Package 'cocons'

August 1, 2024

Type Package

Title Covariate-Based Covariance Functions for Nonstationary Spatial Modeling

Version 0.1.1

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 7.2.3

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2024-08-01 08:50:06 UTC

Contents

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

Description

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

Author(s)

Federico Blasi [aut, cre], <federico.blasi@math.uzh.ch>, Reinhard Furrer [ctb]

---

coco *coco Class*

Description

Creates a coco S4 object.

Usage

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

Arguments

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 matrix of n x r response realizations, one realization per column. When considering only one realization, a vector can also be provided.
model.list	A list specifying a model for each aspect of the spatial structure.
info	A list specifying characteristics of the coco object.
output	Empty or an output from optimparallel output, including as well boundaries.

Details

This S4 class is the centerpiece of the cocons package. Two types of coco objects can be created. One is a "dense" type, meaning that the associated nonstationary covariance functions are dense, or a "sparse" object, which, in combination with the Tapering approach, induce zeroes in the covariance matrix to make it sparse and to unlock a set of efficient algorithms to speed up estimation and prediction routines.

Another important component of the coco S4 class is the model.list specification, involving different formulas provided as a list, where each of them specifies a source of nonstationarity, based on covariates. 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.

Lastly, arguments for the info list argument involve:

- 'lambda': a positive scalar specifying the regularization parameter.
- 'smooth_limits': specifying the allowed range of variation for the spatially varying smoothness.

- 'taper': specifying the desired taper function from the spam package (only for 'sparse' coco objects).
- 'delta': specifying the taper range/scale (only for 'sparse' coco objects).
- 'cat.vars': index of those variables in the data object which are categorical or should not be scaled during the optimization.

Value

Creates a coco object.

Author(s)

Federico Blasi

See Also

`spam::cov.wend1()`

`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 for object class coco*

Description

Optimizer based on multi-thread Optimparallel L-BFGS-B optimizers.

Usage

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

Arguments

`coco.object`	a coco object. See ?coco()
`boundaries`	if provided, a list with lower, init, and upper values. if not is computed based on generic fast_init_boundaries()
`ncores`	number of cores to be used for the optimization routine.
`optim.control`	list with settings to be passed to the optimparallel function
`optim.type`	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 implementations only allow single realizations for 'pmle' optim.type.

Value

a fitted coco object

Author(s)

Federico Blasi

cocoPredict *Prediction for object class coco*

Description

Prediction for a fitted coco object.

Usage

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

Arguments

coco.object	a fitted coco object
newdataset	a data.frame including covariates present in model.list at prediction locations
newlocs	a matrix with locations related to prediction locations, matching indexing of newdataset
type	whether "mean" or "pred", which gives a point prediction for the former, as well as a combination of point prediction as well as prediction uncertainty for the latter
...	when coco.object has multiple realizations, specifying 'index.pred' specifying which column of coco.object\@z should be used to perform predictions

Value

a list with trend, and mean predictions and uncertainty quantification, if 'pred' is specified.

Author(s)

Federico Blasi

cocoSim *Marginal and conditional simulation of Gaussian processes with non-stationary covariance function*

Description

simulates a Gaussian process with nonstationary covariance function from an coco object.

Usage

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

Arguments

coco.object	a vector of length p, where p is the number of parameters for
pars	a vector of length p, where p is the number of parameters for each of the models
n	number of realizations to simulate
seed	seed number. default set to NULL.
standardize	logical argument describing whether provided covariates should be standardize (TRUE) or not (FALSE). By default set to TRUE
type	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	if set 'cond' then a conditional simulation takes place.
cond.info	a list containing information to perform a conditional simulation.

Value

a matrix n x dim(data)[1]

Author(s)

Federico Blasi

---

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_optimized_predict_range *Sparse covariance function*

Description

Sparse covariance function

Usage

```
cov_rns_taper_optimized_predict_range(
  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

`cov_rns_taper_optimized_range` *Sparse covariance function*

Description

Sparse covariance function

Usage

```
cov_rns_taper_optimized_range(
  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

getAIC *Retrieve AIC*

Description

Retrieve the Akaike information criterion from a fitted coco object

Usage

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

Arguments

coco.object	a fitted coco object.

Value

a list with 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	a fitted coco object.

