `"mle"` for classical Maximum Likelihood approach, or `"pmle"` to factor out the spatial trend (when handling `"dense"` coco objects), or to factor out the global marginal standard deviation parameter (when considering `"sparse"` coco objects).

Details

Current implementation only allows a single realization for `"pmle"` `optim.type`.

Value

(`S4`) An optimized S4 object of class `coco`.

Author(s)

Federico Blasi

References

[1] Byrd, Richard H., et al. *"A limited memory algorithm for bound constrained optimization."* SIAM Journal on scientific computing 16.5 (1995): 1190-1208.

[2] Gerber, Florian, and Reinhard Furrer. *"optimParallel: An R package providing a parallel version of the L-BFGS-B optimization method."* R Journal 11.1 (2019): 352-358.

See Also

[optimParallel]

Examples

```
## Not run:
model.list <- list('mean' = 0,
                   'std.dev' = formula( ~ 1 + cov_x + cov_y),
                   'scale' = formula( ~ 1 + cov_x + cov_y),
                   'aniso' = 0,
                   'tilt' = 0,
                   'smooth' = 3/2,
                   'nugget' = -Inf)

coco_object <- coco(type = 'dense',
                    data = holes[[1]][1:100,],
                    locs = as.matrix(holes[[1]][1:100,1:2]),
                    z = holes[[1]][1:100,]$z,
                    model.list = model.list)

optim_coco <- cocoOptim(coco_object,
boundaries = getBoundaries(coco_object,
lower.value = -3, 3))

plot(optim_coco)
```

```
print(optim_coco)

getEstims(optim_coco)

## End(Not run)
```

`cocoPredict` *Prediction routines for nonstationary spatial models*

Description

Computes the point predictions and standard errors based on conditional Gaussian distributions.

Usage

```
cocoPredict(coco.object, newdataset, newlocs, type = 'mean', ...)
```

Arguments

`coco.object`	(`S4`) a fitted coco object.
`newdataset`	(`data.frame`) a data.frame containing covariates present in model.list at prediction locations.
`newlocs`	(`matrix`) a matrix with locations related to prediction locations, matching indexing of `newdataset`.
`type`	(`character`) whether `"mean"` or `"pred"`, which gives a point prediction for the former, as well as of point prediction and standard errors for the latter.
`...`	(`character`) when coco.object has multiple realizations, specifying `"index.pred"` specifying which column of `coco.object@z` should be used to perform predictions.

Value

(`list`) a list with the conditional mean, splitted into the systematic large-scale variability `trend`, and due to stochastic `mean`, as well as standard errors `"sd.pred"` if `"pred"` is specified.

Author(s)

Federico Blasi

Examples

```
## Not run:

model.list <- list('mean' = 0,
'std.dev' = formula( ~ 1 + cov_x + cov_y),
'scale' = formula( ~ 1 + cov_x + cov_y),
'aniso' = 0,
'tilt' = 0,
'smooth' = 3/2,
'nugget' = -Inf)

coco_object <- coco(type = 'dense',
data = holes[[1]][1:100, ],
locs = as.matrix(holes[[1]][1:100, 1:2]),
z = holes[[1]][1:100, ]$z,
model.list = model.list)

optim_coco <- cocoOptim(coco_object,
boundaries = getBoundaries(coco_object,
lower.value = -3, 3))

coco_preds <- cocoPredict(optim_coco, newdataset = holes[[2]],
newlocs = as.matrix(holes[[2]][, 1:2]),
type = "pred")

coco_preds

par(mfrow = c(1, 2))

fields::quilt.plot(main = "mean", holes[[2]][, 1:2],
coco_preds$mean, xlim = c(-1, 1), ylim = c(-1, 1))
fields::quilt.plot(main = "se", holes[[2]][, 1:2],
coco_preds$sd.pred, xlim = c(-1, 1), ylim = c(-1, 1))

# Re-do it without considering cov_x and cov_y in the std.dev and scale and compare.


## End(Not run)
```

---

cocoSim *Marginal and conditional simulation of nonstationary Gaussian process*

Description

draw realizations of nonstationary Gaussian processes with covariate-based covariance functions.

