Package 'geostatsp'

June 23, 2020

Type Package

Title Geostatistical Modelling with Likelihood and Bayes

Version 1.8.2

Date 2020-06-22

Depends Matrix (>= 1.2.0),
raster,
sp,
R (>= 3.5.0)

Imports abind,
numDeriv,
methods,
stats

Suggests RandomFields (>= 3.3.4),
rgdal,
parallel,
mapmisc,
ellipse,
pracma,
knitr

Enhances INLA,
diseasemapping,
geoR,
rgeos,
mvtnorm

LinkingTo Matrix

Additional_repositories https://inla.r-inla-download.org/R/testing

Author Patrick Brown <patrick.brown@utoronto.ca>[aut, cre], Robert Hijmans [ctb]

Maintainer Patrick Brown <patrick.brown@utoronto.ca>

Description Geostatistical modelling facilities using Raster and SpatialPoints objects are provided. Non-Gaussian models are fit using INLA, and Gaussian geostatistical models use Maximum Likelihood Estimation. For details see Brown (2015) <doi:10.18637/jss.v063.i12>.

License GPL

NeedsCompilation yes

VignetteBuilder knitr

RoxygenNote 5.0.1

R topics documented:

asImRaster . . . 2
conditionalGmrf . . . 3
excProb . . . 4
gambiaUTM . . . 5
glgm-methods . . . 6
inla.models . . . 9
krigeLgm . . . 10
lgm-methods . . . 12
likfitLgm . . . 17
loaloa . . . 21
matern . . . 22
maternGmrfPrec . . . 25
murder . . . 28
pcPriorRange . . . 32
postExp . . . 33
profLlgm . . . 34
RFsimulate . . . 36
rongelapUTM . . . 38
simLgcp . . . 39
spatialRoc . . . 40
squareRaster-methods . . . 41
stackRasterList . . . 42
swissRain . . . 43
swissRainR . . . 47
variog . . . 48
wheat . . . 49

Index **51**

`asImRaster` *Convert a raster to an im object*

Description

Conversion between rasters and spatstat's im objects

Usage

```
asImRaster(X, ...)
```

Arguments

`X`	A RasterLayer
`...`	additional arguments

Details

This function is deceased. use `as.im.RasterLayer` in `maptools`

conditionalGmrf *Conditional distribution of GMRF*

Description

Distribution of Gaussian Markov Random Field conditional on data observed with noise on the same grid.

Usage

```
conditionalGmrf(param, Yvec, Xmat, NN,
template = NULL, mc.cores = 1,
cellsPerLoop = 10, ...)
```

Arguments

param	vector of named parameters
Yvec	vector of observed data, or matrix with each column being a realisation.
Xmat	Matrix of covariates.
NN	nearest neighbour matrix
template	Raster on which the GMRF is defined
mc.cores	passed to mcmapply
cellsPerLoop	number of cells to compute simultaneously. Larger values consume more memory but result in faster computation.
...	additional arguments passed to maternGmrfPrec

Value

Raster image with layers containing conditional mean and standard deviation.

Author(s)

Patrick Brown

See Also

maternGmrfPrec, lgm

excProb *Exceedance probabilities*

Description

Calculate exceedance probabilities pr(X > threshold) from a fitted geostatistical model.

Usage

```
excProb(x, threshold=0, random=FALSE, template=NULL, templateIdCol=NULL,
nuggetInPrediction=TRUE)
```

Arguments

x	Output from either the lgm or glgm functions, or a list of two-column matrices with columns named x and y containing the posterior distributions of random effects, as produced by inla.
threshold	the value which the exceedance probability is calculated with respect to.
random	Calculate exceedances for the random effects, rather than the predicted observations (including fixed effects).
template	A Raster or SpatialPolygonsDataFrame or SpatialPointsDataFrame object which the results will be contained in.
templateIdCol	The data column in template corresponding to names of marginals
nuggetInPrediction	If TRUE, calculate exceedance probabilities of new observations by adding the nugget effect. Otherwise calculate probabilities for the latent process. Ignored if x is output from glgm.

Details

When x is the output from lgm, pr(Y>threshold) is calculated using the Gaussian distribution using the Kriging mean and conditional variance. When x is from the glgm function, the marginal posteriors are numerically integrated to obtain pr(X > threshold).

Value

Either a vector of exceedance probabilities or an object of the same class as template.

Examples

```
data('swissRain')
swissFit =  lgm("rain", swissRain, grid=30,
boxcox=0.5,fixBoxcox=TRUE,covariates=swissAltitude)
swissExc = excProb(swissFit, 20)
mycol = c("green","yellow","orange","red")
mybreaks = c(0, 0.2, 0.8, 0.9, 1)
plot(swissBorder)
plot(swissExc, breaks=mybreaks, col=mycol,add=TRUE,legend=FALSE)
plot(swissBorder, add=TRUE)
legend("topleft",legend=mybreaks, col=c(NA,mycol))
## Not run:
```

```
swissRain$sqrtrain = sqrt(swissRain$rain)
swissFit2 =  glgm(formula="sqrtrain",data=swissRain, grid=40,
covariates=swissAltitude,family="gaussian")
swissExc = excProb(swissFit2, threshold=sqrt(30))
swissExc = excProb(swissFit2$inla$marginals.random$space, 0,
template=swissFit2$raster)

## End(Not run)
```

`gambiaUTM` *Gambia data*

Description

This data-set was used by Diggle, Moyeed, Rowlingson, and Thomson (2002) to demonstrate how the model-based geostatistics framework of Diggle et al. (1998) could be adapted to assess the source(s) of extrabinomial variation in the data and, in particular, whether this variation was spatially structured. The malaria prevalence data set consists of measurements of the presence of malarial parasites in blood samples obtained from children in 65 villages in the Gambia. Other child- and village-level indicators include age, bed net use, whether the bed net is treated, whether or not the village belonged to the primary health care structure, and a measure of 'greenness' using a vegetation index.

Usage

```
data(gambiaUTM)
```

Format

A SpatialPointsDataFrame, with column `pos` being the binary response for a malaria diagnosis, as well as other child-level indicators such as `netuse` and `treated` being measures of bed net use and whether the nets were treated. The column `green` is a village-level measure of greenness. A UTM coordinate reference system is used, where coordinates are in metres.

Source

http://www.leg.ufpr.br/doku.php/pessoais:paulojus:mbgbook:datasets. For further details on the malaria data, see Thomson et al. (1999).

References

Diggle, P. J., Moyeed, R. A., Rowlingson, R. and Thomson, M. (2002). Childhood Malaria in the Gambia: A case-study in model-based geostatistics. Journal of the Royal Statistical Society. Series C (Applied Statistics), 51(4): 493-506.

Diggle, P. J., Tawn, J. A. and Moyeed, R. A. (1998). Model-based geostatistics (with Discussion). Applied Statistics, 47, 299–350.

Thomson, M. C., Connor, S. J., D'Alessandro, U., Rowlingson, B., Diggle, P., Creswell, M. and Greenwood, B. (2004). Predicting malaria infection in Gambian children from satellite data and bed net use surveys: the importance of spatial correlation in the interpretation of results. American Journal of Tropical Medicine and Hygiene, 61: 2-8.

Examples

```
data("gambiaUTM")

plot(gambiaUTM, main="gambia data")

## Not run:
# get the gambia data

gambia = read.table(
"http://www.leg.ufpr.br/lib/exe/fetch.php/pessoais:paulojus:mbgbook:datasets:gambia.txt",
header=TRUE)

# create projection without epsg code so rgdal doesn't need to be loaded
library(sp)
library(rgdal)
theproj = CRSargs(CRS("+init=epsg:32628"))
theproj = gsub("\+init=epsg:[[:digit:]]+ ", "", theproj)
theproj = CRS(theproj)

gambiaUTM = SpatialPointsDataFrame(gambia[,c("x","y")],
data=gambia[,-(1:2)],
proj4string = theproj)
save(gambiaUTM,
file="~/workspace/diseasemapping/pkg/geostatsp/data/gambiaUTM.RData",
compress="xz")


download.file("http://thematicmapping.org/downloads/TM_WORLD_BORDERS-0.3.zip",
   "borders.zip")
unzip("borders.zip")
worldBorders = readOGR(".", "TM_WORLD_BORDERS-0.3")
africa = worldBorders[worldBorders$REGION ==2,]
plot(gambiaUTM)
plot(spTransform(africa, gambiaUTM@proj4string),add=TRUE)

## End(Not run)
```

`glgm-methods` *Generalized Linear Geostatistical Models*

Description

Fits a generalized linear geostatistical model or a log-Gaussian Cox process using `inla`

Usage

```
## S4 method for signature 'ANY,ANY,ANY,ANY'
glgm(formula, data,  grid, covariates, buffer=0, shape=1, prior, ...)
## S4 method for signature 'formula,Raster,ANY,ANY'
glgm(formula, data,  grid, covariates, buffer=0, shape=1, prior, ...)
## S4 method for signature 'formula,Spatial,ANY,ANY'
glgm(formula, data,  grid, covariates, buffer=0, shape=1, prior, ...)
```

```
## S4 method for signature 'formula,data.frame,Raster,data.frame'
glgm(formula, data,  grid, covariates, buffer=0, shape=1, prior, ...)
lgcp(formula=NULL, data,  grid, covariates=NULL, border, ...)
```

Arguments

data	An object of class SpatialPointsDataFrame containing the data.
grid	Either an integer giving the number of cells in the x direction, or a raster object which will be used for the spatial random effect. If the cells in the raster are not square, the resolution in the y direction will be adjusted to make it so.
covariates	Either a single raster, a list of rasters or a raster stack containing covariate values used when making spatial predictions. Names of the raster layers or list elements correspond to names in the formula. If a covariate is missing from the data object it will be extracted from the rasters. Defaults to NULL for an intercept-only model.
formula	Model formula, defaults to a linear combination of each of the layers in the covariates object. The spatial random effect should not be supplied but the default can be overridden with a f(space,..) term. For glgm the response variable defaults to the first variable in the data object, and formula can be an integer or character string specifying the response variable. For lgcp, the formula should be one-sided.
prior	list with elements named range, sd, sdObs. See Details.
shape	Shape parameter for the Matern correlation function, must be 1 or 2.
buffer	Extra space padded around the data bounding box to reduce edge effects.
border	boundary of the region on which an LGCP is defined, passed to mask
...	Additional options passed to inla

Details

This function performs Bayesian inference for generalized linear geostatistical models with INLA. The Markov random field approximation on a regular lattice is used for the spatial random effect. The range parameter is the distance at which the correlation is 0.13, or

$$cov[U(s+h), U(s)] = (2^{1-\nu}/Gamma(\nu))d^{\nu}besselK(d,\nu)$$

$$d = |h|\sqrt{8\nu}/range$$

where ν is the shape parameter. The range parameter produced by glgm multiplies the range parameter from INLA by the cell size.

Elements of prior can be named range, sd, or sdObs. Elements can consist of:

- a single value giving the prior median for penalized complexity priors (exponential on the sd or 1/range).
- a vector c(u=a,alpha=b) giving an quantile and probability for pc priors. For standard deviations alpha is an upper quantile, for the range parameter b = pr(1/range > 1/a).
- a vector c(lower=a,upper=b) giving a 0.025 and 0.975 quantiles for the sd or range.
- a list of the form list(prior='loggamma',param=c(1,2)) passed directly to inla.
- a two-column matrix of prior densities for the sd or range.

