Package 'gmGeostats'

October 3, 2020

Version 0.10-7

Date 2020-09-28

Title Geostatistics for Compositional Analysis

Depends R (>= 2.10)

Suggests FNN, JADE, jointDiag, DescTools, knitr, rmarkdown (>= 2.3), magrittr, readxl

Imports methods, gstat, compositions (>= 2.0), sp, boot, foreach, utils, RColorBrewer

Description Support for geostatistical analysis of multivariate data, in particular data with restrictions, e.g. positive amounts data, compositional data, distributional data, microstructural data, etc. It includes descriptive analysis and modelling for such data, both from a two-point Gaussian perspective and multipoint perspective. The methods mainly follow Tolosana-Delgado, Mueller and van den Boogaart (2018) <doi:10.1007/s11004-018-9769-3>.

License CC BY-SA 4.0 | GPL (>= 2)

URL https://gitlab.hzdr.de/geomet/gmGeostats

Copyright (C) 2020 by Helmholtz-Zentrum Dresden-Rossendorf and Edith Cowan University; gsi.DS code by Hassan Talebi

RoxygenNote 7.1.1

Encoding UTF-8

VignetteBuilder knitr

LazyData true

Collate 'Anamorphosis.R' 'compositionsCompatibility.R' 'gstatCompatibility.R' 'variograms.R' 'gmValidationStrategy.R' 'gmAnisotropy.R' 'abstractClasses.R' 'accuracy.R' 'data.R' 'exploratools.R' 'genDiag.R' 'geostats.R' 'gmDataFrameStack.R' 'gmSimulation.R' 'gmSpatialMethodParameters.R' 'gmSpatialModel.R' 'grids.R' 'mask.R' 'spSupport.R' 'uncorrelationTest.R' 'zzz.R'

NeedsCompilation yes

Author Raimon Tolosana-Delgado [aut] (<https://orcid.org/0000-0001-9847-0462>),
Ute Mueller [aut],
K. Gerald van den Boogaart [ctb, cre],
Hassan Talebi [ctb, cph],
Helmholtz-Zentrum Dresden-Rossendorf [cph],
Edith Cowan University [cph]

Maintainer K. Gerald van den Boogaart <support@boogaart.de>

Repository CRAN

Date/Publication 2020-10-03 15:20:05 UTC

R topics documented:

`+.gmCgram` *Combination of gmCgram variogram structures*

Description

combination of nested structures of a gmCgram object

Usage

```
## S3 method for class 'gmCgram'
x + y
```

Arguments

x	gmCgram variogram object
y	gmCgram variogram object

Value

The combined nested structures

Examples

```
utils::data("variogramModels")
v1 = setCgram(type=vg.Gau, sill=diag(2), anisRanges = 3*diag(c(3,1)))
v2 = setCgram(type=vg.Exp, sill=0.3*diag(2), anisRanges = 0.5*diag(2))
vm = v1+v2
```

---

accuracy *Compute accuracy and precision*

Description

Computes goodness-of-fit measures (accuracy, precision and joint goodness) adapted or extended from the definition of Deutsch (1997).

Usage

```
accuracy(x, ...)

## S3 method for class 'data.frame'
accuracy(
  x,
  observed = x$observed,
  prob = seq(from = 0, to = 1, by = 0.05),
  method = "kriging",
  outMahalanobis = FALSE,
  ...
)

## S3 method for class 'DataFrameStack'
accuracy(
  x,
  observed,
  vars = intersect(colnames(observed), dimnames(x)[[noStackDim(x)]]),
  prob = seq(from = 0, to = 1, by = 0.05),
  method = ifelse(length(vars) == 1, "simulation", "Mahalanobis"),
  outMahalanobis = FALSE,
  ...
)
```

Arguments

x	data container for the predictions (plus cokriging error variances/covariance) or simulations (and eventually for the true values in univariate problems)
...	generic functionality, currently ignored
observed	either a vector- or matrix-like object of the true values
prob	sequence of cutoff probabilities to use for the calculations
method	which method was used for generating the predictions/simulations? one of c("kriging", "cokriging", "simulation") for x of class "data.frame", or of c("simulation", "mahalanobis", "flow") for x of class DataFrameStack().
outMahalanobis	if TRUE, do not do the final accuracy calculations and return the Mahalanobis norms of the residuals; if FALSE do the accuracy calculations
vars	in multivariate cases, a vector of names of the variables to analyse

Details

For method "kriging", x must contain columns with names including the string "pred" for predictions and "var" for the kriging variance; the observed values can also be included as an extra column with name "observed", or else additionally provided in argument observed. For method "cokriging", the columns of x must contain predictions, cokriging variances and cokriging covariances in columns including the strings "pred", "var" resp. "cov", and observed values can only be provided via observed argument. Note that these are the natural formats when using gstat::predict.gstat() and other (co)kriging functions of that package.

