Package 'geostatsp'

August 4, 2014

Type Package

Title Geostatistics using SpatialPoints and rasters

Version 0.9.999

Date 2014-07-21

Depends methods, Matrix, raster, sp, R (>= 3.0.0)

Suggests pracma, geoR, RandomFields, rgdal, parallel, numDeriv

Enhances spatstat, mapmisc, ellipse

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

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

Description
This package provides to geostatistical modelling facilities using raster and SpatialPoints objects.

License GPL

Repository CRAN

Repository/R-Forge/Project diseasemapping

Repository/R-Forge/Revision 796

Repository/R-Forge/DateTimeStamp 2014-07-25 12:38:31

Date/Publication 2014-08-04 16:48:37

NeedsCompilation yes

R topics documented:

`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 not registered as an S3 method as doing so would require the spatstat package to be a dependency of geostatsp.

Examples

```
myraster = raster(matrix(1:100,10,10),
xmn=0,xmx=10,ymn=0,ymx=10)

if(any(rownames(installed.packages())=='spatstat')){
myIm = asImRaster(myraster)

}
```

---

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, newdata=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, cells=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)
```

---

krigeLgm *Spatial prediction, or Kriging*

Description

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

Usage

```
krigeLgm(formula, data, newdata,  param,
    covariates = NULL, 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
newdata	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 newdata.
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 mcmapply 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$elevation = extract(swissAltitude, swissRain)

## Not run:
swissFit = likfitLgm(data=swissRain,
trend=lograin~ elevation,
param=c(range=46500, nugget=0.05,shape=1,
anisoAngleDegrees=35, anisoRatio=12),
paramToEstimate = c("range","nugget",
"anisoAngleDegrees", "anisoRatio"),
parscale = c(range=5000,nugget=0.01,
anisoRatio=1,anisoAngleDegrees=5)
)
myTrend = swissFit$formula
myParams = swissFit$param
dput(myParams)
# will give the following

## End(Not run)
myParams=structure(c(4.49732366951939, 0.000216147764497527, 46575.4897705083,
```

```
0.0918626121620231, 37.3640527963592, 11.9247798102433, 1, 0.686396444665454
), .Names = c("(Intercept)", "elevation", "range", "nugget",
"anisoAngleDegrees", "anisoRatio", "shape", "variance"))
myTrend =lograin~ elevation


# make sure krige can cope with missing values!
swissAltitude[1:50,1:50] = NA
swissKrige = krigeLgm(data=swissRain, formula = myTrend,
covariates = swissAltitude,
param=myParams,
newdata = 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
## Not run:
swissFit2 = likfitLgm(data=swissRain,
trend=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)

## End(Not run)
myParams2=structure(c(20.7805835664033, 51161.049007108, 8.64565931800056,
37.0906650998427, 7.34077366178755, 1, 0.5, 79.3696192640485), .Names = c("(Intercept)",
"range", "nugget", "anisoAngleDegrees", "anisoRatio", "shape",
"boxcox", "variance"))
myTrend2=rain~1

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


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




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

lgm *Linear Geostatistical Models*

Description

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

Usage

```
lgm(formula, data, newdata, covariates = NULL,  shape = 1,
    fixShape = TRUE, aniso = FALSE, boxcox = 1, fixBoxcox = TRUE,
    nugget = 0, fixNugget = FALSE, expPred = FALSE, nuggetInPrediction = TRUE,
    ...)
```

Arguments

formula	A model formula for the fixed effects, or a character string specifying the response variable.
data	A SpatialPointsDataFrame containing the locations and observations, and possibly covariates.
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.
newdata	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.
shape	Order of the Matern correlation
fixShape	Set to FALSE to estimate the Matern order
aniso	Set to TRUE to use geometric anisotropy.
boxcox	Box-Cox transformation parameter, set to 1 for no transformation.
fixBoxcox	Set to FALSE to estimate the Box-Cox parameter.
nugget	Value for the nugget effect (observation error
fixNugget	Set to FALSE to estimate the nugget effect parameter.
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.
...	Additional arguments passed to likfitLgm. Starting values can be specified with a vector param of named elements

Details

Calls likfitLgm and krigeLgm

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
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.