Value

A list with two components named `inla`, `raster`, and `parameters`. `inla` contains the results of the call to the `inla` function. `raster` is a raster stack with the following layers:

`random.`	mean, sd, X0.0??quant: Posterior mean, standard deviation, and quantiles of the random effect
`predict.`	mean, sd, X0.0??quant: same for linear predictors, on the link scale
`predict.exp`	posterior mean of the exponential of the linear predictor
`predict.invlogit`	Only supplied if a binomial response variable was used.

`parameters` contains a list with elements:

`summary`	a table with parameter estimates and posterior quantiles
`range, sd`	prior and posterior distributions of range and standard deviations

See Also

inla, http://r-inla.org

Examples

```
# use the 'safe' version of INLA on linux systems
if(Sys.info()['sysname'] =='Linux' &
  requireNamespace("INLA", quietly = TRUE)) {
  INLA::inla.setOption(inla.call = system.file(paste("bin/linux/",
          ifelse(.Machine$sizeof.pointer == 4, "32", "64"),
          'bit/inla.static', sep=''), package="INLA")) }

# geostatistical model for the swiss rainfall data
require("geostatsp")
data("swissRain")
swissRain$lograin = log(swissRain$rain)
swissFit =  glgm(formula="lograin", data=swissRain,
grid=30,
covariates=swissAltitude, family="gaussian",
buffer=2000,
prior = list(sd=1, range=100*1000, sdObs = 2),
control.inla = list(strategy='gaussian')
)

if(!is.null(swissFit$parameters) ) {

swissExc = excProb(swissFit, threshold=log(25))

swissExcRE = excProb(swissFit$inla$marginals.random$space,
log(1.5),template=swissFit$raster)

swissFit$parameters$summary

matplot(
swissFit$parameters$range$postK[,'x'],
swissFit$parameters$range$postK[,c('y','prior')],
type="l", lty=1, xlim = c(0, 1000),
xlab = 'km', ylab='dens')
```

```
legend('topright', lty=1, col=1:2, legend=c('post','prior'))

plot(swissFit$raster[["predict.exp"]])

mycol = c("green","yellow","orange","red")
mybreaks = c(0, 0.2, 0.8, 0.95, 1)
plot(swissBorder)
plot(swissExc, breaks=mybreaks, col=mycol,add=TRUE,legend=FALSE)
plot(swissBorder, add=TRUE)
legend("topleft",legend=mybreaks, fill=c(NA,mycol))


plot(swissBorder)
plot(swissExcRE, breaks=mybreaks, col=mycol,add=TRUE,legend=FALSE)
plot(swissBorder, add=TRUE)
legend("topleft",legend=mybreaks, fill=c(NA,mycol))
}




# a log-Gaussian Cox process example

myPoints = SpatialPoints(cbind(rbeta(100,2,2), rbeta(100,3,4)))
myPoints@bbox = cbind(c(0,0), c(1,1))

mycov = raster(matrix(rbinom(100, 1, 0.5), 10, 10), 0, 1, 0, 1)
names(mycov)="x1"


res = lgcp(
formula=~factor(x1),
data=myPoints,
grid=20, covariates=mycov,
prior=list(sd=c(0.9, 1.1), range=c(0.4, 0.41),
control.inla = list(strategy='gaussian'))
)
if(requireNamespace("INLA", quietly = TRUE)) {
plot(res$raster[["predict.exp"]])
plot(myPoints,add=TRUE,col="#0000FF30",cex=0.5)
}
```

`inla.models` *Valid models in INLA*

Description

calls the function of the same name in INLA

Usage

```
inla.models()
```

Value

a list

`krigeLgm` *Spatial prediction, or Kriging*

Description

Perform spatial prediction, producing a raster of predictions and conditional standard deviations.

Usage

```
krigeLgm(formula, data, grid,  covariates = NULL,
param,
    expPred = FALSE, nuggetInPrediction = TRUE,
    mc.cores=getOption("mc.cores", 1L))
```

Arguments

`formula`	Either a model formula, or a data frame of linear covariates.
`data`	A SpatialPointsDataFrame containing the data to be interpolated
`grid`	Either a `raster`, or a single integer giving the number of cells in the X direction which predictions will be made on. If the later the predictions will be a raster of square cells covering the bounding box of `data`.
`covariates`	The spatial covariates used in prediction, either a `raster` stack or list of rasters.
`param`	A vector of named model parameters, as produced by `likfitLgm`
`expPred`	Should the predictions be exponentiated, defaults to `FALSE`.
`nuggetInPrediction`	If `TRUE`, predict new observations by adding the nugget effect. The prediction variances will be adjusted accordingly, and the predictions on the natural scale for logged or Box Cox transformed data will be affected. Otherwise predict fitted values.
`mc.cores`	passed to `mclapply` if greater than 1.

Details

Given the model parameters and observed data, conditional means and variances of the spatial random field are computed.

Value

A raster stack is returned with the following layers:

`fixed`	Estimated means from the fixed effects portion of the model
`random`	Predicted random effect
`krige.var`	Conditional variance of predicted random effect (on the transformed scale if applicable)

predict	Prediction of the response, sum of fixed and random effects. If exp.pred is TRUE, gives predictions on the exponentiated scale, and half of krige.var is added prior to exponentiating
predict.log	If exp.pred=TRUE, the prediction of the logged process.
predict.boxcox	If a box cox transformation was used, the prediction of the process on the transformed scale.

If the prediction locations are different for fixed and random effects (typically coarser for the random effects), a list with two raster stacks is returned.

prediction	A raster stack as above, though the random effect prediction is resampled to the same locations as the fixed effects.
random	the predictions and conditional variance of the random effects, on the same raster as newdata

See Also

lgm

Examples

```
data('swissRain')
swissRain$lograin = log(swissRain$rain)
swissRain[[names(swissAltitude)]] = extract(swissAltitude, swissRain)

if(interactive()  | Sys.info()['user'] =='patrick') {
swissFit = likfitLgm(data=swissRain,
formula=lograin~ CHE_alt,
param=c(range=46500, nugget=0.05,shape=1,
anisoAngleDegrees=35, anisoRatio=12),
paramToEstimate = c("range","nugget",
"anisoAngleDegrees", "anisoRatio")
)
myTrend = swissFit$model$formula
myParams = swissFit$param
dput(myParams)
# will give the following
} else {
myParams=structure(c(0.0951770829449953, 0.77308786208928, 49379.3845752436,
1, 11.673076577513, 0.649925237653982, 1, 2.26103881494066, 0.000146945995279231,
37.2379731166102), .Names = c("nugget", "variance", "range",
"shape", "anisoRatio", "anisoAngleRadians", "boxcox", "(Intercept)",
"CHE_alt", "anisoAngleDegrees"))
myTrend =lograin~ CHE_alt
}

# make sure krige can cope with missing values!
swissAltitude[1:50,1:50] = NA
swissKrige = krigeLgm(data=swissRain,
formula = myTrend,
covariates = swissAltitude,
param=myParams,
grid = 40, expPred=TRUE)
```

```
plot(swissKrige[["predict"]], main="predicted rain")
plot(swissBorder, add=TRUE)

# now with box cox and provide a raster for prediction, no covariates

if(interactive()  | Sys.info()['user'] =='patrick') {
swissFit2 = likfitLgm(
data=swissRain,
formula=rain~1,
param=c(range=52000, nugget=0.1,
shape=1, boxcox=0.5,
anisoAngleDegrees=35, anisoRatio=8),
paramToEstimate = c("range","nugget",
"anisoAngleDegrees", "anisoRatio"),
parscale = c(range=5000,nugget=0.01,
anisoRatio=1,anisoAngleDegrees=5)
)
myTrend2 = swissFit2$trend
myParams2 = swissFit2$param
dput(myParams2)
} else {
myParams2=structure(c(0.865530531647866, 8.76993204385615, 54143.5826959284,
1, 7.36559089705556, 0.647158492167979, 0.5, 5.16254700135706,
37.0794502772753), .Names = c("nugget", "variance", "range",
"shape", "anisoRatio", "anisoAngleRadians", "boxcox", "(Intercept)",
"anisoAngleDegrees"))
myTrend2=rain~1
}

swissRaster = raster(extent(swissBorder), nrows=25, ncols=40)


swissKrige2 = krigeLgm(data=swissRain, formula = myTrend2,
  param=myParams2,
grid = swissRaster)




plot(swissKrige2[["predict"]], main="predicted rain with box-cox")
plot(swissBorder, add=TRUE)
```

---

lgm-methods *Linear Geostatistical Models*

Description

Calculate MLE's of model parameters and perform spatial prediction.

Usage

```
## S4 method for signature 'missing,ANY,ANY,ANY'
lgm(
formula, data, grid, covariates,
buffer=0, shape=1, boxcox=1, nugget = 0,
expPred=FALSE, nuggetInPrediction=TRUE,
reml=TRUE,mc.cores=1,
aniso=FALSE,
fixShape=TRUE,
fixBoxcox=TRUE,
fixNugget = FALSE,
...)
## S4 method for signature 'numeric,ANY,ANY,ANY'
lgm(
formula, data, grid, covariates,
buffer=0, shape=1, boxcox=1, nugget = 0,
expPred=FALSE, nuggetInPrediction=TRUE,
reml=TRUE,mc.cores=1,
aniso=FALSE,
fixShape=TRUE,
fixBoxcox=TRUE,
fixNugget = FALSE,
...)
## S4 method for signature 'character,ANY,ANY,ANY'
lgm(
formula, data, grid, covariates,
buffer=0, shape=1, boxcox=1, nugget = 0,
expPred=FALSE, nuggetInPrediction=TRUE,
reml=TRUE,mc.cores=1,
aniso=FALSE,
fixShape=TRUE,
fixBoxcox=TRUE,
fixNugget = FALSE,
...)
## S4 method for signature 'formula,Spatial,numeric,ANY'
lgm(
formula, data, grid, covariates,
buffer=0, shape=1, boxcox=1, nugget = 0,
expPred=FALSE, nuggetInPrediction=TRUE,
reml=TRUE,mc.cores=1,
aniso=FALSE,
fixShape=TRUE,
fixBoxcox=TRUE,
fixNugget = FALSE,
...)
## S4 method for signature 'formula,Spatial,Raster,missing'
lgm(
formula, data, grid, covariates,
buffer=0, shape=1, boxcox=1, nugget = 0,
expPred=FALSE, nuggetInPrediction=TRUE,
reml=TRUE,mc.cores=1,
aniso=FALSE,
```

```
fixShape=TRUE,
fixBoxcox=TRUE,
fixNugget = FALSE,
...)
## S4 method for signature 'formula,Spatial,Raster,list'
lgm(
formula, data, grid, covariates,
buffer=0, shape=1, boxcox=1, nugget = 0,
expPred=FALSE, nuggetInPrediction=TRUE,
reml=TRUE,mc.cores=1,
aniso=FALSE,
fixShape=TRUE,
fixBoxcox=TRUE,
fixNugget = FALSE,
...)
## S4 method for signature 'formula,Spatial,Raster,Raster'
lgm(
formula, data, grid, covariates,
buffer=0, shape=1, boxcox=1, nugget = 0,
expPred=FALSE, nuggetInPrediction=TRUE,
reml=TRUE,mc.cores=1,
aniso=FALSE,
fixShape=TRUE,
fixBoxcox=TRUE,
fixNugget = FALSE,
...)
## S4 method for signature 'formula,Spatial,Raster,data.frame'
lgm(
formula, data, grid, covariates,
buffer=0, shape=1, boxcox=1, nugget = 0,
expPred=FALSE, nuggetInPrediction=TRUE,
reml=TRUE,mc.cores=1,
aniso=FALSE,
fixShape=TRUE,
fixBoxcox=TRUE,
fixNugget = FALSE,
...)
## S4 method for signature 'formula,Raster,ANY,ANY'
lgm(
formula, data, grid, covariates,
buffer=0, shape=1, boxcox=1, nugget = 0,
expPred=FALSE, nuggetInPrediction=TRUE,
reml=TRUE,mc.cores=1,
aniso=FALSE,
fixShape=TRUE,
fixBoxcox=TRUE,
fixNugget = FALSE,
...)
## S4 method for signature 'formula,data.frame,Raster,data.frame'
lgm(
formula, data, grid, covariates,
buffer=0, shape=1, boxcox=1, nugget = 0,
```

```
expPred=FALSE, nuggetInPrediction=TRUE,
reml=TRUE,mc.cores=1,
aniso=FALSE,
fixShape=TRUE,
fixBoxcox=TRUE,
fixNugget = FALSE,
...)
```