For univariate and multivariate cokriging results (methods "kriging" and "cokriging"), the coverage values are computed based on the Mahalanobis square error, the (square) distance between prediction and true value, using as the positive definite bilinear form of the distance the variance-covariance cokriging matrix. The rationale is that, under the assumption that the random field is Gaussian, the distribution of this Mahalanobis square error is known to follow a $\chi^2(\nu)$ with degrees of freedom ν equal to the number of variables. Having this reference distribution allows us to compute confidence intervals for that Mahalanobis square error, and then count how many of the actually observed errors are included on each one of the intervals (the *coverage*). For a perfect adjustment to the distribution, the plot of coverage vs. nominal confidence (see plot.accuracy) should fall on the $y = x$ line. NOTE: the original definition of Deutsch (1997) for univariate case did not make use of the $\chi^2(1)$ distribution, but instead derived the desired intervals (symmetric!) from the standard normal distribution appearing by normalizing the residual with the kriging variance; the result is the same.

For method "simulation" and object x is a data.frame, the variable names containing the realisations must contain the string "sim", and observed must be a vector with as many elements as rows has x. If x is a DataFrameStack(), then it is assumed that the stacking dimension is running through the realisations; the true values must still be given in observed. In both cases, the method is based on ranks: with them we can calculate which is the frequency of simulations being more extreme than the observed value. This calculation is done considering bilateral intervals around the median of (realisations, observed value) for each location separately.

Method "mahalanobis" ("Mahalanobis" also works) is the analogous for multivariate simulations. It only works for x of class DataFrameStack(), and requires the stacking dimension to run through the realisations and the other two dimensions to coincide with the dimensions of observed, i.e. giving locations by rows and variables by columns. In this case, a covariance matrix will be computed

and this will be used to compute the Mahalanobis square error defined above in method "cokriging": this Mahalanobis square error will be computed for each simulation and for the true value. The simulated Mahalanobis square errors will then be used to generate the reference distribution with which to derive confidence intervals.

Finally, highly experimental "flow" method requires the input to be in the same shape as method "mahalanobis". The method is mostly the same, just that before the Mahalanobis square errors are computed a location-wise flow anamorphosis (`ana()`) is applied to transform the realisations (including the true value as one of them) to joint normality. The rest of the calculations are done as if with method "mahalanobis".

Value

If `outMahalanobis=TRUE` (the primary use), this function returns a two-column dataset of class c("accuracy", "data.frame"), which first column gives the nominal probability cutoffs used, and the second column the actual coverage of the intervals of each of these probabilities. If `outMahalanobis=FALSE`, the output is a vector (for prediction) or matrix (for simulation) of Mahalanobis error norms.

Methods (by class)

- `data.frame`: Compute accuracy and precision
- `DataFrameStack`: Compute accuracy and precision

See Also

Other accuracy functions: `mean.accuracy()`, `plot.accuracy()`, `precision()`, `validate()`, `xvErrorMeasures()`

`ana` *Flow anamorphosis transform Compute a transformation that gaussianizes a certain data set*

Description

Flow anamorphosis transform Compute a transformation that gaussianizes a certain data set

Usage

```
ana(Y, sigma0 = 0.1, sigma1 = 1, steps = 30, sphere = TRUE, weights = NULL)
```

Arguments

`Y`	data set defining the gaussianization
`sigma0`	starting spread of the kernels
`sigma1`	final spread of the kernels
`steps`	number of steps to linearize the transform (default 30 is good)
`sphere`	boolean, should the transform include a spherification step, making `Y` spherical?
`weights`	weights to incorporate in the compuations, length=nrow(Y)

Value

a function with arguments `(x,inv=FALSE)`, where `x` will be the data to apply the transformation to, and `inv=FALSE` will indicate if the direct or the inverse transformation is desired