Usage

```
cocoSim(coco.object, pars, n, seed, standardize,
type = 'classic', sim.type = NULL, cond.info = NULL)
```

Arguments

coco.object	(S4) a coco object.
pars	(numeric vector) a vector of parameters values related to model.list.
n	(integer) number of realizations to simulate.
seed	(integer or NULL) seed number. default set to NULL.
standardize	(TRUE/FALSE) logical argument describing whether provided covariates should be standardize (TRUE) or not (FALSE). By default set to TRUE.
type	(character) whether parameters are related to a classical parameterization ('classic') or a difference parameterization 'diff'. Default set to 'classic'. For 'sparse' coco objects, only 'diff' is available.
sim.type	(character) if set 'cond' then a conditional simulation takes place.
cond.info	(list) a list containing added information to perform conditional simulation.

Details

'cond' sim.type requires specifying in 'cond.info' a list with 'newdataset' a data.frame containing covariates present in model.list at simulation locations, and 'newlocs' a matrix with locations related to the simulation locations, matching indexing of 'newdataset'.

type = 'classic' assumes a simpler parameterization for the covariance function, assuming log-parameterizations for 'std.dev', 'scale', and 'smooth'.

Value

(matrix) a matrix n x dim(data)[1].

Author(s)

Federico Blasi

See Also

coco()

Examples

```
## Not run:

model.list <- list('mean' = 0,
                   'std.dev' = formula( ~ 1 + cov_x + cov_y),
                   'scale' = formula( ~ 1 + cov_x + cov_y),
                   'aniso' = 0,
                   'tilt' = 0,
```

```
                    'smooth' = 0.5,
                    'nugget' = -Inf)

coco_object <- coco(type = 'dense',
                    data = holes[[1]][1:1000,],
                    locs = as.matrix(holes[[1]][1:1000,1:2]),
                    z = holes[[1]][1:1000,]$z,
                    model.list = model.list)

coco_sim <- cocoSim(coco.object = coco_object,
            pars = c(0,0.25,0.25,  # pars related to std.dev
            log(0.25),1,-1),       # pars related to scale
            n = 1,
            standardize = TRUE)

fields::quilt.plot(coco_object@locs,coco_sim)

## End(Not run)
```

`cov_rns` *Dense covariance function (difference parameterization)*

Description

Dense covariance function (difference parameterization)

Usage

```
cov_rns(theta, locs, x_covariates, smooth_limits)
```

Arguments

`theta`	vector of parameters
`locs`	a matrix with locations
`x_covariates`	design data.frame
`smooth_limits`	smooth limits

Value

dense covariance matrix

cov_rns_classic *Dense covariance function (classic parameterization)*

Description

Dense covariance function (classic parameterization)

Usage

```
cov_rns_classic(theta, locs, x_covariates)
```

Arguments

theta	vector of parameters
locs	a matrix with locations
x_covariates	design data.frame

Value

dense covariance matrix with classic parameterization

cov_rns_pred *Dense covariance function*

Description

Dense covariance function

Usage

```
cov_rns_pred(
  theta,
  locs,
  locs_pred,
  x_covariates,
  x_covariates_pred,
  smooth_limits
)
```

Arguments

theta	vector of parameters
locs	a matrix with locations
locs_pred	a matrix with prediction locations
x_covariates	design data.frame
x_covariates_pred	design data.frame at prediction locations
smooth_limits	smooth limits

Value

dense covariance matrix

---

cov_rns_taper *Sparse covariance function*

Description

Sparse covariance function

Usage

```
cov_rns_taper(
  theta,
  locs,
  x_covariates,
  colindices,
  rowpointers,
  smooth_limits
)
```

Arguments

theta	vector of parameters
locs	a matrix with locations
x_covariates	design data.frame
colindices	from spam object
rowpointers	from spam object
smooth_limits	smooth limits

Value

sparse covariance matrix between locs and pred_locs

`cov_rns_taper_pred` *Sparse covariance function*

Description

Sparse covariance function

Usage

```
cov_rns_taper_pred(
  theta,
  locs,
  locs_pred,
  x_covariates,
  x_covariates_pred,
  colindices,
  rowpointers,
  smooth_limits
)
```

Arguments

`theta`	vector of parameters
`locs`	a matrix with locations
`locs_pred`	a matrix with prediction locations
`x_covariates`	design data.frame
`x_covariates_pred`	design data.frame at prediction locations
`colindices`	from spam object
`rowpointers`	from spam object
`smooth_limits`	smooth limits

Value

sparse covariance matrix at locs

getAIC *Retrieve AIC*

Description

Retrieve the Akaike information criterion from a fitted coco object.