See Also

likfitLgm, krigeLgm

Examples

```
data("swissRain")

swissRes =  lgm( formula="rain", data=swissRain, newdata=20,
covariates=swissAltitude, boxcox=0.5, fixBoxcox=TRUE,
shape=1, fixShape=TRUE,
aniso=FALSE, nugget=0, fixNugget=TRUE,
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,
coordinates=data,
param=c(range=1,nugget=0,shape=1),
upper=NULL,lower=NULL, parscale=NULL,
paramToEstimate = c("range","nugget"),
reml=TRUE)

loglikLgm(param,
data, formula, coordinates=data,
reml=TRUE,
minustwotimes=TRUE,
stored=NULL, 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 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 link{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
`stored`	A list of Box-Cox transformed observations and their Jacobian.
`moreParams`	Vector of additional parameters, combined with `param`. Used for passing fixed parameters to `loglikLgm` from within `optim`.

Value

`likfitLgm` produces list with elements

`param`	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), runif(n)),
data=data.frame(cov1 = rnorm(n), cov2 = rpois(n, 0.5))
)
```

```
# simulate a random field
trueParam = c(variance=2^2, range=0.15, shape=2, nugget=0.5^2)

mydat$U = geostatsp::RFsimulate(trueParam,mydat)$variable1

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

## Not run:
par(mfrow=c(1,2))
spplot(mydat, "U", col.regions=rainbow(10), main="U")
spplot(mydat, "Y", col.regions=rainbow(10), main="Y")
par(mfrow=c(1,1))

## End(Not run)

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



myres$summary

## Not run:

# plot variograms of data, true model, and estimated model
myv = variog(mydat, formula=Y ~ cov1 + cov2,option="bin", max.dist=0.5)
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(Y ~ cov1 + cov2, mydat,
param=c(range=0.1,nugget=0,shape=1), paramToEstimate = c("range")
)

myresNoN$summary


# plot variograms of data, true model, and estimated model
```

```
myv = variog(mydat, formula=Y ~ cov1 + cov2,option="bin", max.dist=0.5)
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$U = geostatsp::RFsimulate(trueParamAniso,mydat)$variable1

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( Y ~ cov1 + cov2, 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, c(0.5, 0.5), par=myres$param)
covTrue = matern(myraster, c(0.5, 0.5), par=trueParamAniso)

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

par(mfrow=c(1,1))

## End(Not run)
```

`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, `http://e4ftl01.cr.usgs.gov/MOLT/MOD13Q1.005/` for the EVI data, `http://e4ftl01.cr.usgs.gov/MOTA/MCD12Q1.051/` for 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)
text(768000, 800000,"land type")
commonValues  = order(levels(ltLoa)[[1]]$n,decreasing=TRUE)[1:7]
# get rid of water
commonValues = commonValues[commonValues != 1]

legend("bottomleft",
fill=levels(ltLoa)[[1]][commonValues,"col"],
legend=levels(ltLoa)[[1]][commonValues,"Category"],
bty='n'
)

points(loaloa)

# the following is the code for creating these datasets
## Not run:
dataDir = "/store/patrick/spatialData/"

loaloa = read.table(
"http://www.leg.ufpr.br/lib/exe/fetch.php/pessoais:paulojus:mbgbook:datasets:loaloa.txt",
skip=7)
names(loaloa) = c("long","lat", "N","y","elevation","ndvi","sdndvi")
library(sp)
library(rgdal)

# create projection without epsg code so rgdal doesn't need to be loaded
theproj = CRSargs(CRS("+init=epsg:3064"))
theproj = gsub("\+init=epsg:[[:digit:]]+ ", "", theproj)
theproj = CRS(theproj)

loaloaLL = SpatialPointsDataFrame(loaloa[,c("long","lat")],
data=loaloa[,c("N","y")],
```