Author(s)

K. Gerald van den Boogaart

See Also

anaForward, anaBackward, sphTrans

Examples

```
library(compositions)
data("jura", package="gstat")
Y = acomp(jura.pred[,c(10,12,13)])
plot(Y)
anafun = ana(Y)
class(anafun)
z = anafun(Y)
plot(z)
y = anafun(z, inv=TRUE)
plot(data.frame(orig=Y,recalc=y))
```

`anaBackward` *Backward gaussian anamorphosis backward transformation to multivariate gaussian scores*

Description

Backward gaussian anamorphosis backward transformation to multivariate gaussian scores

Usage

```
anaBackward(
  x,
  Y,
  sigma0,
  sigma1 = 1 + sigma0,
  steps = 30,
  plt = FALSE,
  sphere = TRUE,
  weights = NULL
)
```

Arguments

x	matrix of gaussian scores to be back-transformed
Y	node points defining the transformation (a matrix, same nr of columns)
sigma0	starting spread of the kernels in the forward transform
sigma1	final spread of the kernels in the forward transform
steps	number of steps to linearize the transform (default 30 is good)
plt	boolean, do you want to get a plot of the transformation?
sphere	boolean, should the data be taken as pre-Y-spherified? defaults to true
weights	vector of weights for all computations, length must be equal to number of rows of x

Value

a matrix with the scores back-transformed to the same scale as Y; same dimensions of x

Author(s)

K. Gerald van den Boogaart, Raimon Tolosana-Delgado

See Also

ana() for defining a function that carries over the transformation (by means of a closure), anaBackward() for the explicit back-transformation, sphTrans() for defining a function that carries over the spherification of the data

Examples

```
data("jura", package="gstat")
Y = jura.pred[,c(10,12,13)]
plot(compositions::acomp(Y))
Ylr = compositions::alr(Y)
Xns = matrix(rnorm(500), ncol=2)
plot(Ylr)
points(Xns, col=2, pch=4)
Xlr = anaBackward(x=Xns, Y=Ylr, sigma0=0.1)
qqplot(Xlr[,1], Ylr[,1])
qqplot(Xlr[,2], Ylr[,2])
qqplot(Xlr[,1]+Xlr[,2], Ylr[,1]+Ylr[,2])
```

anaForward *Forward gaussian anamorphosis forward transformation to multivariate gaussian scores*

Description

Forward gaussian anamorphosis forward transformation to multivariate gaussian scores

Usage

```
anaForward(
  x,
  Y,
  sigma0,
  sigma1 = 1 + sigma0,
  steps = 30,
  plt = FALSE,
  sphere = TRUE,
  weights = NULL
)
```

Arguments

x	points to be transformed (a matrix)
Y	node points defining the transformation (another matrix, same nr. of columns as x)
sigma0	starting spread of the kernels
sigma1	final spread of the kernels
steps	number of steps to linearize the transform (default 30 is good)
plt	boolean, do you want to get a plot of the transformation?
sphere	boolean, should the data be pre-Y-spherified first? defaults to true
weights	vector of weights for all computations, length must be equal to number of rows of x

Value

a matrix with the gaussian scores; same dimensions of x

Author(s)

K. Gerald van den Boogaart, Raimon Tolosana-Delgado

See Also

ana() for defining a function that carries over the transformation (by means of a closure), anaBackward() for the explicit back-transformation, sphTrans() for defining a function that carries over the spherification of the data

Examples

```
data("jura", package="gstat")
Y = jura.pred[,c(10,12,13)]
plot(compositions::acomp(Y))
Ylr = compositions::alr(Y)
plot(Ylr)
z = anaForward(x=Ylr, Y=Ylr, sigma0=0.1)
plot(z, asp=1)
shapiro.test(z[,1])
shapiro.test(z[,2])
```

---

anis2D.par2A *Produce anisotropy matrix from angle and anisotropy ratios*

Description

Produce anisotropy matrix (in Cholesky decomposition) from angle and anisotropy ratios

Usage

```
anis2D.par2A(ratio, angle)
```

Arguments

ratio	an anisotropy ratio (min/max range)
angle	direction of maximum range, i.e. largest spatial continuity, measured counter-clockwise from East