Usage

```
getAIC(coco.object)
```

Arguments

coco.object	(S4) a fitted coco S4 object.

Value

(numeric) the associated AIC value

Author(s)

Federico Blasi

getBIC *Retrieve BIC*

Description

Retrieve BIC from a fitted coco object.

Usage

```
getBIC(coco.object)
```

Arguments

coco.object	(S4) a fitted coco S4 object.

Value

(numeric) the associated BIC value

Author(s)

Federico Blasi

getBoundaries *Simple build of boundaries*

Description

provides a generic set of upper and lower bounds for the L-BFGS-B routine

Usage

```
getBoundaries(x, lower.value, upper.value)
```

Arguments

x	(S4) or (list) a coco.object or a par.pos list (as output from getDesignMatrix)
lower.value	(numeric vector) if provided, provides a vector filled with values lower.value.
upper.value	(numeric vector) if provided, provides a vector filled with values upper.value.

Value

(list) a list with boundaries and simple init values for the optim L-BFGS-B routine

Author(s)

Federico Blasi

getBoundariesV2 *Simple build of boundaries (v2)*

Description

provides a generic set of upper and lower bounds for the L-BFGS-B routine

Usage

```
getBoundariesV2(coco.object, mean.limits, std.dev.limits,
scale.limits, aniso.limits, tilt.limits, smooth.limits, nugget.limits)
```

Arguments

coco.object	(S4) a coco object.
mean.limits	(numeric vector) a vector of c(lower,init,upper) values for the associated param.
std.dev.limits	(numeric vector) a vector of c(lower,init,upper) values for the associated param.
scale.limits	(numeric vector) a vector of c(lower,init,upper) values for the associated param.
aniso.limits	(numeric vector) a vector of c(lower,init,upper) values for the associated param.
tilt.limits	(numeric vector) a vector of c(lower,init,upper) values for the associated param.
smooth.limits	(numeric vector) a vector of c(lower,init,upper) values for the associated param.
nugget.limits	(numeric vector) a vector of c(lower,init,upper) values for the associated param.

Value

(list) a list with boundaries for the optim L-BFGS-B routine

Author(s)

Federico Blasi

---

getBoundariesV3 *Simple build of boundaries (v3)*

Description

provides a generic set of upper and lower bounds for the L-BFGS-B routine

Usage

```
getBoundariesV3(coco.object, mean.limits, global.lower,
std.dev.max.effects,
scale.max.effects, aniso.max.effects, tilt.max.effects,
smooth.max.effects, nugget.max.effects)
```

Arguments

coco.object	(S4) a coco object.
mean.limits	(numeric vector) a vector of c(lower,init,upper) values for the associated param.
global.lower	(numeric vector) a vector of c(lower,init,upper) values for the associated param.
std.dev.max.effects	(numeric vector) a vector of c(lower,init,upper) values for the associated param.
scale.max.effects	(numeric vector) a vector of c(lower,init,upper) values for the associated param.
aniso.max.effects	(numeric vector) a vector of c(lower,init,upper) values for the associated param.
tilt.max.effects	(numeric vector) a vector of c(lower,init,upper) values for the associated param.
smooth.max.effects	(numeric vector) a vector of c(lower,init,upper) values for the associated param.
nugget.max.effects	(numeric vector) a vector of c(lower,init,upper) values for the associated param.

Value

(list) a list with boundaries for the optim L-BFGS-B routine

Author(s)

Federico Blasi

getCIs *Compute Confidence Intervals for a coco object*

Description

Compute confidence intervals for a (fitted) coco object.

Usage

```
getCIs(coco.object, inv.hess, alpha = 0.05)
```

Arguments

coco.object	(S4) a fitted coco S4 object.
inv.hess	(matrix) Inverse of the Hessian.
alpha	(numeric) confidence level.

Value

(numeric matrix) a matrix with confidence intervals for each parameter in the model

Author(s)

Federico Blasi

getCondNumber *Condition number for (fitted) coco objects*

Description

Computes the condition number of the associated correlation matrix of the fitted coco object.