Arguments

formula	A model formula for the fixed effects, or a character string specifying the response variable.
data	A SpatialPointsDataFrame or Raster layer, brick or stack containing the locations and observations, and possibly covariates.
grid	Either a raster, or a single integer giving the number of cells in the X direction which predictions will be made on. If the later the predictions will be a raster of square cells covering the bounding box of data.
covariates	The spatial covariates used in prediction, either a raster stack or list of rasters. Covariates in formula but not in data will be extracted from covariates.
shape	Order of the Matern correlation
boxcox	Box-Cox transformation parameter (or vector of parameters), set to 1 for no transformation.
nugget	Value for the nugget effect (observation error) variance, or vector of such values.
expPred	Should the predictions be exponentiated, defaults to FALSE.
nuggetInPrediction	If TRUE, predict new observations by adding the nugget effect. The prediction variances will be adjusted accordingly, and the predictions on the natural scale for logged or Box Cox transformed data will be affected. Otherwise predict fitted values.
reml	If TRUE (the default), use restricted maximum likelihood.
mc.cores	If mc.cores>1, this argument is passed to mclapply and computations are done in parallel where possible.
aniso	Set to TRUE to use geometric anisotropy.
fixShape	Set to FALSE to estimate the Matern order
fixBoxcox	Set to FALSE to estimate the Box-Cox parameter.
fixNugget	Set to FALSE to estimate the nugget effect parameter.
buffer	Extra distance to add around grid.
...	Additional arguments passed to likfitLgm. Starting values can be specified with a vector param of named elements

Details

When data is a SpatialPointsDataFrame, parameters are estimated using optim to maximize the log-likelihood function computed by likfitLgm and spatial prediction accomplished with krigeLgm.

With data being a Raster object, a Markov Random Field approximation to the Matern is used (experimental). Parameters to be estimated should be provided as vectors of possible values, with optimization only considering the parameter values supplied.

Value

A list is returned which includes a RasterStack named predict having layers:

fixed	Estimated means from the fixed effects portion of the model
random	Predicted random effect
krigeSd	Conditional standard deviation of predicted random effect (on the transformed scale if applicable)
predict	Prediction of the response, sum of predicted fixed and random effects. For Box-Cox or log-transformed data on the natural (untransformed) scale.
predict.log	If exp.pred=TRUE, the prediction of the logged process.
predict.boxcox	If a box cox transformation was used, the prediction of the process on the transformed scale.

In addition, the element summery contains a table of parameter estimates and confidence intervals. optim contains the output from the call to the optim function.

See Also

likfitLgm, krigeLgm

Examples

```
data("swissRain")

swissRes =  lgm( formula="rain",
data=swissRain[1:60,], grid=20,
covariates=swissAltitude, boxcox=0.5, fixBoxcox=TRUE,
shape=1, fixShape=TRUE,
aniso=FALSE, nugget=0, fixNugget=FALSE,
nuggetInPrediction=FALSE
)

swissRes$summary

plot(swissRes$predict[["predict"]], main="predicted rain")
plot(swissBorder, add=TRUE)



## Not run:
load(url("http://www.filefactory.com/file/frd1mhownd9/n/CHE_adm0_RData"))

library('RColorBrewer')
par(mar=c(0,0,0,3))
plot(gadm)
plot(mask(projectRaster(
swissRes$predict[["predict"]],  crs=gadm@proj4string),gadm),
add=T,alpha=0.6, col=brewer.pal(9, "Blues"))
plot(gadm, add=TRUE)


## End(Not run)
```

likfitLgm *Likelihood Based Parameter Estimation for Gaussian Random Fields*

Description

Maximum likelihood (ML) or *restricted maximum likelihood* (REML) parameter estimation for (transformed) Gaussian random fields.

Usage

```
likfitLgm(formula, data,
paramToEstimate = c("range","nugget"),
reml=TRUE,
coordinates=data,
param=NULL,
upper=NULL,lower=NULL, parscale=NULL,
verbose=FALSE)


loglikLgm(param,
data, formula, coordinates=data,
reml=TRUE,
minustwotimes=TRUE,
moreParams=NULL)
```

Arguments

formula	A formula for the fixed effects portion of the model, specifying a response and covariates. Alternately, data can be a vector of observations and formula can be a model matrix.
data	An object of class SpatialPointsDataFrame, a vector of observations, or a data frame containing observations and covariates.
coordinates	A SpatialPoints object containing the locations of each observation, which defaults to data. Alternately, coordinates can be a symmetricMatrix-class or dist object reflecting the distance matrix of these coordinates (though this is only permitted if the model is isotropic).
param	A vector of model parameters, with named elements being amongst range,nugget,boxcox,shape,a and possibly variance (see matern). When calling likfitLgm this vector is a combination of starting values for parameters to be estiamated and fixed values of parameters which will not be estimated. For loglikLgm, it is the covariance parameters for which the likelihood will be evaluated.
reml	Whether to use Restricted Likelihood rather than Likelihood, defaults to TRUE.

paramToEstimate
: Vector of names of model parameters to estimate, with parameters excluded from this list being fixed. The variance parameter and regression coefficients are always estimated even if not listed.

lower
: Named vector of lower bounds for model parameters passed to optim, defaults are used for parameters not specified.

upper
: Upper bounds, as above.

parscale
: Named vector of scaling of parameters passed as control=list(parscale=parscale) to optim.

minustwotimes
: Return -2 times the log likelihood rather than the likelihood

moreParams
: Vector of additional parameters, combined with param. Used for passing fixed parameters to loglikLgm from within optim.

verbose
: if TRUE information is printed by optim.

Value

likfitLgm produces list with elements

parameters
: Maximum Likelihood Estimates of model parameters

varBetaHat
: Variance matrix of the estimated regression parameters

optim
: results from optim

trend
: Either formula for the fixed effects or names of the columns of the model matrix, depending on trend supplied.

summary
: a table of parameter estimates, standard errors, confidence intervals, p values, and a logical value indicating whether each parameter was estimated as opposed to fixed.

resid
: residuals, being the observations minus the fixed effects, on the transformed scale.

loglikLgm returns a scalar value, either the log likelihood or -2 times the log likelihood. Attributes of this result include the vector of parameters (including the MLE's computed for the variance and coefficients), and the variance matrix of the coefficient MLE's.

See Also

lgm

Examples

```
n=40
mydat = SpatialPointsDataFrame(
cbind(runif(n), seq(0,1,len=n)),
data=data.frame(cov1 = rnorm(n), cov2 = rpois(n, 0.5))
)

# simulate a random field
trueParam = c(variance=2^2, range=0.35, shape=2, nugget=0.5^2)
set.seed(1)

mydat@data= cbind(mydat@data , RFsimulate(model=trueParam,x=mydat)@data)

# add fixed effects
```

```
mydat$Y = -3 + 0.5*mydat$cov1 + 0.2*mydat$cov2 +
mydat$sim + rnorm(length(mydat), 0, sd=sqrt(trueParam["nugget"]))

spplot(mydat, "sim", col.regions=rainbow(10), main="U")
spplot(mydat, "Y", col.regions=rainbow(10), main="Y")


myres = likfitLgm(
formula=Y ~ cov1 + cov2,
data=mydat,
param=c(range=0.1,nugget=0.1,shape=2),
paramToEstimate = c("range","nugget")
)



myres$summary[,1:4]

if(interactive()  | Sys.info()['user'] =='patrick') {

# plot variograms of data, true model, and estimated model
myv = variog(mydat, formula=Y ~ cov1 + cov2,option="bin", max.dist=0.5)
# myv will be NULL if geoR isn't installed
if(!is.null(myv)){
plot(myv, ylim=c(0, max(c(1.2*sum(trueParam[c("variance", "nugget")]),myv$v))),
main="variograms")
distseq = seq(0, 0.5, len=50)
lines(distseq,
sum(myres$param[c("variance", "nugget")]) - matern(distseq, param=myres$param),
col='blue', lwd=3)
lines(distseq,
sum(trueParam[c("variance", "nugget")]) - matern(distseq, param=trueParam),
col='red')

legend("bottomright", fill=c("black","red","blue"),
legend=c("data","true","MLE"))
}

# without a nugget
myresNoN = likfitLgm(
formula=Y ~ cov1 + cov2,
data=mydat,
param=c(range=0.1,nugget=0,shape=1),
paramToEstimate = c("range")
)

myresNoN$summary[,1:4]


# plot variograms of data, true model, and estimated model
myv = variog(mydat, formula=Y ~ cov1 + cov2,option="bin", max.dist=0.5)

if(!is.null(myv)){
plot(myv, ylim=c(0, max(c(1.2*sum(trueParam[c("variance", "nugget")]),myv$v))),
main="variograms")

distseq = seq(0, 0.5, len=50)
```

```
lines(distseq,
sum(myres$param[c("variance", "nugget")]) - matern(distseq, param=myres$param),
col='blue', lwd=3)
lines(distseq,
sum(trueParam[c("variance", "nugget")]) - matern(distseq, param=trueParam),
col='red')

lines(distseq,
sum(myresNoN$param[c("variance", "nugget")]) -
matern(distseq, param=myresNoN$param),
col='green', lty=2, lwd=3)
legend("bottomright", fill=c("black","red","blue","green"),
legend=c("data","true","MLE","no N"))
}


# calculate likelihood
temp=loglikLgm(param=myres$param,
data=mydat,
formula = Y ~ cov1 + cov2,
reml=FALSE, minustwotimes=FALSE)



# an anisotropic example


trueParamAniso = param=c(variance=2^2, range=0.2, shape=2,
 nugget=0,anisoRatio=4,anisoAngleDegrees=10, nugget=0)

mydat$U = geostatsp::RFsimulate(trueParamAniso,mydat)$sim

mydat$Y = -3 + 0.5*mydat$cov1 + 0.2*mydat$cov2 +
mydat$U + rnorm(length(mydat), 0, sd=sqrt(trueParamAniso["nugget"]))



par(mfrow=c(1,2))
oldmar = par("mar")
par(mar=rep(0.1, 4))

plot(mydat, col=as.character(cut(mydat$U, breaks=50, labels=heat.colors(50))),
pch=16, main="aniso")

plot(mydat, col=as.character(cut(mydat$Y, breaks=50, labels=heat.colors(50))),
pch=16,main="iso")
par(mar=oldmar)


myres = likfitLgm(
formula=Y ~ cov1 + cov2,
data=mydat,
param=c(range=0.1,nugget=0,shape=2, anisoAngleDegrees=0, anisoRatio=2),
paramToEstimate = c("range","nugget","anisoRatio","anisoAngleDegrees")
)

myres$summary
```

```
par(mfrow=c(1,2))

myraster = raster(nrows=30,ncols=30,xmn=0,xmx=1,ymn=0,ymx=1)
covEst = matern(myraster, y=c(0.5, 0.5), par=myres$param)
covTrue = matern(myraster, y=c(0.5, 0.5), par=trueParamAniso)

plot(covEst, main="estimate")
plot(covTrue, main="true")

par(mfrow=c(1,1))
}
```

`loaloa` *Loaloa prevalence data from 197 village surveys*

Description

Location and prevalence data from villages, elevation an vegetation index for the study region.