Value

a 3x3 matrix of anisotropy, in Cholesky form

---

as.AnisotropyScaling *Convert to anisotropy scaling matrix*

Description

Convert an anisotropy specification to a scaling matrix

Usage

```
as.AnisotropyScaling(x)

## S3 method for class 'AnisotropyScaling'
as.AnisotropyScaling(x)

## S3 method for class 'numeric'
as.AnisotropyScaling(x)
```

Arguments

x	an object convertible to an anisotropy scaling matrix; see details

Details

Method `as.AnisotropyScaling.numeric()` expects a vector of two numbers in 2D, or a vector of 5 numbers in 3D. These are in 2D, the azimuth of maximum continuity (in degrees, clockwise from North) and the anisotropy ratio of short/long range. In 3D these are: 1,2) the azimuth and the dip of the direction of maximal continuity; 3) the angle of rotation around the axis of the first direction; 4,5) the anisotropy ratios of the ranges of the second/first and third/first directions of maximal continuity. All angles are given in degrees, all ratios must be smaller or equal to 1.

Value

A matrix A such that for any lag vector h, the variogram model turns isotropic in terms of $u = A \cdot h$.

Methods (by class)

- `AnisotropyScaling`: Convert to anisotropy scaling matrix
- `numeric`: Convert to anisotropy scaling matrix

Examples

```
as.AnisotropyScaling(c(30,0.5))
```

---

as.array.DataFrameStack

Convert a stacked data frame into an array

Description

Recast a `DataFrameStack()` into a named array.

Usage

```
## S3 method for class 'DataFrameStack'
as.array(x, ...)
```

Arguments

x	input `DataFrameStack()`
...	generic consistency

Value

the data recasted as an array with appropriate names.

Examples

```
ll = lapply(1:3, function(i) as.data.frame(matrix(1:10+(i-1)*10, ncol=2)))
dfs = DataFrameStack(ll, stackDimName="rep")
as.array(dfs)
```

---

as.CompLinModCoReg *Recast a model to the variogram model of package "compositions"*

Description

Recast a variogram model specified in any of the models of "gstat" or "gmGeostats" in the format of `compositions::CompLinModCoReg()`

Usage

```
as.CompLinModCoReg(v, ...)
```

Arguments

v	variogram model object to convert
...	further parameters for generic functionality

Value

The variogram model recast to "CompLinModCoReg"

---

as.function.gmCgram *Convert a gmCgram object to an (evaluable) function*

Description

Evaluate a gmCgram on some h values, or convert the gmCgram object into an evaluable function

Usage

```
## S3 method for class 'gmCgram'
as.function(x, ...)

## S3 method for class 'gmCgram'
predict(object, newdata, ...)
```

Arguments

x	a gmCgram object
...	extra arguments for generic functionality
object	gmCgram object
newdata	matrix, data.frame or Spatial object containing coordinates

Value

a function that can be evaluated normally, with an argument X and possibly another argument Y; both must have the same number of columns than the geographic dimension of the variogram (i.e. dim(x$M)[3]).

Functions

- predict.gmCgram: predict a gmCgram object on some lag vector coordinates

See Also

Other gmCgram functions: [.gmCgram(), [[.gmCgram(), as.gmCgram.variogramModelList(), length.gmCgram(), ndirections(), plot.gmCgram(), variogramModelPlot()

Examples

```
utils::data("variogramModels")
v1 = setCgram(type=vg.Gau, sill=diag(2)+0.5, anisRanges = 2*diag(c(3,0.5)))
v2 = setCgram(type=vg.Exp, sill=0.3*diag(2), anisRanges = 0.5*diag(2))
vm = v1+v2
vgf = as.function(vm)
(h = rbind(c(0,1), c(0,0), c(1,1)))
vgf(h)
predict(vm, h)
```

---

as.gmCgram.variogramModelList

Convert theoretical structural functions to gmCgram format

Description

Convert covariance function or variogram models to the format gmCgram of package gmGeostats

Usage

```
## S3 method for class 'variogramModelList'
as.gmCgram(m, ...)

as.gmCgram(m, ...)

## Default S3 method:
as.gmCgram(m, ...)
```

Arguments

m	model to be converted
...	further parameters

Value

the covariance/variogram model, recasted to class gmCgram. This is a generic function. Methods exist for objects of class LMCAnisCompo (for compositional data) and variogramModelList (as provided by package gstat).