Usage

```
getCondNumber(coco.object)
```

Arguments

coco.object	(S4) a fitted coco S4 object.

Value

(numeric) the condition number.

Author(s)

Federico Blasi

getCovMatrix *Covariance matrix for "coco" class*

Description

Compute the covariance matrix of `coco.object`.

Usage

```
getCovMatrix(coco.object, type = 'global', index = NULL)
```

Arguments

`coco.object`	(`S4`) a fitted `coco()` object.
`type`	(`character`) whether `'global'` to retrieve the regular covariance matrix, or `'local'` to retrieve global covariance. based on the local aspects of a specific location (not implemented yet).
`index`	(`integer`) index to perform local covariance matrix (not implemented yet).

Value

(`matrix`) a n x n covariance matrix.

Author(s)

Federico Blasi

getCRPS *Based on a set of predictions retrieves the Logrank*

Description

Retrieves the Continuous Ranked Probability Score (CRPS) [1].

Usage

```
getCRPS(z.pred, mean.pred, sd.pred)
```

Arguments

`z.pred`	(`numeric vector`).
`mean.pred`	(`numeric vector`).
`sd.pred`	(`numeric vector`).

Value

(numeric vector) retrieves CRPS.

Author(s)

Federico Blasi

References

[1] Gneiting, Tilmann, and Adrian E. Raftery. *"Strictly proper scoring rules, prediction, and estimation."* Journal of the American statistical Association 102.477 (2007): 359-378.

---

getDesignMatrix *Create an efficient design matrix based on a list of aspect models*

Description

Creates a unique design matrix based on model specification for each of the different potentially spatially varying aspects.

Usage

```
getDesignMatrix(model.list, data)
```

Arguments

model.list	(list) a list of formulas, one for each source of nonstationarity, specifying the models.
data	(data.frame) a data.frame.

Value

(list) a list with two elements: a design matrix of dimension (n x p), and a par.pos object, indexing columns of the design matrix to each of the spatially-varying functions.

Author(s)

Federico Blasi

getEstims *Retrieve estimates from a fitted coco object*

Description

Retrieve estimates from a fitted coco object.

Usage

```
getEstims(coco.object)
```

Arguments

coco.object	(S4) a fitted coco S4 object.

Value

(list) a list with the estimates parameters for the different aspects

Author(s)

Federico Blasi

getHessian *getHessian*

Description

returns the approximate (observed) Hesian (inverse of Fisher Information Matrix)

Usage

```
getHessian(coco.object, ncores = parallel::detectCores() - 1,
eps = .Machine$double.eps^(1/4))
```

Arguments

coco.object	(S4) a fitted coco object.
ncores	(integer) number of cores used for the computation.
eps	(numeric) ...

Value

(numeric matrix) a symmetric matrix pxp of the approximated (observed) Hessian

Author(s)

Federico Blasi

getLoglik *Retrieve the loglikelihood value*

Description

Retrieve the loglikelihood value from a fitted coco object.

Usage

```
getLoglik(coco.object)
```

Arguments

coco.object	(S4) a fitted coco S4 object.

Value

(numeric) wrap for value from a OptimParallel object

Author(s)

Federico Blasi

getLogScore *Computes the Log-Score*

Description

Retrieves the Log-Score [1].

Usage

```
getLogScore(z.pred, mean.pred, sd.pred)
```

Arguments

z.pred	(numeric vector).
mean.pred	(numeric vector).
sd.pred	(numeric vector).

Value

(numeric vector) retrieves Log-Score.

Author(s)

Federico Blasi

References

[1] Gneiting, Tilmann, and Adrian E. Raftery. *"Strictly proper scoring rules, prediction, and estimation."* Journal of the American statistical Association 102.477 (2007): 359-378.

---

getModelLists *Builds the necessary input for building covariance matrices*

Description

Returns a list of parameter vectors for each of the aspects.

Usage

```
getModelLists(theta, par.pos, type = 'diff')
```

Arguments

theta	(numeric vector) a vector of length p, where p is the number of parameters for each of the models.
par.pos	(list) a list detailing in which position of each aspect the elements of theta should be placed. Expected to be par.pos output of getDesignMatrix.
type	(character) whether parameters are related to a classical parameterization ('classic') or a difference parameterization 'diff' . Default set to 'diff'.