Usage

```
data("loaloa")
```

Format

`loaloa` is a SpatialPolygonsDataFrame of the data, with columns `N` being the number of individuals tested and `y` being the number of positives. `elevationLoa` is a raster of elevation data. `eviLoa` is a raster of vegetation index for a specific date. `ltLoa` is land type. `ltLoa` is a raster of land types. 1 2 5 6 7 8 9 10 11 12 13 14 15 `tempLoa` is a raster of average temperature in degrees C.

Source

http://www.leg.ufpr.br/doku.php/pessoais:paulojus:mbgbook:datasets for the loaloa data, https://lpdaac.usgs.gov/dataset_discovery/modis/modis_products_table for EVI and land type and http://srtm.csi.cgiar.org for the elevation data.

Examples

```
data("loaloa")
plot(loaloa, main="loaloa villages")

# elevation
plot(elevationLoa, col=terrain.colors(100), main="elevation")
points(loaloa)

# vegetation index
plot(eviLoa, main="evi")
points(loaloa)

plot(tempLoa, main="temperature")
points(loaloa)
```

```
# land type, a categorical variable
plot(ltLoa)
mapmisc::legendBreaks("bottomleft",ltLoa, bty='n')
points(loaloa)
```

`matern` *Evaluate the Matern correlation function*

Description

Returns the Matern covariance for the distances supplied.

Usage

```
matern( x, param=c(range=1, variance=1, shape=1),
type=c('variance','cholesky','precision', 'inverseCholesky'),
y=NULL)
## S3 method for class 'SpatialPoints'
matern(x,  param,
type=c('variance','cholesky','precision', 'inverseCholesky'),
y=NULL)
## Default S3 method:
matern( x, param,
type=c('variance','cholesky','precision', 'inverseCholesky'),
y=NULL)
## S3 method for class 'dist'
matern( x, param,
type=c('variance','cholesky','precision', 'inverseCholesky'),
y=NULL)
## S3 method for class 'Raster'
matern( x,  param,
type=c('variance','cholesky','precision', 'inverseCholesky'),
y=NULL)
## S3 method for class 'SpatialPointsDataFrame'
matern(x, param,
type=c('variance','cholesky','precision', 'inverseCholesky'),
y=NULL)
fillParam(param)
```

Arguments

`x`	A vector or matrix of distances, or `Raster` or `SpatialPoints` of locations, see Details below.
`param`	A vector of named model parameters with, at a minimum names `range` and `shape` (see Details), and optionally `variance` (defaults to 1) and `nugget` (defaults to zero). For Geometric Anisotropy add `anisoRatio` and either `anisoAngleDegrees` or `anisoAngleRadians`

type — specifies if the variance matrix, the Cholesky decomposition of the variance matrix, the precision matrix, or the inverse of the Cholesky L matrix is returned.

y — Covariance is calculated for the distance between locations in x and y. If y=NULL, covariance of x with itself is produced. However, if x is a matrix or vector it is assumed to be a set of distances and y is ignored.

Details

The formula for the Matern correlation function is

$$M(x) = \frac{variance}{\Gamma(shape)} 2^{1-shape} \left(\frac{x\sqrt{8shape}}{range} \right)^{shape} besselK(x\sqrt{8shape}/range, shape)$$

The range argument is sqrt(8*shape)*phi.geoR, sqrt(8*shape)*scale.whittle.RandomFields, and 2*scale.matern.RandomFields.

Geometric anisotropy is only available when x is a Raster or SpatialPoints. The parameter 'anisoAngle' refers to rotation of the coordinates anti-clockwise by the specified amount prior to calculating distances, which has the effect that the contours of the correlation function are rotated clockwise by this amount. anisoRatio is the amount the Y coordinates are divided by by post rotation prior to calculating distances. A large value of anisoRatio makes the Y coordinates smaller and increases the correlation in the Y direction.

When x or y are rasters, cells are indexed row-wise starting at the top left.

Value

When x is a vector or matrix or object of class dist, a vector or matrix of covariances is returned. With x being SpatialPoints, y must also be SpatialPoints and a matrix of correlations between x and y is returned. When x is a Raster, and y is a single location a Raster of covariances between each pixel centre of x and y is returned.

Examples

```
param=c(shape=2.5,range=1,variance=1)
u=seq(0,4,len=200)
uscale = sqrt(8*param['shape'])* u / param['range']

theMaterns = cbind(
dist=u,
manual= param['variance']* 2^(1- param['shape']) *
( 1/gamma(param['shape'])  )  *
uscale^param['shape'] * besselK(uscale , param['shape']),
geostatsp=geostatsp::matern(u, param=param)
)
head(theMaterns)
matplot(theMaterns[,'dist'],
theMaterns[,c('manual','geostatsp')],
col=c('red','blue'), type='l',
xlab='dist', ylab='var')
legend('topright', fill=c('red','blue'),
legend=c('manual','geostatsp'))



# example with raster
```

```
myraster = raster(nrows=40,ncols=60,xmn=-3,xmx=3,ymn=-2,ymx=2)
param = c(range=2, shape=2,anisoRatio=2,
anisoAngleDegrees=-25,variance=20)

# plot correlation of each cell with the origin
myMatern = matern(myraster, y=c(0,0), param=param)


plot(myMatern, main="anisortopic matern")


# correlation matrix for all cells with each other
myraster = raster(nrows=4,ncols=6,xmn=-3,xmx=3,ymn=-2,ymx=2)
myMatern = matern(myraster, param=c(range=2, shape=2))
dim(myMatern)

# plot the cell ID's
values(myraster) = seq(1, ncell(myraster))
mydf = as.data.frame(myraster, xy=TRUE)
plot(mydf$x, mydf$y, type='n', main="cell ID's")
text(mydf$x, mydf$y, mydf$layer)
# correlation between bottom-right cell and top right cell is
myMatern[6,24]

# example with points
mypoints = SpatialPoints(
cbind(runif(8), runif(8))
)
# variance matrix from points
m1=matern(mypoints, param=c(range=2,shape=1.4,variance=4,nugget=1))
# cholesky of variance from distances
c2=matern(dist(mypoints@coords), param=c(range=2,shape=1.4,variance=4,nugget=1),type='cholesky')

# check it's correct
quantile(as.vector(m1- tcrossprod(c2)))

# example with vector of distances
range=3
distVec = seq(0, 2*range, len=100)
shapeSeq = c(0.5, 1, 2,20)
theCov = NULL
for(D in shapeSeq) {
theCov = cbind(theCov, matern(distVec, param=c(range=range, shape=D)))
}
matplot(distVec, theCov, type='l', lty=1, xlab='distance', ylab='correlation',
main="matern correlations")
legend("right", fill=1:length(shapeSeq), legend=shapeSeq,title='shape')
# exponential

distVec2 = seq(0, max(distVec), len=20)
points(distVec2, exp(-2*(distVec2/range)),cex=1.5, pch=5)
# gaussian
points(distVec2, exp(-2*(distVec2/range)^2), col='blue',cex=1.5, pch=5)
legend("bottomleft", pch=5, col=c('black','blue'), legend=c('exp','gau'))

# comparing to geoR and RandomFields
```

```
if (requireNamespace("RandomFields", quietly = TRUE) &
requireNamespace("geoR", quietly = TRUE)
) {

covGeoR = covRandomFields = NULL

for(D in shapeSeq) {
covGeoR = cbind(covGeoR,
geoR::matern(distVec, phi=range/sqrt(8*D), kappa=D))
covRandomFields = cbind(covRandomFields,
RandomFields::RFcov(x=distVec,
model=RandomFields::RMmatern(nu=D, var=1,
scale=range/2) ))
}



matpoints(distVec, covGeoR, cex=0.5, pch=1)
matpoints(distVec, covRandomFields, cex=0.5, pch=2)

legend("topright", lty=c(1,NA,NA), pch=c(NA, 1, 2),
legend=c("geostatsp","geoR","RandomFields"))
}
```

`maternGmrfPrec` *Precision matrix for a Matern spatial correlation*

Description

Produces the precision matrix for a Gaussian random field on a regular square lattice, using a Markov random field approximation.

Usage

```
maternGmrfPrec(N, ...)
## S3 method for class 'dgCMatrix'
maternGmrfPrec(N,
param=c(variance=1, range=1, shape=1, cellSize=1),
  adjustEdges=FALSE,...)
## Default S3 method:
maternGmrfPrec(N, Ny=N,
  param=c(variance=1, range=1, shape=1, cellSize=1),
  adjustEdges=FALSE, ...)
NNmat(N, Ny=N, nearest=3, adjustEdges=FALSE)
## S3 method for class 'Raster'
NNmat(N, Ny=N, nearest=3, adjustEdges=FALSE)
## Default S3 method:
NNmat(N, Ny=N, nearest=3, adjustEdges=FALSE)
```

Arguments

N	Number of grid cells in the x direction, or a matrix denoting nearest neighbours.
Ny	Grid cells in the y direction, defaults to N for a square grid
param	Vector of model parameters, with named elements: scale, scale parameter for the correlation function; prec, precision parameter; shape, Matern differentiability parameter (0, 1, or 2); and cellSize, the size of the grid cells. Optionally, variance and range can be given in place of prec and scale, when the former are present and the latter are missing the reciprocal of the former are taken.
adjustEdges	If TRUE, adjust the precision matrix so it does not implicitly assume the field takes values of zero outside the specified region. Defaults to FALSE. Can be a character string specifying the parameters to use for the correction, such as 'optimal' or 'optimalShape', with TRUE equivalent to 'theo'
nearest	Number of nearest neighbours to compute
...	Additional arguments passed to maternGmrfPrec.dsCMatrix

Details

The numbering of cells is consistent with the raster package. Cell 1 is the top left cell, with cell 2 being the cell to the right and numbering continuing row-wise.

The nearest neighbour matrix N has: N[i,j]=1 if i=j; takes a value 2 if i and j are first 'rook' neighbours; 3 if they are first 'bishop' neighbours; 4 if they are second 'rook' neighbours; 5 if 'knight' neighbours; and 6 if third 'rook' neighbours.