Methods (by class)

- variogramModelList: Convert theoretical structural functions to gmCgram format
- default: Convert theoretical structural functions to gmCgram format

See Also

Other gmCgram functions: [.gmCgram(), [[.gmCgram(), as.function.gmCgram(), length.gmCgram(), ndirections(), plot.gmCgram(), variogramModelPlot()

---

as.gmEVario *Convert empirical structural function to gmEVario format*

Description

Convert empirical covariance functions or variograms to the format gmEVario of package gmGeostats

Usage

```
as.gmEVario(vgemp, ...)
```

Arguments

vgemp	variogram/covariance function to be converted
...	further parameters

Value

the empirical covariance function or variogram, recasted to class gmEVario. This is a generic function. Methods exist for objects of class logratioVariogramlogratioVariogramAnisotropy (for compositional data) and gstatVariogram (from package gstat).

See Also

Other gmEVario functions: gsi.EVario2D(), ndirections(), plot.gmEVario(), variogramModelPlot()

`as.gmSpatialModel` *Recast spatial object to gmSpatialModel format*

Description

Recast a spatial data object model to format gmSpatialModel

Usage

```
as.gmSpatialModel(object, ...)

## Default S3 method:
as.gmSpatialModel(object, ...)

## S3 method for class 'gstat'
as.gmSpatialModel(object, V = NULL, ...)
```

Arguments

`object`	object to recast
`...`	extra parameters for generic functionality
`V`	optional, if the original data in the sptail object was compositional, which log-contrasts were used to express it? Thsi can be either one string of "alr", "ilr" or "clr", or else a (Dx(D-1))-element matrix of logcontrasts to pass from compositions to logratios

Value

The same spatial object re-structured as a "gmSpatialModel", see gmSpatialModel

Methods (by class)

- `default`: Recast spatial object to gmSpatialModel format
- `gstat`: Recast spatial object to gmSpatialModel format

See Also

Other gmSpatialModel: `gmSpatialModel-class`, `make.gmCompositionalGaussianSpatialModel()`, `make.gmCompositionalMPSSpatialModel()`, `make.gmMultivariateGaussianSpatialModel()`, `predict.gmSpatialModel()`

as.gstat *Convert a regionalized data container to gstat*

Description

Convert a regionalized data container to a "gstat" model object

Usage

```
as.gstat(object, ...)

## Default S3 method:
as.gstat(object, ...)
```

Arguments

object	regionalized data container
...	accessory parameters (currently not used)

Value

A regionalized data container of class "gstat", eventually with variogram model included. See gstat::gstat() for more info.

Functions

- as.gstat.default: default does nothing

Examples

```
data("jura", package = "gstat")
X = jura.pred[,1:2]
Zc = jura.pred[,7:13]
gg = make.gmCompositionalGaussianSpatialModel(Zc, X, V="alr", formula = ~1)
as.gstat(gg)
```

as.gstatVariogram *Represent an empirical variogram in "gstatVariogram" format*

Description

Represent an empirical variogram in "gstatVariogram" format, from package "gstat"; see gstat::variogram() for details.

Usage

```
as.gstatVariogram(vgemp, ...)

## Default S3 method:
as.gstatVariogram(vgemp, ...)

## S3 method for class 'gmEVario'
as.gstatVariogram(vgemp, ...)

## S3 method for class 'logratioVariogram'
as.gstatVariogram(
  vgemp,
  V = NULL,
  dir.hor = 0,
  dir.ver = 0,
  prefix = NULL,
  ...
)

## S3 method for class 'logratioVariogramAnisotropy'
as.gstatVariogram(vgemp, V = NULL, ...)
```

Arguments

vgemp	empirical variogram of any kind
...	further parameters (for generic functionality)
V	eventually, indicator of which logratio should be used (one of: a matrix of log-contrasts, or of the strings "ilr", "alr" or "clr")
dir.hor	eventually, which horizontal direction is captured by the variogram provided (seldom to be touched!)
dir.ver	eventually, which vertical direction is captured by the variogram provided (seldom to be touched!)
prefix	prefix name to use for the variables created (seldom needed)

Value

The function returns an object of class "gstatVariogram" containing the empirical variogram provided. See `gstat::variogram()` for details.