Value

(list) a list of different spatial aspects and mean required for the cov.rns functions

Author(s)

Federico Blasi

getModHess *Retrieves the modified inverse of the hessian*

Description

Based on the inverse of the Hessian (based on the difference parameterization for the std.dev and scale parameters), retrieves the modified inverse of the hessian (i.e. std.dev and scale).

Usage

```
getModHess(coco.object, inv.hess)
```

Arguments

coco.object	(S4) a fitted coco S4 object.
inv.hess	(matrix) Inverse of the Hessian.

Value

(numeric matrix) the modified inverse of the hessian matrix

Author(s)

Federico Blasi

GetNeg2loglikelihood *GetNeg2loglikelihood*

Description

compute the negative 2 log likelihood based on theta

Usage

```
GetNeg2loglikelihood(theta, par.pos, locs, x_covariates,
smooth.limits, z, n, lambda)
```

Arguments

theta	(numeric vector) a vector with parameters values.
par.pos	(list) par.pos list.
locs	(matrix) spatial location matrix.
x_covariates	(data.frame) design matrix.
smooth.limits	(numeric vector) smooth.limits.
z	(numeric vector) a vector of observed values.
n	(integer) dim(z)[1].
lambda	(numeric) regularization parameter.

Value

value

Author(s)

Federico Blasi

`GetNeg2loglikelihoodProfile` *GetNeg2loglikelihoodProfile*

Description

compute the negative 2 log likelihood based on theta

Usage

```
GetNeg2loglikelihoodProfile(theta, par.pos, locs, x_covariates,
smooth.limits, z, n, x_betas,lambda)
```

Arguments

`theta`	(`numeric vector`) a vector with parameters values.
`par.pos`	(`list`) par.pos list.
`locs`	(`matrix`) spatial location matrix.
`x_covariates`	(`data.frame`) design matrix.
`smooth.limits`	(`numeric vector`) smooth.limits.
`z`	(`numeric vector`) a vector of observed values.
`n`	(`integer`) dim(z)[1].
`x_betas`	(`matrix`) or (`data.frame`) design matrix for the trend.
`lambda`	(`numeric`) regularization parameter.

Value

value

Author(s)

Federico Blasi

GetNeg2loglikelihoodTaper *GetNeg2loglikelihoodTaper*

Description

compute the negative 2 log likelihood based on theta

Usage

```
GetNeg2loglikelihoodTaper(theta, par.pos, ref_taper, locs,
x_covariates, smooth.limits, cholS, z, n, lambda)
```

Arguments

`theta`	(`numeric vector`) a vector with parameters values.
`par.pos`	(`list`) par.pos list from getDesignMatrix.
`ref_taper`	(`S4`) spam object based on a compact-supported covariance function.
`locs`	(`matrix`) spatial location matrix.
`x_covariates`	(`data.frame`) design matrix.
`smooth.limits`	(`numeric vector`) smooth.limits.
`cholS`	(`S4`) Cholesky object from spam.
`z`	(`numeric vector`) a vector of observed values.
`n`	(`numeric`) dim(z)[1].
`lambda`	(`numeric`) regularization parameter.

Value

value

Author(s)

Federico Blasi

GetNeg2loglikelihoodTaperProfile
GetNeg2loglikelihoodTaperProfile

Description

compute the negative 2 log likelihood based on theta

Usage

```
GetNeg2loglikelihoodTaperProfile(theta, par.pos, ref_taper,
locs, x_covariates, smooth.limits, cholS, z, n, lambda)
```

Arguments

theta	(numeric vector) a vector with parameters values.
par.pos	(list) par.pos list.
ref_taper	(S4) spam object based on a taper based covariance function.
locs	(matrix) spatial location matrix.
x_covariates	(data.frame) design matrix.
smooth.limits	(numeric vector) smooth.limits.
cholS	(S4) Cholesky object from spam.
z	(numeric vector) a vector of observed values.
n	(integer) dim(z)[1].
lambda	(numeric) regularization parameter.

Value

value

Author(s)

Federico Blasi

getPen *Returns the penalization term*

Description

Returns the penalization term.

Usage

```
getPen(n, lambda, theta_list, smooth.limits)
```

Arguments

n	(integer).
lambda	(numeric).
theta_list	(list).
smooth.limits	(numeric vector).

Value

(numeric) retrieves penalization term.

