

# Package: cocons (via r-universe)

September 18, 2024

**Type** Package

**Title** Covariate-Based Covariance Functions for Nonstationary Spatial Modeling

**Version** 0.1.3

**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](mailto:federico.blasi@math.uzh.ch)>

**Description** Estimation and prediction of nonstationary Gaussian process with modular covariate-based covariance functions. An induced compact-supported nonstationary covariance function is provided to speed up computations when handling densely sampled domains.

**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>

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.2

**VignetteBuilder** knitr

**Repository** <https://blasif.r-universe.dev>

**RemoteUrl** <https://github.com/blasif/cocons>

**RemoteRef** HEAD

**RemoteSha** a3ec7aeb637eac52b7e0e9be14638431d49bf4e

## Contents

cocons-package . . . . .	3
coco . . . . .	3
coco-class . . . . .	6
cocoOptim . . . . .	6
cocoPredict . . . . .	8
cocoSim . . . . .	9
cov_rns . . . . .	11
cov_rns_classic . . . . .	11
cov_rns_pred . . . . .	12
cov_rns_taper . . . . .	13
cov_rns_taper_pred . . . . .	13
getAIC . . . . .	14
getBIC . . . . .	15
getBoundaries . . . . .	15
getBoundariesV2 . . . . .	16
getBoundariesV3 . . . . .	17
getCIs . . . . .	18
getCondNumber . . . . .	18
getCovMatrix . . . . .	19
getCRPS . . . . .	19
getDesignMatrix . . . . .	20
getEstims . . . . .	21
getHessian . . . . .	21
getLoglik . . . . .	22
getLogScore . . . . .	22
getModelLists . . . . .	23
getModHess . . . . .	24
GetNeg2loglikelihood . . . . .	24
GetNeg2loglikelihoodProfile . . . . .	25
GetNeg2loglikelihoodTaper . . . . .	26
GetNeg2loglikelihoodTaperProfile . . . . .	27
getScale . . . . .	28
getSpatEffects . . . . .	28
getTrend . . . . .	29
holes . . . . .	29
is.formula . . . . .	30
plot,coco,missing-method . . . . .	30
plotOptimInfo . . . . .	31
show . . . . .	32
stripes . . . . .	32
summary . . . . .	33

## Index

34

---

`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. An induced compact-supported nonstationary covariance function is provided to speed up computations when handling densely sampled domains. 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 and in `help(coco)`.

## 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

<code>type</code>	(character) One of two available types "dense" or "sparse". See description.
<code>data</code>	(data.frame) A data.frame with covariates information, where <code>colnames(data)</code> matches <code>model.list</code> specification.
<code>locs</code>	(matrix) A matrix with spatial locations.
<code>z</code>	(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.
<code>model.list</code>	(list) A list specifying a model for each aspect of the spatial structure.
<code>info</code>	(list or NULL) A list specifying characteristics of the coco object.
<code>output</code>	(list or NULL) Empty or the resulting object from running <code>optimParallel</code> , adding to this a list with boundary information (check <code>getBoundaries</code> 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
<i>mean</i>	$\mu$	Mean function	$\mathbf{X}_1\beta$
<i>std.dev</i>	$\sigma^X$	Marginal standard deviation	$\exp(0.5\mathbf{X}_2\alpha)$
<i>scale</i>	$\Sigma^X$	Local scale	$\exp(\mathbf{X}_3\theta_1)$
<i>aniso</i>	$\Sigma^X$	Local geometric anisotropy	$\exp(\mathbf{X}_4\theta_2)$
<i>tilt</i>	$\Sigma^X$	(Restricted) local tilt	$\cos(\text{logit}^{-1}(\mathbf{X}_5\theta_3))$
<i>smooth</i>	$\nu^X$	Local smoothness	$(\nu_u - \nu_l)/(1 + \exp(-\mathbf{X}_6\phi)) + \nu_l$
<i>nugget</i>	$\sigma_\epsilon^X$	Local micro-scale variability	$\exp(\mathbf{X}_7\zeta)$

where  $\beta$ ,  $\alpha$ ,  $\theta_1$ ,  $\theta_2$ ,  $\theta_3$ ,  $\phi$ , and  $\zeta$  are the parameter vectors of each model,  $\nu_l$ , and  $\nu_u$  are the lower and upper bounds limiting the range of variation of the spatially-varying smoothness, and where  $\mathbf{X}_\ell$  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).
- "skip.scale": (integer vector) index of those variables in data that should not be scaled during the optimization.

### 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	<i>An S4 class to store information</i>
------------	---

---

### 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 specifying 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	<i>Optimizer for nonstationary spatial models</i>
-----------	---

---

### Description

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

### Usage

```
cocoOptim(coco.object, boundaries = list(),
ncores = "auto", optim.type, optim.control)
```

### 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` (character or integer) number of threads for the optimization routine. If "auto" then it is set based on optimal number of threads (when available) or a fraction of it.

`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).

`optim.control` (list) list with settings to be passed to the `optimParallel` function [2].

### 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)
```

```
summary(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 <a href="#">coco</a> 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))

## 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 <a href="#">coco</a> object.
pars	(numeric vector) a vector of parameters values related to <code>model.list</code> .
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 standardized (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  $n \times \dim(\text{data})[1]$ .