Methods (by class)

- `default`: Represent an empirical variogram in "gstatVariogram" format
- `gmEVario`: Represent an empirical variogram in "gstatVariogram" format
- `logratioVariogram`: Represent an empirical variogram in "gstatVariogram" format
- `logratioVariogramAnisotropy`: Represent an empirical variogram in "gstatVariogram" format

Examples

```
data("jura", package = "gstat")
X = jura.pred[,1:2]
Zc = compositions::acomp(jura.pred[,7:13])
lrvg = gmGeostats::logratioVariogram(data=Zc, loc=X)
as.gstatVariogram(lrvg, V="alr")
```

---

as.list.DataFrameStack *Convert a stacked data frame into a list of data.frames*

Description

Recast a DataFrameStack() into a list of data.frames

Usage

```
## S3 method for class 'DataFrameStack'
as.list(x, ...)
```

Arguments

x	input DataFrameStack()
...	generic consistency

Value

the data recasted as list of data.frames

Examples

```
ar = array(1:30, dim = c(5,2,3), dimnames=list(obs=1:5, vars=c("A","B"), rep=1:3))
dfs = DataFrameStack(ar, stackDim="rep")
as.list(dfs)
```

---

as.LMCAnisCompo *Recast compositional variogram model to format LMCAnisCompo*

Description

Recast a compositional variogram model of any sort to a variation-variogram model of class "LMCAnisCompo".

Usage

```
as.LMCAnisCompo(m, ...)

## S3 method for class 'LMCAnisCompo'
as.LMCAnisCompo(m, ...)

## S3 method for class 'gmCgram'
as.LMCAnisCompo(m, V = NULL, orignames = rownames(V), ...)

## S3 method for class 'CompLinModCoReg'
as.LMCAnisCompo(m, varnames, ...)

## S3 method for class 'gstat'
as.LMCAnisCompo(m, ...)

## S3 method for class 'variogramModelList'
as.LMCAnisCompo(m, V = NULL, orignames = NULL, ...)
```

Arguments

m	original variogram model
...	arguments for generic functionality
V	eventually, a specification of the way m is presently represented
orignames	eventually, vector of names of the components, if V is provided and it does not have rownnames
varnames	a vector with the component names

Value

the variogram model recasted to class "LMCAnisCompo"

Methods (by class)

- LMCAnisCompo: Recast compositional variogram model to format LMCAnisCompo
- gmCgram: Recast compositional variogram model to format LMCAnisCompo
- CompLinModCoReg: Recast a variogram model from package "compositions" to format LMCAnisCompo
- gstat: Recast compositional variogram model to format LMCAnisCompo
- variogramModelList: Recast compositional variogram model to format LMCAnisCompo

as.logratioVariogram *Recast empirical variogram to format logratioVariogram*

Description

Recast an empirical compositional variogram of any sort to a variation-variogram of class "logratioVariogram".

Usage

```
as.logratioVariogram(vgemp, ...)
```

Arguments

vgemp	empirical variogram
...	parameters for generic functionality

Value

the same model in the new format.

as.logratioVariogramAnisotropy *Convert empirical variogram to "logratioVariogramAnisotropy"*

Description

Convert an empirical variogram from any format to class "logratioVariogramAnisotropy"

Usage

```
as.logratioVariogramAnisotropy(vgemp, ...)

## Default S3 method:
as.logratioVariogramAnisotropy(vgemp, ...)

## S3 method for class 'logratioVariogram'
as.logratioVariogramAnisotropy(vgemp, ...)

## S3 method for class 'logratioVariogramAnisotropy'
as.logratioVariogramAnisotropy(vgemp, ...)
```

Arguments

vgemp	an empirical variogram
...	further parameters

Value

The empirical variogram as a "logratioVariogramAnisotropy" object

Methods (by class)

- `default`: default method, making use of `as.logratioVariogram()`
- `logratioVariogram`: method for "logratioVariogram" class
- `logratioVariogramAnisotropy`: identity transformation

`as.variogramModel` *Convert an LMC variogram model to gstat format*

Description

Convert a linear model of coregionalisation to the format of package gstat. See `gstat::vgm()` for details.