Value

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

Value

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

Value

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

Value

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

Author(s)

Federico Blasi

`getCIs` *Compute Confidence Intervals for an coco object*

Description

Compute confidence intervals for a (fitted) coco object

Usage

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

Arguments

`coco.object`	a coco class (fitted) object
`inv.hess`	Inverse of the Hessian
`alpha`	confidence level

Value

a matrix with confidence intervals for each parameter in the model

Author(s)

Federico Blasi

`getCondNumber` *Computes the condition number of the associated correlation matrix of the fitted coco object*

Description

Compute the trend of the (fitted) coco object

Usage

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

Arguments

`coco.object`	a coco class (fitted) object

Value

the condition number

Author(s)

Federico Blasi

getCovMatrix *Covariance matrix from a fitted coco object*

Description

Retrieves the associated covariance matrix from a fitted coco object

Usage

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

Arguments

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

Value

a vector with the adjusted trend

Author(s)

Federico Blasi

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

Description

Retrieves the estimated spatial effects of the spatial structure

Usage

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

Arguments

z.pred	...
mean.pred	...
sd.pred	...

Value

retrieves CRPS

Author(s)

Federico Blasi

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

Value

a list() with two elements: a design matrix of dimension (n x p), and a par.pos object, indexing columns of the design matrix refered to each aspect models.

Author(s)

Federico Blasi

Examples

```
model.list <- list(   "mean" = 0,
                      "std.dev" = as.formula(" ~ 1 + lati_s * long_s"),
                      "scale" = as.formula(" ~ 1 + elev_s"),
                      "aniso" = as.formula(" ~ 1 + elev_s"),
                      "tilt" = as.formula(" ~ 1 + elev_s"),
                      "smooth" = as.formula(" ~ 1"),
                      "nugget" = -Inf)
```

getEstims *Retrieve estimates from a fitted coco object*

Description

Retrieve estimates from a fitted coco object

Usage

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

Arguments

coco.object	a fitted coco object.

Value

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	a fitted coco object
ncores	number of cores used for the computation
eps	...

Value

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	a coco class (fitted) object

Value

wrap for value from a OptimParallel object

Author(s)

Federico Blasi

getLogScore *Based on a set of predictions retrieves the LogScore*

Description

Retrieves the estimated spatial effects of the spatial structure

Usage

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

Arguments

z.pred	...
mean.pred	...
sd.pred	...

Value

retrieves LogScore

Author(s)

Federico Blasi

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

Value

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

Compute confidence intervals for a (fitted) coco object

Usage

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

Arguments

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

Value

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`	a vector with parameters values
`par.pos`	par.pos list
`locs`	spatial location matrix
`x_covariates`	design matrix
`smooth.limits`	smooth.limits
`z`	a vector of observed values
`n`	dim(z)[1]
`lambda`	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	a vector with parameters values
par.pos	par.pos list
locs	spatial location matrix
x_covariates	design matrix
smooth.limits	smooth.limits
z	a vector of observed values
n	dim(z)[1]
x_betas	design matrix for the trend
lambda	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	a vector with parameters values
par.pos	par.pos list
ref_taper	spam object based on a taper based covariance function
locs	spatial location matrix
x_covariates	design matrix
smooth.limits	smooth.limits
cholS	Cholesky object from spam
z	a vector of observed values
n	dim(z)[1]
lambda	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	a vector with parameters values
par.pos	par.pos list
ref_taper	spam object based on a taper based covariance function
locs	spatial location matrix
x_covariates	design matrix
smooth.limits	smooth.limits
cholS	Cholesky object from spam
z	a vector of observed values
n	dim(z)[1]
lambda	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	...
lambda	...
theta_list	...
smooth.limits	...

Value

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	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	if provided, it centers covariates based on this information
sd.vector	if provided, it scales covariates based on this information

Value

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 *Retrieves the estimated spatial effects of the spatial structure*

Description

Retrieves the estimated surfaces for different sources of nonstationarity

Usage

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

Arguments

coco.object	an coco class (fitted) object

Value

a list with the different estimated surfaces

Author(s)

Federico Blasi

`getTrend` *Computes the trend of the coco object*

Description

Compute the trend of the (fitted) coco object

Usage

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

Arguments

`coco.object`	a coco class (fitted) object

Value

a vector with the adjusted trend

Author(s)

Federico Blasi

`holes` *Holes Data Set*

Description

Description of the holes data set.

Usage

```
holes
```

Format

A data frame with rows and variables:

var1 Description of var1

var2 Description of var2

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.
type	The type of plot.
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

Description of the stripes data set.

Usage

```
stripes
```

Format

A data frame with rows and variables:

var1 Description of var1

var2 Description of var2

Source

Source of the data

Examples

```
data(stripes)
```

Index