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    0    0    0    6    0    0    0
[2,]    0    0    5    4    5    0    0
[3,]    0    5    3    2    3    5    0
[4,]    6    4    2    1    2    4    6
[5,]    0    5    3    2    3    5    0
[6,]    0    0    5    4    5    0    0
[7,]    0    0    0    6    0    0    0
```

Value

A sparse matrix dsCMatrix-class object, containing a precision matrix for a Gaussian random field or (from the NNmat function) a matrix denoting neighbours.

Examples

```
# produces the matrix above
matrix(NNmat(11, 11, nearest=5)[,11*5+6],11, 11)

params=c(range = 3,shape=2, variance=5^2)

myGrid = squareRaster(extent(0,20,0,10), 40)

# precision matrix without adjusting for edge effects
precMat =maternGmrfPrec(N=myGrid, param=params)

attributes(precMat)$info$precisionEntries

midcell = cellFromRowCol(myGrid,
```

```
round(nrow(myGrid)/2), round(ncol(myGrid)/2)) # the middle cell
edgeCell = cellFromRowCol(myGrid, 5,5)# cell near corner

# show precision of middle cell
precMid=matrix(precMat[,midcell],
nrow(myGrid), ncol(myGrid), byrow=TRUE)

precMid[round(nrow(precMid)/2) + seq(-5, 5),
round(ncol(precMid)/2) + seq(-3, 3)]

# and with the adjustment
precMatCorr =maternGmrfPrec(
N = myGrid, param=params,
adjustEdges=TRUE)

if(interactive() | Sys.info()['user'] =='patrick') {

# variance matrices
varMat = Matrix::solve(precMat)
varMatCorr = Matrix::solve(precMatCorr)

# compare covariance matrix to the matern
xseq = seq(-ymax(myGrid), ymax(myGrid), len=1000)/1.5
plot(xseq, matern(xseq, param=params),
 type = 'l',ylab='cov', xlab='dist',
 ylim=c(0, params["variance"]*1.1),
 main="matern v gmrf")

# middle cell
varMid=matrix(varMat[,midcell],
nrow(myGrid), ncol(myGrid), byrow=TRUE)
varMidCorr=matrix(varMatCorr[,midcell],
nrow(myGrid), ncol(myGrid), byrow=TRUE)
xseqMid = yFromRow(myGrid) - yFromCell(myGrid, midcell)
points(xseqMid, varMid[,colFromCell(myGrid, midcell)],
col='red')
points(xseqMid, varMidCorr[,colFromCell(myGrid, midcell)],
 col='blue', cex=0.5)

# edge cells
varEdge=matrix(varMat[,edgeCell],
  nrow(myGrid), ncol(myGrid), byrow=TRUE)
varEdgeCorr = matrix(varMatCorr[,edgeCell],
  nrow(myGrid), ncol(myGrid), byrow=TRUE)
xseqEdge = yFromRow(myGrid) - yFromCell(myGrid, edgeCell)
points(xseqEdge,
varEdge[,colFromCell(myGrid, edgeCell)],
pch=3,col='red')
points(xseqEdge,
  varEdgeCorr[,colFromCell(myGrid, edgeCell)],
  pch=3, col='blue')

legend("topright", lty=c(1, NA, NA, NA, NA),
  pch=c(NA, 1, 3, 16, 16),
col=c('black','black','black','red','blue'),
legend=c('matern', 'middle','edge','unadj', 'adj')
)
```

```
# construct matern variance matrix

myraster = attributes(precMat)$raster
covMatMatern = matern(myraster, param=params)

  prodUncor = crossprod(covMatMatern, precMat)
  prodCor = crossprod(covMatMatern, precMatCorr)

  quantile(Matrix::diag(prodUncor),na.rm=TRUE)
  quantile(Matrix::diag(prodCor),na.rm=TRUE)

  quantile(prodUncor[lower.tri(prodUncor,diag=FALSE)],na.rm=TRUE)
  quantile(prodCor[lower.tri(prodCor,diag=FALSE)],na.rm=TRUE)
}
```

`murder` *Murder locations*

Description

Locations of murders in Toronto 1990-2014

Usage

```
data("murder")
```

Format

`murder` is a SpatialPoints object of murder locations. `torontoPdens`, `torontoIncome`, and `torontoNight` are rasters containing population density (per hectare), median household income, and ambient light respectively. `torontoBorder` is a SpatialPolygonsDataFrame of the boundary of the city of Toronto.

Source

Murder data:http://www.thestar.com/news/crime/torontohomicidemap.html,

Lights: https://ngdc.noaa.gov/eog/viirs/download_ut_mos.html

Boundary files: http://www12.statcan.gc.ca/census-recensement/2011/geo/bound-limit/bound-limit-2006-eng.cfm

Income: http://www12.statcan.gc.ca/census-recensement/2006/dp-pd/tbt/Ap-eng.cfm?LANG=E&APATH=3&DETAIL=0&DIM=0&FL=A&FREE=0&GC=0&GID=0&GK=0&GRP=1&PID=88982&PRID=0&PTYPE=88971,97154&S=0&SHOWALL=0&SUB=0&Temporal=2006&THEME=66&VID=0&VNAMEE=&VNAMEF=

Examples

```
data("murder")
plot(torontoBorder)
points(murder, col="#0000FF40", cex=0.5)

data("torontoPop")

# light
data("murder")
plot(torontoNight, main="Toronto ambient light")
plot(torontoBorder, add=TRUE)
points(murder, col="#0000FF40", cex=0.5)


# income
plot(torontoIncome, main="Toronto Income")
points(murder, col="#0000FF40", cex=0.5)
plot(torontoBorder, add=TRUE)

# population density
plot(torontoPdens, main="Toronto pop dens")
points(murder, col="#0000FF40", cex=0.5)
plot(torontoBorder, add=TRUE)


## Not run:
#building the dataset
fpath <- system.file("extdata", "murder1990.csv", package="geostatsp")
murderList=list()
# Load in datafiles retrieved from
# http://www.thestar.com/news/crime/torontohomicidemap.html
# Each year's murders are in a separate file, with
# 1997, for example, being '/inst/extdata/murder1997.csv'
# this file was obtained by selecting the year '1997' from the
# menu marked 'select year', then clicking 'Download' at the bottom right
# selecting 'data' and in the new window clicking 'Underlying',
# then 'show all columnns' and 'dowload all rows as text file'
for(Dyear in 1990:2014){
Dfile = gsub("1990", Dyear, fpath)
murderList[[as.character(Dyear)]] = read.table(Dfile, sep=",", header=TRUE,
comment.char="", as.is=TRUE, quote="\"")
}
murderFull = do.call(rbind, murderList)

murder = murderFull[,c(
"age.of.victim","sex.of.victim",
"Homicide.type.groupings","method",
 "URL","Details..if.available.")]
names(murder) = tolower(gsub("\..+$", "", names(murder)))
names(murder) = gsub("^homicide$", "type", names(murder))
murder$method = gsub(",", "", murder$method)
murder$sex = factor(murder$sex)
murder$type = factor(murder$type)
murder$date = as.Date(
as.character(murderFull$Date),
format="
```

```
murder$year = as.integer(format(murder$date, format="
murder$date = as.Date(murder$date)

mcoord = as.matrix(
murderFull[,c("Addresses._longitude",
"Addresses._latitude")]
)

murder=SpatialPointsDataFrame(
coords=mcoord,
data=murder,
proj4string=CRS("+init=epsg:4326")
)

# get rid of children to keep this from being too morbid
murder = murder[which(murder$age >= 18),]

utm = CRS("+proj=utm +zone=17 +datum=NAD83 +units=m +no_defs +ellps=GRS80 +towgs84=0,0,0")
murder = spTransform(murder, utm)

# boundary of toronto
zipfileB = paste(tempfile(), ".zip", sep="")
download.file(
paste('http://www12.statcan.gc.ca/census-recensement/2011/geo/',
'bound-limit/files-fichiers/gcd_000b06a_e.zip',sep=''),
zipfileB)
unzip(zipfileB, exdir=tempdir())
theshpB = grep("shp$",unzip(zipfileB,list=TRUE)$Name,  value=TRUE)
theshpB = gsub("\.shp$", "", theshpB)
torontoBorder = rgdal::readOGR(tempdir(),theshpB,
stringsAsFactors=FALSE)
torontoBorder = torontoBorder[
grep("toronto",
as.character(torontoBorder$CDNAME),
ignore.case=TRUE),
]


torontoBorder = spTransform(
torontoBorder, CRS(proj4string(murder))
)
library('mapmisc')
toMap = openmap(torontoBorder)
map.new(torontoBorder)
plot(toMap,add=TRUE)
plot(torontoBorder,add=TRUE)
points(murder,col='green')


save(murder, torontoBorder,
file="~/workspace/diseasemapping/pkg/geostatsp/data/murder.RData",
compress='xz')

# census tract boundaries
zipfileCT = file.path(tempdir(),'toCt.zip')
download.file(
paste('http://www12.statcan.gc.ca/census-recensement/2011/geo/bound-limit/',
```

```
'files-fichiers/gct_000b06a_e.zip', sep=''),
zipfileCT
)

unzip(zipfileCT, exdir=tempdir())
theshpCT = grep("shp$",unzip(zipfileCT,list=TRUE)$Name,  value=TRUE)
theshpCT = gsub("\.shp$", "", theshpCT)

toCt2006 = readOGR(tempdir(),theshpCT)
toCt2006 = spTransform(toCt2006, CRS(proj4string(torontoBorder)))
aspoints = SpatialPoints(toCt2006)
projection(aspoints) = CRS(proj4string(torontoBorder))
inToronto = over(aspoints, torontoBorder)$CDNAME
toCt2006 =toCt2006[!is.na(inToronto),]
toCt2006 = spTransform(toCt2006,
CRS(proj4string(murder)))


# income
zipfileI = paste(tempfile(), ".zip", sep="")
download.file(
paste('http://www12.statcan.gc.ca/census-recensement/2006/dp-pd/tbt/',
'OpenDataDownload.cfm?PID=96273', sep=""),
zipfileI)

unzip(zipfileI, exdir=tempdir())
thesdmx = grep("^Generic",unzip(zipfileI,list=TRUE)$Name,  value=TRUE)
library('rsdmx')
toIncSdmx = readSDMX(file.path(tempdir(), thesdmx), isURL=FALSE)
toInc = as.data.frame(toIncSdmx)
toIncSub = toInc[grep("^535", toInc$GEO),]
toIncSub = toIncSub[toIncSub$DIM0==3 & toIncSub$DIM1==1,]
rownames(toIncSub) = toIncSub$GEO
toCt2006$income_median_household = toIncSub[
gsub("\.", "", as.character(toCt2006$CTUID)),
'obsValue']

torontoIncome = rasterize(toCt2006,
squareRaster(torontoBorder, 150),
field='income_median_household')



nightFile = paste(tempfile(), ".tif.gz", sep="")

# higher resolution at
paste('http://mapserver.ngdc.noaa.gov/viirs_data/viirs_composite/'
'npp_20120418to20120426_20121011to20121023_sloff_15asec.'
'x2y1.75N180W.c20121120.avg_dnb.tif.gz',sep='')
download.file(
'http://www.worldgrids.org/lib/exe/fetch.php?media=lnmdms3a.tif.gz',
nightFile)
library("R.utils")
gunzip(nightFile,overwrite=TRUE)
nightFull = raster(gsub("\.gz$", "", nightFile))

border2 = spTransform(torontoBorder, CRS(projection(nightFull)))
```

```
toMap2 = openmap(border2)

torontoNight = crop(nightFull, raster::extend(extent(border2),0.1))
torontoNight = projectRaster(torontoNight, crs=CRS(proj4string(murder)))
cscaleNight = colourScale(torontoNight,breaks=9,style='equal',dec=0)

map.new(torontoBorder)
plot(toMap,add=TRUE)
plot(torontoNight,add=TRUE,
col=cscaleNight$col,
breaks=cscaleNight$breaks,
legend=FALSE,
alpha=0.5)
plot(torontoBorder,add=TRUE)
legendBreaks('bottomright', cscaleNight)

save(torontoIncome, torontoNight,
torontoPdens,
file="../pkg/geostatsp/data/torontoPop.RData",
compress='xz')