Usage

```
as.variogramModel(m, ...)

## Default S3 method:
as.variogramModel(m, ...)

## S3 method for class 'gmCgram'
as.variogramModel(m, ...)

## S3 method for class 'LMCAnisCompo'
as.variogramModel(m, V = NULL, prefix = NULL, ensurePSD = TRUE, ...)

## S3 method for class 'CompLinModCoReg'
as.variogramModel(m, V = "alr", prefix = NULL, ensurePSD = TRUE, ...)
```

Arguments

`m`	variogram model
`...`	further arguments for generic functionality
`V`	eventually, specification of the logratio representation to use for compositional data (one of: a matrix of log-contrasts to use, or else one of the strings "alr", "clr" or "ilr")
`prefix`	optional, name prefix for the generated variables if a transformation is used
`ensurePSD`	logical, should positive-definiteness be enforced? defaults to TRUE, which may produce several scary looking but mostly danger-free warnings

Value

The LMC model specified in the format of package gstat, i.e. as the result of using `gstat::vgm()`

Methods (by class)

- `default`: Convert an LMC variogram model to gstat format
- `gmCgram`: Convert an LMC variogram model to gstat format
- `LMCAnisCompo`: Convert an LMC variogram model to gstat format
- `CompLinModCoReg`: Convert an LMC variogram model to gstat format

Examples

```
data("jura", package = "gstat")
X = jura.pred[,1:2]
Zc = compositions::acomp(jura.pred[,7:13])
lrmd = compositions::CompLinModCoReg(formula=~nugget()+sph(1.5), comp=Zc)
as.variogramModel(lrmd, V="alr")
```

CholeskyDecomposition *Create a parameter set specifying a LU decomposition simulation algorithm*

Description

Create a parameter set describing a Cholesky (or LU) decomposition algorithm to two-point simulation, mostly for covariance or variogram-based gaussian random fields.

Usage

```
CholeskyDecomposition(nsim = 1, ...)
```

Arguments

`nsim`	number of realisations desired
`...`	further parameters, currently ignored

Value

an S3-list of class "gmCholeskyDecomposition" containing the few elements given as arguments to the function. This is just a compact way to provide further functions such as `predict.gmSpatialModel()` with appropriate triggers for choosing a prediction method or another, in this case for triggering LU or Cholesky decomposition simulation.

Examples

```
(chols_local = CholeskyDecomposition(nsim=100, nBands=300))
## then run predict(..., pars=chols_local)
```

coloredBiplot.genDiag *Colored biplot for gemeralised diagonalisations Colored biplot method for objects of class genDiag*

Description

Colored biplot for gemeralised diagonalisations Colored biplot method for objects of class genDiag

Usage

```
## S3 method for class 'genDiag'
coloredBiplot(x, choices = 1:2, scale = 0, pc.biplot = FALSE, ...)
```

Arguments

x	a generalized diagonalisation object, as obtained from a call to Maf (or to UWEDGE or RJD, on the help page of Maf).
choices	which factors should be represented? vector of 2 indices; defaults to c(1,2)
scale	deprecated, kept for coherence with link{biplot.princomp}
pc.biplot	same as the homonimous argument from link{biplot.princomp}: boolean, to scale variables down by sqrt(n) and observations up by the same factor.
...	further arguments to coloredBiplot

Value

nothing. Function is called exclusively to produce the plot

References

Mueller, Tolosana-Delgado, Grunsky and McKinley (2021) Biplots for Compositional Data Derived from Generalised Joint Diagonalization Methods. Applied Computational Geosciences (under review)

See Also

Other generalised Diagonalisations: Maf(), predict.genDiag()

Examples

```
data("jura", package="gstat")
juracomp = compositions::acomp(jura.pred[, -(1:6)])
lrvg = logratioVariogram(data=juracomp, loc=jura.pred[,1:2])
mf = Maf(juracomp, vg=lrvg)
mf
compositions::coloredBiplot(mf, xlabs.col=as.integer(jura.pred$Rock)+2)
```

constructMask *Constructs a mask for a grid*

Description

Constructs a mask for a grid

Usage

```
constructMask(grid, method = "maxdist", maxval = NULL, x = NULL)
```

Arguments

grid	a grid, see details for more info
method	which construction method? currently one of 'maxdist', 'sillprop' or 'point2polygon'
maxval	for maxdist and sillprop methods, maximum reference value
x	extra information for the grid construction, see details

Details

Method 'maxdist' defines the mask as all points within a maximum distance (must be given in maxval) from the reference data (given in x: this is expected to be the original complete data, with coordinates and variables). For method 'sillprop' the mask is defined by those points which total kriging variance is below a fixed proportion (given in maxval, default=0.99) of the total variogram model sill (variogram model given in x, of class "variogramModelList"). In this method, the argument grid is expected to be the output of a cokriging analysis. Finally, method 'point2poly' created the mask by taking the points internal to a "SpatialPolygon" object (given in x).