```
proj4string = CRS("+init=epsg:4326"))
loaloa=spTransform(loaloaLL, theproj)
loaloa$villageID = seq(1, length(loaloa))



if(FALSE) {
install.packages("RCurl")
install.packages("rgeos")
install.packages('mapdata')
install.packages("MODIS", repos="http://r-forge.r-project.org")
}

# load the package and set options

library(MODIS)
MODISoptions(gdalPath="/usr/bin/", localArcPath=dataDir)
options()[grep("MODIS", names(options()), value=T)]

# EVI, get 12 month average for 2002
myProduct = "MOD13Q1"
getProduct(myProduct)

thehdf=getHdf(product=myProduct,begin="2002-01-01",end="2002-12-31",
extent=extent(spTransform(loaloa, CRS("+init=epsg:4326"))))

# the HDR files have multiple layers
# find the layer with EVI data
layerNames = getSds(thehdf[[1]][1])$SDSnames
eviLayer = grep("EVI$", layerNames)
theString = rep(0, length(layerNames))
theString[eviLayer] = 1
theString = paste(theString, collapse="")

# convert downloaded HDR files to nice tiff files
# on a 2km grid, with the same projection as loaloa
runGdal(product=myProduct,begin="2002-01-01",end="2002-12-31",
outProj = proj4string(loaloa),
pixelSize=2000, job="loa",
SDSstring = theString,
extent=extent(spTransform(loaloa, CRS("+init=epsg:4326"))))

# find names of all the tiff files
thenames = preStack(
path = paste(options()$MODIS_outDirPath, "loa",sep=""),
pattern=myProduct)
# create a raster stack of EVI for each day
eviLoa= stack(thenames)
# crop out the study region (20km buffer around the loaloa data)
eviLoa = crop(eviLoa,
extend(extent(loaloa),20000))

# compute the yearly average
```

```
eviAvg = raster::overlay(eviLoa, fun=function(x) {
mean(x, na.rm=T)
})

# plot the data
plot(eviAvg)
points(loaloa)
range(extract(eviLoa, loaloa))
# save data as an R object
eviLoa = eviAvg

##################
# land cover data
##################
myProduct = "MCD12Q1"
getProduct(myProduct)

thehdf=getHdf(product=myProduct,
begin="2001-01-01",end="2010-12-30",
extent=extent(
spTransform(loaloa, 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="2002-12-30",
outProj = proj4string(loaloa),
pixelSize=2000, job="loa",
SDSstring = theString,
extent=extent(spTransform(loaloa, CRS("+init=epsg:4326"))))

# find file name
thenames = preStack(
path = paste(options()$MODIS_outDirPath, "loa",sep=""),
pattern=myProduct)
ltLoa = raster(thenames)
ltLoa = crop(ltLoa, extend(extent(loaloa),20000))

# convert to a raster of factors
ltLoa = as.factor(ltLoa)

# get land type labels
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(ltLoa)[[1]]$Category =
labelVec[as.character(levels(ltLoa)[[1]]$ID)]

# assign colours to most common land types
commonValues  = sort(table(values(ltLoa)),decreasing=TRUE)
ctable = levels(ltLoa)[[1]]
rownames(ctable) = as.character(ctable$ID)
ctable$n = NA
ctable[names(commonValues),"n"] = commonValues

rownames(ctable) = ctable$Category

ctable$col = NA
ctable["Snow and ice","col"] = "#FFFFFF"
ctable["Water bodies","col"] = NA
ctable["Urban and built-up","col"] = "#C31400"
ctable["Croplands","col"] = "#FFFF88"
ctable["Grasslands","col"] = "#CCDD11"
ctable["Permanent Wetlands","col"] ="#99BB99"
ctable["Cropland/natural vegetation mosaic","col"] = "#CC7711"
ctable["Barren or sparsely vegetated","col"] = "#111111"


forests = grep("forest",ctable$Category,value=T,ignore.case=T)
ctable[forests,"col"] = RColorBrewer::brewer.pal(length(forests), "Greens")

savannas = grep("savannas|shrublands",ctable$Category,value=T,ignore.case=T)
ctable[savannas,"col"] = RColorBrewer::brewer.pal(length(savannas), "Oranges")