## End(Not run) # end don't run
```

`pcPriorRange` *PC prior for range parameter*

Description

Creates a penalized complexity prior for the range parameter

Usage

```
pcPriorRange(q, p=0.5, cellSize=1)
```

Arguments

`q`	Lower quantile for the range parameter
`p`	probability that the range is below this quantile, defaults to the median
`cellSize`	size of grid cells, can be a raster.

Details

q is the quantile in spatial units, usually meters, and the scale parameter follows an exponential distribution. A prior PC prior distribution for the range parameter in units of grid cells, which INLA requires, is computed.

Value

A list with

lambda	parameter for the exponential distribution (for scale in units of cells), in the same parametrization as dexp
priorScale	matrix with x and y columns with prior of scale parameter
priorRange	matris with x and y columns with prior of range parameter, in meters (or original spatial units)
inla	character string specifying this prior in inla's format

Examples

```
# pr(range < 100km) = 0.1, 200m grid cells
x = pcPriorRange(q=100*1000, p=0.1, cellSize = 200)
rangeSeq = seq(0, 1000, len=1001)
plot(rangeSeq, x$dprior$range(rangeSeq*1000)*1000,
  type='l', xlab="range, 1000's km", ylab='dens')
cat(x$inla)
```

---

postExp *Exponentiate posterior quantiles*

Description

Converts a summary table for model parameters on the log scale to the natural or exponentiated scale.

Usage

```
postExp(x,
exclude = grep('^(range|aniso|shape|boxcox)', rownames(x)),
invLogit=FALSE)
```

Arguments

x	a matrix or data frame as returned by glgm
exclude	vector of parameters not transformed, defaults to the range parameter
invLogit	Converts intercept parameter to inverse-logit scale when TRUE. Can also be a vector of parameters to inverse-logit transform.

Examples

```
require("geostatsp")
data("swissRain")
swissRain$lograin = log(swissRain$rain)

if(requireNamespace('INLA', quietly=TRUE)) {
swissFit =  glgm(formula="lograin", data=swissRain,
grid=20,
covariates=swissAltitude/1000, family="gaussian",
prior = list(sd=1, range=100*1000, sdObs = 2),
```

```
control.inla = list(strategy='gaussian', int.strategy='eb'),
control.mode = list(theta=c(1.6542995, 0.7137123,2.2404179))
)
postExp(swissFit$parameters$summary)

swissFitLgm =  lgm( formula="lograin",
data=swissRain[1:60,], grid=20,
covariates=swissAltitude/1000,
shape=1, fixShape=TRUE,
aniso=FALSE, nugget=0, fixNugget=FALSE
)

swissFitLgm$summary
}
```

`profLlgm` *Joint confidence regions*

Description

Calculates profile likelihoods and approximate joint confidence regions for covariance parameters in linear geostatistical models.

Usage

```
profLlgm(fit, mc.cores = 1, ...)
informationLgm(fit,  ...)
```

Arguments

`fit`	Output from the `lgm` function
`mc.cores`	Passed to `mclapply`
`...`	For `profLlgm`, one or more vectors of parameter values at which the profile likelihood will be calculated, with names corresponding to elements of `fit$param`. For `informationLgm`, arguments passed to `hessian`

Value

`one or more vectors`	of parameter values
`logL`	A vector, matrix, or multi-dimensional array of profile likelihood values for every combination of parameter values supplied.
`full`	Data frame with profile likelihood values and estimates of model parameters
`prob,breaks`	vector of probabilities and chi-squared derived likelihood values associated with those probabilities
`MLE,maxLogL`	Maximum Likelihood Estimates of parameters and log likelihood evaluated at these values
`basepars`	combination of starting values for parameters re-estimated for each profile likelihood and values of parameters which are fixed.
`col`	vector of colours with one element fewer than the number of probabilities
`ci,ciLong`	when only one parameter is varying, a matrix of confidence intervals (in both wide and long format) is returned.

Author(s)

Patrick Brown

See Also

lgm, mcmapply, hessian

Examples

```
# this example is time consuming
# the following 'if' statement ensures the CRAN
# computer doesn't run it
if(interactive() | Sys.info()['user'] =='patrick') {

library('geostatsp')
data('swissRain')

swissFit = lgm(data=swissRain, formula=rain~ CHE_alt,
grid=10, covariates=swissAltitude,
shape=1,  fixShape=TRUE,
boxcox=0.5, fixBoxcox=TRUE,
aniso=TRUE,reml=TRUE,
param=c(anisoAngleDegrees=37,anisoRatio=7.5,
range=50000))


x=profLlgm(swissFit,
anisoAngleDegrees=seq(30, 43 , len=4)
)


plot(x[[1]],x[[2]], xlab=names(x)[1],
ylab='log L',
ylim=c(min(x[[2]]),x$maxLogL),
type='n')
abline(h=x$breaks[-1],
col=x$col,
lwd=1.5)
axis(2,at=x$breaks,labels=x$prob,line=-1.2,
tick=FALSE,
las=1,padj=1.2,hadj=0)
abline(v=x$ciLong$par,
lty=2,
col=x$col[as.character(x$ciLong$prob)])
lines(x[[1]],x[[2]], col='black')



}
```

RFsimulate *Simulation of Random Fields*

Description

This function simulates conditional and unconditional Gaussian random fields, calling the function in the RandomFields package of the same name.

Usage

```
## S4 method for signature 'ANY,Raster'
RFsimulate(model, x,data=NULL,
 err.model=NULL, n = 1, ...)
## S4 method for signature 'numeric,SpatialGrid'
RFsimulate(model, x,data=NULL,
 err.model=NULL, n = 1, ...)
## S4 method for signature 'numeric,SpatialPixels'
RFsimulate(model, x, data=NULL,
err.model=NULL, n = 1, ...)
## S4 method for signature 'numeric,SpatialPoints'
RFsimulate(model, x, data=NULL,
  err.model=NULL, n = 1, ...)
## S4 method for signature 'numeric,GridTopology'
RFsimulate(model, x, data=NULL,
 err.model=NULL, n = 1, ...)
## S4 method for signature 'RMmodel,GridTopology'
RFsimulate(model, x, data=NULL,
 err.model=NULL, n = 1, ...)
## S4 method for signature 'RMmodel,SpatialPoints'
RFsimulate(model, x, data=NULL,
 err.model=NULL, n = 1, ...)
## S4 method for signature 'matrix,Raster'
RFsimulate(model, x,  data=NULL,
 err.model=NULL, n = nrow(model), ...)
## S4 method for signature 'matrix,Spatial'
RFsimulate(model, x,data=NULL,
err.model=NULL, n = nrow(model), ...)
## S4 method for signature 'data.frame,ANY'
RFsimulate(model, x,data=NULL,
err.model=NULL, n = nrow(model), ...)
modelRandomFields(param, includeNugget=FALSE)
```

Arguments

model	object of class RMmodel, a vector of named model parameters, or a matrix where each column is a model parameter
x	Object of type GridTopology or Raster or SpatialPoints or SpatialPixels.
data	For conditional simulation and random imputing only. If data is missing, unconditional simulation is performed.Object of class SpatialPointsDataFrame; coordinates and response values of measurements in case that conditional simulation is to be performed

err.model	For conditional simulation and random imputing only. Usually err.model=RMnugget(var=var), or not given at all (error-free measurements).
n	number of realizations to generate.
...	for advanced use: further options and control parameters for the simulation that are passed to and processed by RFoptions
param	A vector of named parameters
includeNugget	If FALSE, the nugget parameter is ignored.

Details

If model is a matrix, a different set of parameters is used for each simulation. If data has the same number of columns as model has rows, a different column i is used with parameters in row i.

Value

An object of the same class as x, with the exception of x being a GridTopology where a Raster is returned.

Author(s)

Patrick E. Brown <patrick.brown@utoronto.ca>

See Also

RFsimulate, RFfit, RFgetModelInfo, RFgui, RMmodel, RFoptions, RFsimulateAdvanced, RFsimulate.more.exampl

Examples

```
library('geostatsp')
model <- c(var=5, range=1,shape=0.5)


myraster = raster(nrows=20,ncols=30,xmn=0,ymn=0,xmx=6,ymx=4,
crs=CRS("+proj=utm +zone=17 +datum=NAD27 +units=m +no_defs"))

set.seed(0)

simu <- RFsimulate(model, x=myraster, n=3)

plot(simu[['sim2']])

# conditional simulation
firstSample = RFsimulate(
model,
x=SpatialPoints(myraster)[seq(1,ncell(myraster), len=100), ],
n=3
)

secondSample = RFsimulate(
model = cbind(var=5:3, range=1:3, shape=seq(0.5, 1.5, len=3)),
x= myraster,
data=firstSample,n=4
)
```

```
plot(secondSample)

# convert the model to RandomFields format and plot
if(requireNamespace('RandomFields', quietly=TRUE)) {
RandomFields::plot(modelRandomFields(model))
}
```

`rongelapUTM` *Rongelap data*

Description

This data-set was used by Diggle, Tawn and Moyeed (1998) to illustrate the model-based geostatistical methodology introduced in the paper. discussed in the paper. The radionuclide concentration data set consists of measurements of γ-ray counts at 157 locations.

Usage

```
data(rongelapUTM)
```

Format

A SpatialPolygonsDataFrame, with columns `count` being the radiation count and `time` being the length of time the measurement was taken for. A UTM coordinate reference system is used, where coordinates are in metres.

Source

http://www.leg.ufpr.br/doku.php/pessoais:paulojus:mbgbook:datasets. For further details on the radionuclide concentration data, see Diggle,Harper and Simon (1997), Diggle, Tawn and Moyeed (1998) and Christensen (2004).

References

Christensen, O. F. (2004). Monte Carlo maximum likelihood in model-based geostatistics. Journal of computational and graphical statistics **13** 702-718.

Diggle, P. J., Harper, L. and Simon, S. L. (1997). Geostatistical analysis of residual contamination from nuclea testing. In: *Statistics for the environment 3: pollution assesment and control* (eds. V. Barnet and K. F. Turkmann), Wiley, Chichester, 89-107.

Diggle, P. J., Tawn, J. A. and Moyeed, R. A. (1998). Model-based geostatistics (with Discussion). Applied Statistics, 47, 299–350.

Examples

```
data("rongelapUTM")
plot(rongelapUTM, main="Rongelap island")
## Not run:
load(url("http://www.filefactory.com/file/54e8egxfddul/n/MHL_adm0_RData"))

rongelapLL = spTransform(rongelapUTM, gadm@proj4string)
```

```
plot(rongelapLL)
plot(gadm, add=T)

## End(Not run)
```

---

simLgcp *Simulate a log-Gaussian Cox process*

Description

Give covariates and model parameters, simulates a log-Gaussian Cox process

Usage