**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\_rms                      *Dense covariance function (difference parameterization)*

---

### Description

Dense covariance function (difference parameterization)

### Usage

```
cov_rms(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\_rms\_classic              *Dense covariance function (classic parameterization)*

---

### Description

Dense covariance function (classic parameterization)

### Usage

```
cov_rms_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	<i>Dense covariance function</i>
--------------	----------------------------------

---

**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	<i>Sparse covariance function</i>
---------------	-----------------------------------

---

**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	<i>Sparse covariance function</i>
--------------------	-----------------------------------

---

**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	<i>Retrieve AIC</i>
--------	---------------------

---

**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	<i>Simple build of boundaries (v2)</i>
-----------------	--

---

**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	<i>Covariance matrix for "coco" class</i>
--------------	---

---

**Description**

Compute the covariance matrix of `coco.object`.

**Usage**

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

**Arguments**

<code>coco.object</code>	(S4) a fitted <code>coco()</code> object.
<code>type</code>	(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).
<code>index</code>	(integer) index to perform local covariance matrix (not implemented yet).

**Value**

(matrix) a  $n \times n$  covariance matrix.

**Author(s)**

Federico Blasi

---

getCRPS	<i>Based on a set of predictions computes the Continuous Ranked Probability Score</i>
---------	---

---

**Description**

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

**Usage**

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

**Arguments**

<code>z.pred</code>	(numeric vector).
<code>mean.pred</code>	(numeric vector).
<code>sd.pred</code>	(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	<i>Retrieve estimates from a fitted coco object</i>
-----------	---

---

**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	<i>getHessian</i>
------------	-------------------

---

**Description**

returns the approximate (observed) Hessian (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	<i>Retrieve the loglikelihood value</i>
-----------	---

---

**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	<i>Based on a set of predictions computes the Log-Score</i>
-------------	---

---

**Description**

Computes 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	<i>Builds the necessary input for building covariance matrices</i>
---------------	--

---

**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 <a href="#">getDesignMatrix</a> .
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.rms functions

**Author(s)**

Federico Blasi

---

getModHess	<i>Retrieves the modified inverse of the hessian</i>
------------	--

---

**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	<i>GetNeg2loglikelihood</i>
----------------------	-----------------------------

---

**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 <a href="#">getDesignMatrix</a> .
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

---

getScale	<i>Fast and simple standardization for the design matrix.</i>
----------	---

---

**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	<i>Evaluates the spatially-varying functions from a coco object at locs</i>
----------------	---

---

**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	<i>Computes the spatial trend of a (fitted) coco object</i>
----------	---

---

**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	<i>Holes Data Set</i>
-------	-----------------------

---

**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, delta = NULL)
```

**Arguments**

x	An object of class coco.
y	Not used.
...	Additional arguments passed to <a href="#">quilt.plot</a> .
type	(character or NULL) The type of plot. NULL or "ellipse" for drawing ellipse of the convolution kernels.
index	(integer vector) For plotting local correlation plots.
factr	(numeric) Factor rate for size of ellipses.
delta	(numeric) when type "ellipse", "delta" of nearest.dist must be specified.

**Value**

Several plots are created.

**Author(s)**

Federico Blasi

---

plotOptimInfo	<i>Plot log info detailed</i>
---------------	-------------------------------

---

**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

---

show	<i>Show Method for Coco Class</i>
------	-----------------------------------

---

**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	<i>Stripes Data Set</i>
---------	-------------------------

---

**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)
```

---

summary

*Summary Method for Coco Class*

---

**Description**

method summary for objects of class 'coco'.

**Usage**

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

**Arguments**

object	(S4) An object of class 'coco'.
inv.hess	(numeric matrix or NULL) inverse of the approximated hessian matrix (getHessian)

**Value**

summary the coco object

**Author(s)**

Federico Blasi

# Index

## \* datasets

- holes, [29](#)
- stripes, [32](#)

coco, [3](#), [6](#), [8](#), [10](#)  
coco(), [19](#)  
coco-class, [6](#)  
cocons (cocons-package), [3](#)  
cocons-package, [3](#)  
cocoOptim, [6](#)  
cocoPredict, [8](#)  
cocoSim, [9](#)  
cov\_rns, [11](#)  
cov\_rns\_classic, [11](#)  
cov\_rns\_pred, [12](#)  
cov\_rns\_taper, [13](#)  
cov\_rns\_taper\_pred, [13](#)

getAIC, [14](#)  
getBIC, [15](#)  
getBoundaries, [4](#), [6](#), [15](#)  
getBoundariesV2, [16](#)  
getBoundariesV3, [17](#)  
getCIs, [18](#)  
getCondNumber, [18](#)  
getCovMatrix, [19](#)  
getCRPS, [19](#)  
getDesignMatrix, [15](#), [20](#), [23](#), [26](#)  
getEstims, [21](#)  
getHessian, [21](#)  
getLoglik, [22](#)  
getLogScore, [22](#)  
getModelLists, [23](#)  
getModHess, [24](#)  
GetNeg2loglikelihood, [24](#)  
GetNeg2loglikelihoodProfile, [25](#)  
GetNeg2loglikelihoodTaper, [26](#)  
GetNeg2loglikelihoodTaperProfile, [27](#)  
getScale, [28](#)  
getSpatEffects, [28](#)

getTrend, [29](#)

holes, [29](#)

is.formula, [30](#)

optimParallel, [4](#), [7](#)

plot, coco, missing-method, [30](#)  
plot, coco-method  
(plot, coco, missing-method), [30](#)  
plotOptimInfo, [31](#)

quilt.plot, [31](#)

show, [32](#)  
show, coco-method (show), [32](#)  
spam::cov.wend1(), [5](#)  
stripes, [32](#)  
summary, [33](#)  
summary, coco-method (summary), [33](#)