# put labels back in the raster
levels(ltLoa) = list(ctable)

# assign colours to values
forctable = rep(NA, nrow(ctable))
forctable[ctable$ID+1] = ctable$col

ltLoa@legend@colortable = forctable


# plot land type
plot(ltLoa, legend=FALSE)
legend("bottomleft",
fill=levels(ltLoa)[[1]]$col,
legend=levels(ltLoa)[[1]][
match(as.integer(names(commonValues)),
levels(ltLoa)[[1]]$ID),
"Category"]
```

```
)

############
# elevation
#############
myProduct = "SRTM"
getProduct(myProduct)
getCollection(myProduct)
# doesn't seem to be available!
# do it the hard way...

# download six tiles

theTiles=c("3811","3911","3812","3912", "4011", "4012")
if(FALSE) {
for(D in theTiles) {
D2 = paste(substr(D,1,2), "_", substr(D,3,4),sep="")
print(D2)

download.file(paste(
"ftp://srtm.csi.cgiar.org/SRTM_V41/SRTM_Data_GeoTiff/srtm_",
D2, ".zip",sep=""),
paste(dataDir, "srtm_", D2, ".zip",sep="")
)
}
}

# extent to crop to
forCrop = extend(projectExtent(
loaloa,crs="+init=epsg:4326")@extent,
0.5)

for(D in theTiles) {
print(D)
D2 = paste(substr(D,1,2), "_", substr(D,3,4),sep="")
print(D2)

unzip(paste(dataDir,"srtm_", D2, ".zip",sep=""), exdir=dataDir)
elevUnc = raster(paste(dataDir, "srtm_", D2, ".tif",sep=""))

elevC = crop(elevUnc, forCrop)

if(D == theTiles[1]) {
result = elevC
} else {
result = merge(result, elevC)
}
}

plot(result)

elevAgg=raster::aggregate(result, fact=30)
```

```
elevationLoa = projectRaster(elevAgg, crs=CRS(proj4string(loaloa)))



#################
# temperature
###################

myProduct = "MOD11C3"
thehdf=getHdf(product=myProduct,begin="2002-01-01",end="2002-12-31",
extent=extent(spTransform(loaloa, CRS("+init=epsg:4326"))))

# the HDR files have multiple layers
# find the layer with EVI data
layerNames = getSds(thehdf[[1]][1])$SDSnames
theLayer = grep("^LST_Day", layerNames)
theString = rep(0, length(layerNames))
theString[theLayer] = 1
theString = paste(theString, collapse="")

# convert downloaded HDR files to nice tiff files
# on a 2km grid, with the same projection as loaloa
runGdal(product=myProduct,begin="2002-01-01",end="2002-12-31",
SDSstring = theString, job="loa"
)

# find names of all the tiff files
thenames = preStack(
path = paste(options()$MODIS_outDirPath, "loa",sep=""),
pattern=myProduct)
# create a raster stack of EVI for each day
tempLoa= stack(thenames)
# crop out the study region (20km buffer around the loaloa data)
tempLoa = crop(tempLoa,
extend(extent(projectExtent(loaloa,crs="+init=epsg:4326")),0.5)
)
tempLoa = projectRaster(tempLoa, crs=proj4string(loaloa))

# compute the yearly average
tempAvg = raster::overlay(tempLoa, fun=function(x) {
mean(x, na.rm=T)
})

# modis says the temperatures are scaled by 0.02
# to get degrees celcius do
tempLoa = tempAvg*0.02-273.15
# plot the data
plot(tempLoa)
points(loaloa)
range(extract(tempLoa, loaloa))
```