```
simLgcp(param, covariates=NULL, betas=NULL,
offset=NULL,
  rasterTemplate=covariates[[1]], n=1, ...)
simPoissonPP(intensity)
```

Arguments

param	A vector of named model parameters with, at a minimum names range and shape (see Details), and optionally variance (defaults to 1). For Geometric Anisotropy add anisoRatio and either anisoAngleDegrees or anisoAngleRadians
covariates	Either a raster stack or list of rasters and SpatialPolygonsDataFrames (with the latter having only a single data column).
betas	Coefficients for the covariates
offset	Vector of character strings corresponding to elements of covariates which are offsets
rasterTemplate	Raster on which the latent surface is simulated, defaults to the first covariate.
n	number of realisations to simulate
...	additional arguments, see RFsimulate.
intensity	Raster of the intensity of a Poisson point process.

Value

A list with a events element containing the event locations and a raster element containing a raster stack of the covariates, spatial random effect, and intensity.

Examples

```
mymodel = c(mean=-0.5, variance=1,
range=2, shape=2)

myraster = raster(nrows=15,ncols=20,xmn=0,xmx=10,ymn=0,ymx=7.5)

# some covariates, deliberately with a different resolution than myraster
covA = covB = myoffset = raster(extent(myraster), 10, 10)
```

```
values(covA) = as.vector(matrix(1:10, 10, 10))
values(covB) = as.vector(matrix(1:10, 10, 10, byrow=TRUE))
values(myoffset) = round(seq(-1, 1, len=ncell(myoffset)))

myCovariate = list(a=covA, b=covB, offsetFooBar = myoffset)

myLgcp=simLgcp(param=mymodel,
covariates=myCovariate,
betas=c(a=-0.1, b=0.25),
offset='offsetFooBar',
rasterTemplate=myraster)

plot(myLgcp$raster[["intensity"]], main="lgcp")
points(myLgcp$events)

myIntensity = exp(-1+0.2*myCovariate[["a"]])
myPoissonPP = simPoissonPP(myIntensity)[[1]]
plot(myIntensity, main="Poisson pp")
points(myPoissonPP)
```

`spatialRoc` *Sensitivity and specificity*

Description

Calculate ROC curves using model fits to simulated spatial data

Usage

```
spatialRoc(fit, rr = c(1, 1.2, 1.5, 2), truth, border=NULL,
random = FALSE, prob = NULL, spec = seq(0,1,by=0.01))
```

Arguments

`fit`	A fitted model from the `lgcp` function
`rr`	Vector of relative risks exceedance probabilities will be calculated for. Values are on the natural scale, with `spatialRoc` taking logs when appropriate.
`truth`	True value of the spatial surface, or result from `simLgcp` function. Assumed to be on the log scale if `random=TRUE` and on the natural scale otherwise.
`border`	optional, `SpatialPolygonsDataFrame` specifying region that calculations will be restricted to.
`random`	compute ROC's for relative intensity (`FALSE`) or random effect (`TRUE`)
`prob`	Vector of exceedance probabilities
`spec`	Vector of specificities for the resulting ROC's to be computed for.

Details

Fitted models are used to calculate exceedance probabilities, and a location is judged to be above an `rr` threshold if this exceedance probability is above a specified probability threshold. Each raster cell of the true surface is categorized as being either true positive, false positive, true negative, and false negative and sensitivity and specificity computed. ROC curves are produced by varying the probability threshold.

Value

An array, with dimension 1 being probability threshold, dimension 2 being the relative risk threshold, dimension 3 being sensitivity and specificity. If `fit` is a list of model fits, dimension 4 corresponds to elements of `fit`.

Author(s)

Patrick Brown

See Also

`lgcp`, `simLgcp`, `excProb`

---

squareRaster-methods *Create a raster with square cells*

Description

Given a raster object, an extent, or a bounding box, a raster of with square cells and having the extent and number of cells specified is returned.

Usage

```
## S4 method for signature 'matrix'
squareRaster(x,cells=NULL, buffer=0)
## S4 method for signature 'Raster'
squareRaster(x,cells=NULL, buffer=0)
## S4 method for signature 'Spatial'
squareRaster(x,cells=NULL, buffer=0)
## S4 method for signature 'Extent'
squareRaster(x,cells=NULL, buffer=0)
```

Arguments

`x`	A bounding box from a `SpatialPoints` or `SpatialPolygons` object or an `Extent` from a Raster.
`cells`	The number of cells in the x direction. If NULL the number of columns of x is used.
`buffer`	Additional area to add around the resulting raster

Value

A `rasterLayer` with square cells

Examples

```
myraster = raster(matrix(0,10,10),xmn=0,xmx=10,ymn=0,ymx=12.3)

squareRaster(myraster)

squareRaster(extent(myraster), cells=10)

squareRaster(bbox(myraster), cells=10)
```

`stackRasterList` *Converts a list of rasters, possibly with different projections and resolutions, to a single raster stack.*

Description

This function is intended to be used prior to passing covariates to krigeLgm in order for the rasters for all covariates to have the same projection and same resolution.

Usage

```
stackRasterList(x, template = x[[1]], method = "ngb", mc.cores=NULL)
spdfToBrick(x,
    template,
    logSumExpected=FALSE,
    pattern = '^expected_[[:digit:]]+$'
)
```

Arguments

`x`	A list of `Raster` or `SpatialPolygonsDataFrames` for `stackRasterList` and `spdfToBrick` respectively
`template`	A raster whose projection and resolution all other rasters will be aligned with.
`method`	The method to use, either "ngb", or "bilinear". Can be a vector of the same length as x to specify different methods for each raster. If `method` has names which correspond to the names of `x`, the names will be used instead of the order to assign methods to rasters.
`mc.cores`	If non-null, `mclapply` is used with this argument specifying the number of cores.
`logSumExpected`	return the log of the sum of offsets
`pattern`	expression to identify layers to rasterize in `x`

Value

A raster brick, with one layer for each variable.

Examples

```
mylist = list(a=raster(matrix(1:9, 3, 3), 0,1,0,1,
       crs="+proj=utm +zone=17 +ellps=GRS80 +units=m +no_defs"),
b=raster(matrix(1:25, 5, 5), -1, 2, -1, 2,
   crs="+proj=utm +zone=17 +ellps=GRS80 +units=m +no_defs")
)
mystack = stackRasterList(mylist)
mystack



mylist = list(a=raster(matrix(1:36, 6, 6,byrow=TRUE), 0,1000,0,1000,
       crs="+proj=utm +zone=17 +ellps=GRS80 +units=m +no_defs"),
b=raster(matrix(1:144, 12, 12), -85.49, -85.48, 0, 0.01,
   crs="+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs "),
c=raster(matrix(1:100, 10, 10), -5000,5000,-5000,5000,
       crs="+proj=utm +zone=17 +ellps=GRS80 +units=m +no_defs")
)

if(.Platform$OS.type=='unix' & require('rgdal', quietly=TRUE)) {
mystack = stackRasterList(mylist,mc.cores=2)
mystack
}

plot(mystack[["b"]], main="stack b")
plot(mystack[['a']],add=TRUE,col=grey(seq(0,1,len=12)),alpha=0.8,legend=FALSE)
```

`swissRain` *Swiss rainfall data*

Description

Data from the SIC-97 project: Spatial Interpolation Comparison.

Usage

```
data("swissRain")
```

Format

`swissRain` is a SpatialPolygonsDataFrame 100 daily rainfall measurements made in Switzerland on the 8th of May 1986. `swissAltitude` is a raster of elevation data, and `swissLandType` is a raster of land cover types.

Source

https://wiki.52north.org/AI_GEOSTATS/AI_GEOSTATSData and http://srtm.csi.cgiar.org and https://lpdaac.usgs.gov/dataset_discovery/modis/modis_products_table

Examples

```
data("swissRain")
plot(swissAltitude, main="elevation")
points(swissRain)
plot(swissBorder, add=TRUE)


# land type, a categorical variable
commonValues  = sort(table(values(swissLandType)),decreasing=TRUE)[1:5]
commonValues=commonValues[!names(commonValues)==0]

thelevels = levels(swissLandType)[[1]]$ID
thebreaks = c(-0.5, 0.5+thelevels)
thecol = rep(NA, length(thelevels))
names(thecol) = as.character(thelevels)

thecol[names(commonValues)] = rainbow(length(commonValues))

plot(swissLandType, breaks=thebreaks, col=thecol,legend=FALSE,
main="land type")
points(swissRain)
plot(swissBorder, add=TRUE)


legend("topleft",fill=thecol[names(commonValues)],
legend=levels(swissLandType)[[1]][
match(as.integer(names(commonValues)),
levels(swissLandType)[[1]]$ID),
"Category"],
bty='n'
)

# code to assemble the dataset
## Not run:
dataDir = tempdir()

download.file(
"https://wiki.52north.org/pub/AI_GEOSTATS/AI_GEOSTATSData/sic97data_01.zip",
destfile=paste(dataDir, "swiss.zip",sep=""))
swissFile = unzip(paste(dataDir, 'swiss.zip',sep=""), exdir=dataDir)
swissRain = read.table(grep("sic_obs.dat",swissFile, value=TRUE),
sep=',',
col.names=c('ID','x','y','rain'))
# the following seems to make the coordinates line up with epsg:2056
swissRain$x = swissRain$x - 17791.29 + 2672591
swissRain$y = swissRain$y - 13224.66 + 1200225
# the readme file says rain is in tenths of mm
swissRain$rain= swissRain$rain / 10
library(sp)
library(rgdal)
# create projection without epsg code so rgdal doesn't need to be loaded
#theproj = CRSargs(CRS("+init=epsg:2056"))
theproj = CRS(paste("+proj=somerc +lat_0=46.9524055555556",
"+lon_0=7.43958333333333 +k_0=1 +x_0=2600000 +y_0=1200000"))

#theproj = gsub("\+init=epsg:[[:digit:]]+ ", "", theproj)
```

```
#theproj = CRS(theproj)

swissRain = SpatialPointsDataFrame(swissRain[,c('x','y')], data=swissRain[,c('ID','rain')],
proj4string=theproj)



#######
# Swiss Border
#######


library('raster')
swissBorder = getData('GADM', country='CHE', level=0)
isChar = which(unlist(lapply(swissBorder@data, is.character)))
isUtf = which(
unlist(lapply(swissBorder@data[,isChar],
Encoding)) == 'UTF-8')
swissBorder = swissBorder[,
!match(names(swissBorder), names(isUtf), nomatch=0)
]
library(rgdal)
swissBorder = spTransform(swissBorder, CRS(proj4string(swissRain)))

############
# land type
############
# see loaloa's help file for installation of the MODIS package
library(MODIS)
MODISoptions(gdalPath="/usr/bin/",
localArcPath=dataDir, outDirPath=dataDir)
options()[grep("MODIS", names(options()), value=TRUE)]

myProduct = "MCD12Q1"
getProduct(myProduct)

thehdf=getHdf(product=myProduct,
begin="2002-01-01",end="2002-01-02",
tileH=18, tileV=4)
# extent=extent(spTransform(swissBorder, CRS("+init=epsg:4326")))

layerNames = getSds(thehdf[[1]][1])$SDSnames
ltLayer = grep("Type_1$", layerNames)
theString = rep(0, length(layerNames))
theString[ltLayer] = 1
theString = paste(theString, collapse="")

runGdal(product=myProduct,
begin="2002-01-01",end="2002-01-02",
outProj = proj4string(swissRain),
pixelSize=2000, job="loa",
SDSstring = theString,
tileH=17:18, tileV=3:4)