Author(s)

Federico Blasi

getScale *Fast and simple standardization for the design matrix.*

Description

Centers and scale the design matrix.

Usage

```
getScale(x, mean.vector = NULL, sd.vector = NULL)
```

Arguments

x	(S4) or (matrix) a coco object, or a n x p matrix with covariate information to introduce, where the first column is a column of ones.
mean.vector	(numeric vector) if provided, it centers covariates based on this information.
sd.vector	(numeric vector) if provided, it scales covariates based on this information.

Value

(list) a list with a scaled design matrix of dimension n x (p+1), and a set of mean and sd vectors employed to scale the matrix

Author(s)

Federico Blasi

---

getSpatEffects *Computes the spatially-varying functions from a coco object*

Description

Evaluates the spatially-varying functions of the nonstationary spatial structure.

Usage

```
getSpatEffects(coco.object)
```

Arguments

coco.object	(S4) a fitted coco S4 object.

Value

(list) a list with the different estimated surfaces.

Author(s)

Federico Blasi

---

getTrend *Computes the spatial trend of a (fitted) coco object*

Description

Compute the trend of the (fitted) coco object.

Usage

```
getTrend(coco.object)
```

Arguments

coco.object	(S4) a fitted coco S4 object.

Value

(numeric vector) a vector with the adjusted trend.

Author(s)

Federico Blasi

---

holes *Holes Data Set*

Description

The synthetic "holes" provides a set of training and test data.frame of a Gaussian process realization with a (inherently dense) nonstationary covariance function. Four holes are present in the training dataset, and the task is to predict them.

Usage

```
holes
```

Format

A list with training and test data.frame with rows and variables:

x first spatial coordinate

y second spatial coordinate

cox_x first spatial characteristic

cov_y second spatial characteristic

z response variable

Source

Source of the data

Examples

```
data(holes)
```

is.formula *check whether an object belongs to a formula class*

Description

check whether an object belongs to a formula class

Usage

```
is.formula(x)
```

Arguments

x	an R object

Value

TRUE/FALSE

Author(s)

Federico Blasi

plot,coco,missing-method *Plot Method for Coco Class*

Description

This method plots objects of class coco.

Usage

```
## S4 method for signature 'coco,missing'
plot(x, y, ..., type = NULL, index = NULL, factr = 0.1, plot.control = NULL)
```

Arguments

x	An object of class coco.
y	Not used.
...	Additional arguments passed to the plot function. when type "ellipse" , delta of nearest.dist must be specified.
type	The type of plot. NULL or "ellipse" for drawing ellipse of the convolution kernels.
index	For plotting local correlation plots.
factr	Factor rate for size of ellipses.
plot.control	Additional plot control parameters.

Value

A plot is created.

`plotOptimInfo` *Plot log info detailed*

Description

plot output of optim

Usage

```
plotOptimInfo(coco.object, ...)
```

Arguments

`coco.object`	an optimized coco.object
`...`	arguments for par()

Value

Outputs a sequence of plots detailing parameters during the optimization routine

Author(s)

Federico Blasi

`print` *Print Method for Coco Class*

Description

This method prints objects of class 'coco'.

Usage

```
## S4 method for signature 'coco'
print(x, inv.hess = NULL, ...)
```

Arguments

`x`	An object of class 'coco'.
`inv.hess`	inverse of the approximated hessian matrix (getHessian)
`...`	Additional arguments to be passed to plot.

Value

print the coco object

Author(s)

Federico Blasi

---

show *Show Method for Coco Class*

Description

This method show objects of class 'coco'.

Usage

```
## S4 method for signature 'coco'
show(object)
```

Arguments

object	An object of class 'coco'.

Value

A plot is created.

Author(s)

Federico Blasi

---

stripes *Stripes Data Set*

Description

The synthetic "stripes" provides a set of training and test data.frame of a Gaussian process realization with a (inherently sparse) nonstationary covariance function. Several stripes are present in the training dataset, and the task is to predict them.

Usage

```
stripes
```

Format

A list with training and test data.frame with rows and variables:

x first spatial coordinate

y second spatial coordinate

cox_x first spatial characteristic

cov_y second spatial characteristic

cov_xy third spatial characteristic

z response variable

Source

Source of the data

Examples

```
data(stripes)
```

Index