```
# save data
save(loaloa, eviLoa, ltLoa, tempLoa, elevationLoa,
file="~/workspace/diseasemapping/pkg/geostatsp/data/loaloa.RData",
compress="xz")


## End(Not run)
```

---

matern *Evaluate the Matern correlation function*

Description

Returns the Matern covariance for the distances supplied.

Usage

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

Arguments

x	A vector or matrix of distances, or Raster or SpatialPoints of locations, see Details below.
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.
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

Details

The formula for the Matern correlation function is

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

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

```
# 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)

# plot correlation of each cell with the origin
myMatern = matern(myraster, 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 = SpatialPointsDataFrame(cbind(runif(5), runif(5)),data=data.frame(id=1:5))
matern(mypoints, param=param)


# 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("topright", fill=1:length(shapeSeq), legend=shapeSeq,title='shape')
# exponential

distVec2 = seq(0, max(distVec), len=20)
points(distVec2, exp(-2*(distVec2/range)),cex=0.5)
# gaussian
points(distVec2, exp(-2*(distVec2/range)^2), col='blue',cex=0.5)
legend("right", pch=1, col=c('black','blue'), legend=c('exp','gau'))

# comparing to geoR and RandomFields

covGeoR = covRandomFields = NULL

for(D in shapeSeq) {
covGeoR = cbind(covGeoR,
geoR::matern(distVec, phi=range/sqrt(8*D), kappa=D))
covRandomFields = cbind(covRandomFields,
RandomFields::CovarianceFct(distVec, model="matern",
param = c(mean=0, variance=1, nugget=0,
scale=range/2, nu=D) ))
}



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

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

gmrfPrecUncond(x,
N = attributes(x)$Nx, Ny=attributes(x)$Ny,
param = attributes(x)$model,
border=param["shape"]+1)
```

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.
border	number of cells from the edge of the grid to adjust for edge effects.
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'
x	An MRF precision matrix produced by maternGmrfPrec
nearest	Number of nearest neighbours to compute
...	Additional arguments passed to maternGmrfPrec.dgCMatrix

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 `dgCMatrix` 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(7, 7)[,25], 7, 7)


params=c(range = 3,
cellSize=0.5,
shape=2,
variance=5^2)

data("nn32")
# precision matrix without adjusting for edge effects
precMat =maternGmrfPrec(nn32, param=params)
## Not run:
# and with the adjustment
precMatCorr =maternGmrfPrec(nn32, param=params, adjustEdges=TRUE)

midcell = 32*16 + 16 # the middle cell
edgeCell = 32*5 + 5 # cell near corner

# show precision of cell 32,32
precMid=matrix(precMat[,midcell], 32, 32, byrow=TRUE)
precMid[seq(16-4, 16+4), seq(16-4, 16+4)]

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

```
# compare covariance matrix to the matern
xseq = seq(-20*params["cellSize"], 20*params["cellSize"], len=1000)
plot(xseq, matern(xseq, param=params),
 type = 'l',ylab='cov', xlab='dist',ylim=c(0, params["variance"]),
 main="matern v gmrf")



# middle cell
varMid=matrix(varMat[,midcell], 32, 32, byrow=TRUE)
varMidCorr=matrix(varMatCorr[,midcell], 32, 32, byrow=TRUE)
xseqMid = params["cellSize"] *seq(-16,15)
points(xseqMid, varMid[,16], col='red')
points(xseqMid, varMidCorr[,16], col='blue', cex=0.5)

# edge cells
varEdge=matrix(varMat[,edgeCell], 32, 32, byrow=TRUE)
varEdgeCorr=matrix(varMatCorr[,edgeCell], 32, 32, byrow=TRUE)
xseqEdge = params["cellSize"] *seq(-5, 26)
points(xseqEdge, varEdge[,5], pch=3,col='red')
points(xseqEdge, varEdgeCorr[,5], 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
Nx = attributes(precMat)$Nx
Ny = attributes(precMat)$Ny
cellSize = params["cellSize"]

myraster = raster(nrows=Ny, ncols=Nx,
xmn=0,ymn=0, xmx=Nx*cellSize, ymx=Ny*cellSize)
covMatMatern = matern(myraster, param=params)

  prodUncor = covMatMatern %*% precMat
  prodCor = 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)