# extent=extent(spTransform(swissBorder, CRS("+init=epsg:4326")))

# find file name
```

```
thenames = preStack(
path = paste(options()$MODIS_outDirPath, "loa",sep=""),
pattern=myProduct)
swissLandType = raster(thenames)
swissLandType = crop(swissLandType, extend(extent(swissBorder),20000))


swissLandType = as.factor(swissLandType)

# labels of land types
library(XML)
labels = readHTMLTable("http://nsidc.org/data/ease/ancillary.html")
labels = labels[[grep("Land Cover Classes", names(labels))]]
classCol = grep("Class Number", names(labels))
labels[,classCol] = as.integer(as.character(labels[,classCol]))

labels[  grep("Water", labels$Category),
classCol
] = 0
labelVec = as.character(labels$Category)
names(labelVec) = as.character(labels[,classCol])


levels(swissLandType)[[1]]$Category =
labelVec[as.character(levels(swissLandType)[[1]]$ID)]

levels(swissLandType)[[1]]$col = NA

theForests = grep("forest", levels(swissLandType)[[1]]$Category,
ignore.case=TRUE)

library(RColorBrewer)
levels(swissLandType)[[1]][theForests,"col"] =
brewer.pal(length(theForests)+1, "Greens")[-1]

levels(swissLandType)[[1]][
grep("snow", levels(swissLandType)[[1]]$Category,ignore.case=TRUE),
"col"] = "#FFFFFF"

levels(swissLandType)[[1]][
grep("water", levels(swissLandType)[[1]]$Category,ignore.case=TRUE),
"col"] = "#0000FF"

levels(swissLandType)[[1]][
grep("grass", levels(swissLandType)[[1]]$Category,ignore.case=TRUE),
"col"] = "#CCBB00"


stillNA = is.na(levels(swissLandType)[[1]]$col)
levels(swissLandType)[[1]][stillNA, "col"] =
brewer.pal(sum(stillNA), "Set3")



swissLandType@legend@colortable = levels(swissLandType)[[1]]$col

levels(swissLandType)[[1]]$n = table(values(swissLandType))
```

```
plot(swissLandType)
mostCommon = levels(swissLandType)[[1]]$n >= 700
legend("topright",
fill=levels(swissLandType)[[1]][mostCommon,"col"],
legend = substr(
levels(swissLandType)[[1]][mostCommon,"Category"],
1, 12)
)

table(extract(swissLandType, swissRain), exclude=NULL)


####
# SwissAltitude
###
library(raster)
download.file('http://biogeo.ucdavis.edu/data/diva/alt/CHE_alt.zip',
destfile=paste(dataDir,'CHE_alt.zip',sep=""))
swissAfile = unzip(paste(dataDir, 'CHE_alt.zip',sep=""), exdir=dataDir)
swissAltitude = raster(grep("CHE_alt.gri", swissAfile, value=TRUE))
swissAltitude = projectRaster(swissAltitude,
crs=CRS(proj4string(swissRain)))
swissAltitude = aggregate(swissAltitude,fact=2)


save(swissRain, swissAltitude, swissBorder, swissLandType,
file="~/research/diseasemapping/pkg/geostatsp/data/swissRain.RData",
compress="xz")

## End(Not run)
```

`swissRainR` *Raster of Swiss rain data*

Description

A raster image of Swiss rain and elevation, and a nearest neighbour matrix corresponding to this raster.

Usage

```
data(swissRainR)
```

Format

`swissRainR` is a RasterBrick of Swiss elevation and precipitation, and `swissNN` is a matrix of nearest neighbours.

Source

See examples

Examples

```
## Not run:
library('raster')
border = raster::getData('GADM',country='CHE',level=0)

myextent = extent(5 ,12,45,49)

theres=10
rain = raster::getData('worldclim', var='prec',
res=theres,mask=FALSE)#,lon=20,lat=47)
rain = raster::crop(rain,myextent)
plot(rain[['prec3']])
plot(border,add=TRUE)

alt = raster::getData('worldclim', var='alt',res=theres)
alt = raster::crop(alt,extent(rain))
plot(alt)
plot(border,add=TRUE)


bio = raster::getData('worldclim', var='bio',res=theres)
bio = raster::crop(bio,extent(rain))
plot(alt)
plot(border,add=TRUE)

swissRainR = rain
# rain[[c('prec2','prec7')]]
swissRainR = addLayer(swissRainR,alt)

library('geostatsp')
swissNN = NNmat(swissRainR)

save(swissRainR, swissNN,file=
"/home/patrick/workspace/diseasemapping/pkg/gmrf/data/swissRainR.RData",
compress='xz' )


## End(Not run)
data('swissRainR')
plot(swissRainR[['prec7']])

plot(swissRainR[['alt']])

swissNN[1:4,1:5]
```

`variog` *Compute Empirical Variograms and Permutation Envelopes*

Description

These are wrappers for `variog` and `variog.mc.env` in the geoR package.

Usage

```
variog(geodata, ...)
## S3 method for class 'SpatialPointsDataFrame'
variog(geodata, formula, ...)
## Default S3 method:
variogMcEnv(geodata, ...)
## S3 method for class 'SpatialPointsDataFrame'
variogMcEnv(geodata, formula, ...)
```

Arguments

geodata	An object of class SpatialPointsDataFrame or of a class suitable for variog in the geoR package.
formula	A formula specifying the response variable and fixed effects portion of the model. The variogram is performed on the residuals.
...	additional arguments passed to variog in the geoR package.

Value

As variog or variog.mc.env

See Also

variog and variog.mc.env.

Examples

```
data("swissRain")
swissRain$lograin = log(swissRain$rain)
swissv= variog(swissRain, formula=lograin ~ 1,option="bin")
swissEnv = variogMcEnv(swissRain, lograin ~ 1, obj.var=swissv,nsim=9)
if(!is.null(swissv)){
plot(swissv, env=swissEnv, main = "Swiss variogram")
}
```

---

wheat *Mercer and Hall wheat yield data*

Description

Mercer and Hall wheat yield data, based on version in Cressie (1993), p. 455.

Usage

```
data(wheat)
```

Format

wheat is a raster where the values refer to wheat yields.

Examples

```
data("wheat")
plot(wheat, main="Mercer and Hall Data")
```

Index

∗Topic **datasets**
gambiaUTM, 5
loaloa, 21
murder, 28
rongelapUTM, 38
swissRain, 43
swissRainR, 47
wheat, 49
∗Topic **models**
conditionalGmrf, 3
spatialRoc, 40
∗Topic **spatial**
RFsimulate, 36

asImRaster, 2

conditionalGmrf, 3

elevationLoa *(*loaloa*)*, 21
eviLoa *(*loaloa*)*, 21
excProb, 4, *41*

fillParam *(*matern*)*, 22

gambiaUTM, 5
glgm, *33*
glgm *(*glgm-methods*)*, 6
glgm,ANY,ANY,ANY,ANY-method *(*glgm-methods*)*, 6
glgm,formula,data.frame,Raster,data.frame-method *(*glgm-methods*)*, 6
glgm,formula,Raster,ANY,ANY-method *(*glgm-methods*)*, 6
glgm,formula,Spatial,ANY,ANY-method *(*glgm-methods*)*, 6
glgm-methods, 6
GridTopology, *36*

hessian, *34, 35*

informationLgm *(*profLlgm*)*, 34
inla, *7, 8*
inla.models, 9

krigeLgm, 10, *15, 16*, *42*

lgcp, *40, 41*
lgcp *(*glgm-methods*)*, 6
lgm, *3, 4*, *11*, *18*, *34, 35*
lgm *(*lgm-methods*)*, 12
lgm,character,ANY,ANY,ANY-method *(*lgm-methods*)*, 12
lgm,formula,data.frame,Raster,data.frame-method *(*lgm-methods*)*, 12
lgm,formula,Raster,ANY,ANY-method *(*lgm-methods*)*, 12
lgm,formula,Spatial,numeric,ANY-method *(*lgm-methods*)*, 12
lgm,formula,Spatial,Raster,data.frame-method *(*lgm-methods*)*, 12
lgm,formula,Spatial,Raster,list-method *(*lgm-methods*)*, 12
lgm,formula,Spatial,Raster,missing-method *(*lgm-methods*)*, 12
lgm,formula,Spatial,Raster,Raster-method *(*lgm-methods*)*, 12
lgm,missing,ANY,ANY,ANY-method *(*lgm-methods*)*, 12
lgm,numeric,ANY,ANY,ANY-method *(*lgm-methods*)*, 12
lgm-methods, 12
likfitLgm, *10*, *15, 16*, 17
loaloa, 21
loglikLgm *(*likfitLgm*)*, 17
ltLoa *(*loaloa*)*, 21

mask, *7*
matern, *17*, 22
maternGmrfPrec, *3*, 25
mclapply, *10*, *15*, *34*, *42*
mcmapply, *3*, *35*
modelRandomFields *(*RFsimulate*)*, 36
murder, 28

NNmat *(*maternGmrfPrec*)*, 25

optim, *15, 16*, *18*

pcPrior *(*pcPriorRange*)*, 32
pcPriorRange, 32

postExp, 33
profLlgm, 34

raster, *10*, *15*
RFfit, *37*
RFgetModelInfo, *37*
RFgui, *37*
RFoptions, *37*
RFsimulate, 36, *37*, *39*
RFsimulate,ANY,Raster-method
(RFsimulate), 36
RFsimulate,data.frame,ANY-method
(RFsimulate), 36
RFsimulate,matrix,Raster-method
(RFsimulate), 36
RFsimulate,matrix,Spatial-method
(RFsimulate), 36
RFsimulate,numeric,GridTopology-method
(RFsimulate), 36
RFsimulate,numeric,SpatialGrid-method
(RFsimulate), 36
RFsimulate,numeric,SpatialPixels-method
(RFsimulate), 36
RFsimulate,numeric,SpatialPoints-method
(RFsimulate), 36
RFsimulate,RMmodel,GridTopology-method
(RFsimulate), 36
RFsimulate,RMmodel,SpatialPoints-method
(RFsimulate), 36
RFsimulate-methods (RFsimulate), 36
RFsimulate.more.examples, *37*
RFsimulateAdvanced, *37*
RMmodel, *36*, *37*
rongelapUTM, 38

simLgcp, 39, *40*, *41*
simPoissonPP (simLgcp), 39
SpatialPixels, *36*
SpatialPoints, *36*
SpatialPointsDataFrame, *36*
spatialRoc, 40
spdfToBrick (stackRasterList), 42
squareRaster (squareRaster-methods), 41
squareRaster,Extent-method
(squareRaster-methods), 41
squareRaster,matrix-method
(squareRaster-methods), 41
squareRaster,Raster-method
(squareRaster-methods), 41
squareRaster,Spatial-method
(squareRaster-methods), 41
squareRaster-methods, 41
stackRasterList, 42
swissAltitude (swissRain), 43
swissBorder (swissRain), 43
swissLandType (swissRain), 43
swissNN (swissRainR), 47
swissRain, 43
swissRainR, 47

tempLoa (loaloa), 21
torontoBorder (murder), 28
torontoIncome (murder), 28
torontoNight (murder), 28
torontoPdens (murder), 28

variog, *48*, 48, *49*
variog.mc.env, *48*, *49*
variogMcEnv (variog), 48

wheat, 49