## End(Not run)
```

murder *Murder locations*

Description

Locations of murders in Toronto 2005-2010

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

http://www.thestar.com/news/crime/torontohomicidemap.html, http://ngdc.noaa.gov/eog/viirs/download_viirs_ntl.html, http://www12.statcan.gc.ca/census-recensement/2011/geo/bound-limit/bound-limit-2011-eng.cfm, http://www12.statcan.gc.
recensement/2011/dp-pd/tbt-tt/Index-eng.cfm

Examples

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

murder2 = murder[!duplicated(coordinates(murder)),]


data("torontoPop")
plot(torontoIncome, main="Toronto Income")
points(murder2, col="#0000FF40", cex=0.5)

plot(torontoPdens, main="Toronto pop dens")
points(murder2, col="#0000FF40", cex=0.5)
```

nn32 *Nearest neighbour matrices*

Description

Some large nearest neighbour matrices.

Usage

```
data("nn32")
```

Format

Sparse matrices produced by NNmat, showing the neighbours of cells on a 64 by 64 grid in nn64, a 128 by 128 grid in nn128, and 256 in nn256.

Examples

```
data("nn32")
dim(nn32)
dim(nn64)
dim(nn128)
x = 50
y=31
matrix(nn128[,128*(y-1)+x], 128, 128)[ x+seq(-4,4),y+seq(-4, 4)]

length(nn128@x)

## Not run:
nn32 = NNmat(32)
nn64 = NNmat(64)
nn128 = NNmat(128)
#save(nn32,nn64,nn128, file=
"/home/patrick/workspace/diseasemapping/pkg/geostatsp/data/nn32.RData",
compress='xz')

## End(Not run)
```

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 = NULL, ...)
informationLgm(fit,  ...)
```

Arguments

fit	Output from the lgm function
mc.cores	Passed to mcmapply
...	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~ SRTM_1km,
newdata=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]])



}
```

RFsimulate *Simulation of Random Fields*

Description

This function simulates conditional and unconditional Gaussian random fields:

Here, only the simulation of Gaussian random fields is described. For other kind of random fields (binary, max-stable, etc.) or more sophisticated approaches see RFsimulateAdvanced.

Usage

```
RFsimulate(model, x, y=NULL, z=NULL, T=NULL, grid,
 data, distances, dim, err.model, n=1, ...)
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	vector of x coordinates, or object which can be coerced to GridTopology.
y	optional vector of y coordinates
z	optional vector of z coordinates
T	optional vector of time coordinates, T must always be an equidistant vector. Instead of T=seq(from=From, by=By, len=Len) one may also write T=c(From, By, Len).
grid	logical; RandomFields can find itself the correct value in nearly all cases. See also RFsimulateAdvanced.
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
distances	another alternative to pass the (relative) coordinates.
dim	Only used if distances are given.
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

If x is a raster, RFsimulate returns a RasterLayer or RasterBrick is returned. Otherwise a SpatialPointsDataFrame or SpatialGridDataFrame . modelRandomFields returns an object of class RMmodel.

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de> http://ms.math.uni-mannheim.de, Patrick Brown <patrick.brown@utoronto.ca> http://pbrown.ca

See Also

RFsimulate, RFempiricalvariogram, RFfit, RFgetModelInfo, RFgui, RMmodel, RFoptions, RFsimulateAdvanced, RFsimulate.more.examples

Examples

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

## Not run:
RandomFields::plot(modelRandomFields(model))

## End(Not run)

myraster = raster(nrows=60,ncols=60,xmn=0,ymn=0,xmx=10,ymx=10,
crs="+init=epsg:2081")

set.seed(0)

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


plot(simu[[2]])



## Not run:
data("swissRain")
swissRain$sqrtrain = sqrt(swissRain$rain)


# estimate parameters
swissRes =  lgm(data=swissRain, newdata=20, formula="sqrtrain",
covariates=swissAltitude,
shape=1, fixShape=TRUE,
aniso=FALSE, nugget=0, fixNugget=TRUE,
nuggetInPrediction=FALSE
)


# simulate from the random effect conditional on
#   the observed data

swissSim = geostatsp::RFsimulate(model=swissRes$param,
data=swissRes[,'resid'],
x=swissRes$predict,
n=3
)

# plot the simulated random effect
plot(swissSim[[1]])
plot(swissBorder, add=TRUE)

## End(Not run)
```

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,
  rasterTemplate=covariates[[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
`rasterTemplate`	Raster on which the latent surface is simulated, defaults to the first covariate.
`...`	additional arguments, see `GaussRF` in the RandomFields package.
`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 = 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))

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

myLgcp=simLgcp(mymodel, myCovariate, betas=c(a=-0.1, b=0.25),
rasterTemplate=myraster)
```

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

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

---

squareRaster *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

```
squareRaster(x,  cells=NULL)
## S3 method for class 'RasterLayer'
squareRaster(x,cells=NULL)
## S3 method for class 'BasicRaster'
squareRaster(x,cells=NULL)
## S3 method for class 'matrix'
squareRaster(x,cells=NULL)
## S3 method for class 'Extent'
squareRaster(x,cells=NULL)
## S3 method for class 'SpatialPoints'
squareRaster(x,cells=NULL)
## S3 method for class 'SpatialPointsDataFrame'
squareRaster(x,cells=NULL)
## S3 method for class 'SpatialPolygons'
squareRaster(x,cells=NULL)
## S3 method for class 'SpatialPolygonsDataFrame'
squareRaster(x,cells=NULL)
```

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.

Value

A raster.

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)
```

Arguments

`x`	A list of rasters
`template`	A raster whose projection and resolution all other rasters will be aligned with. Defaults to the first raster in `x`
`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, `mcmapply` is used with this argument specifying the number of cores.

Value

A raster stack.

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') {
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 of the sic.100 data, with data column `rain` being the rainfall values in millimetres. `swissAltitude` is a raster of elevation data, and `swissLandType` is a raster of land cover types.

Source

ftp://ftp.geog.uwo.ca/SIC97 and http://srtm.csi.cgiar.org and https://lpdaac.usgs.gov/products/modis_products_table/mcd12q1

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 = "/store/patirck/spatialData/"
download.file("http://mldata.org/repository/data/download/spat-interp-comparison-1997/",
destfile=paste(dataDir, "swiss.zip",sep=""))
unzip(paste(dataDir, 'swiss.zip',sep=""), exdir=dataDir)
swissRain = read.table(paste(dataDir, "sic_obs.dat",sep=""),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 = gsub("\+init=epsg:[[:digit:]]+ ", "", theproj)
theproj = CRS(theproj)

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



#######
```

```
# Swiss Border
#######

load(url("http://biogeo.ucdavis.edu/data/gadm2/R/CHE_adm0.RData"))
library(rgdal)
swissBorder = spTransform(gadm, 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",
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,
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=""))
unzip(paste(dataDir, 'CHE_alt.zip',sep=""), exdir=dataDir)
swissAltitude = raster(paste(dataDir, "CHE_alt.gri",sep=""))
swissAltitude = projectRaster(swissAltitude,
crs=CRS(proj4string(swissRain)))
swissAltitude = aggregate(swissAltitude,fact=2)


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

## End(Not run)
```

---

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:
variog.mc.env(geodata, ...)
## S3 method for class 'SpatialPointsDataFrame'
variog.mc.env(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 = variog.mc.env(swissRain, lograin ~ 1, obj.var=swissv,nsim=9)
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