Value

a logical vector with as many elements as points in the grid, with TRUE for those points within the mask, and FALSE for those outside the mask.

See Also

Other masking functions: getMask(), print.mask(), setMask(), unmask()

Examples

```
## with data.frame
x = 1:23
y = 1:29
xy = expand.grid(x=x, y=y)
xyz.df = data.frame(xy, z = rnorm(29*23)*ifelse(abs(xy$x-xy$y)<3, 1, NA)+(xy$x+xy$y)/2)
mask.df = constructMask(grid = xy, method = "maxdist", maxval = 3, x=xyz.df)
image(mask.df)
par(mfrow=c(1,1))
mask.df
```

```
xyz.df.masked = setMask(xyz.df, mask.df)
dim(xyz.df.masked)
summary(xyz.df.masked)
xyz.df.unmasked = unmask(xyz.df.masked)
dim(xyz.df.unmasked)
length(x)*length(y)
summary(xyz.df.unmasked)
## with SpatialGrid
library(sp)
library(magrittr)
xy.sp = sp::SpatialPoints(coords = xy)
meandiff = function(x) mean(diff(x))
xy.gt = GridTopology(c(min(x),min(y)), c(meandiff(x), meandiff(y)), c(length(x),length(y)))
aux = sp::SpatialPixelsDataFrame(grid = xy.gt, data=xyz.df, points = xy.sp)
xyz.sgdf = as(aux, "SpatialGridDataFrame")
image_cokriged(xyz.sgdf, ivar="z")
par(mfrow=c(1,1))
ms = function(x) sortDataInGrid(x, grid=xy.gt)
mask.gt = constructMask(grid = xy.gt, method = "maxdist", maxval = 3, x=xyz.sgdf)
image(x,y,matrix(ms(xyz.sgdf@data$z), nrow=23, ncol=29))
image(x,y,matrix(ms(mask.gt), nrow=23, ncol=29))
image(mask.gt)
par(mfrow=c(1,1))
xyz.sgdf.masked = setMask(x = xyz.sgdf, mask = mask.gt)
getMask(xyz.sgdf.masked)
image(x,y,matrix(ms(xyz.sgdf@data$z), nrow=23, ncol=29))
points(xyz.sgdf.masked@coords)
```

`DataFrameStack` *Create a data frame stack*

Description

Make a stacked data frame, that is, a stack of data.frames representing e.g. repeated measurements, parallel time series, or a stack of multivariate realisation of some random process. If a data frame is analogous to a matrix, a DataFrameStack is analogous to an array. It is highly recommendable to work with named dimensions in stacked data frames.

Usage

```
DataFrameStack(x, ...)

as.DataFrameStack(x, ...)

## S3 method for class 'data.frame'
DataFrameStack(x, stackDim = 2, dim = NULL, Dimnames = NULL, ...)

## S3 method for class 'data.frame'
as.DataFrameStack(x, stackDim = 2, dim = NULL, Dimnames = NULL, ...)
```

```
## S3 method for class 'array'
DataFrameStack(x, stackDim = 2, ...)

## S3 method for class 'array'
as.DataFrameStack(x, stackDim = 2, ...)

## S3 method for class 'list'
DataFrameStack(x, stackDimName = NULL, Dimnames = NULL, ...)

## S3 method for class 'list'
as.DataFrameStack(x, stackDimName = NULL, Dimnames = NULL, ...)
```

Arguments

x	object containing the individual data sets, this can be an arra, a list or a data.frame
...	further parameters, ignored but necessary for generic functionality
stackDim	for "array" and "data.frame" input, which dimension (name or index) identifies the stacking dimension; this is typically the replication, realisation or time slice index
dim	for "data.frame" input, how is the data provided to be arranged in slices?
Dimnames	for "list" and "data.frame input, which names do you want to give to the resulting array-like structure; note that for input "array" it is necessary that these names are already given to the input array beforehand!
stackDimName	for "list" input, which name or index do you want to give to the stacking dimension?

Value

The data provided reorganised as a DataFrameStack, with several additional attributes.

Functions

- DataFrameStack: create a DataFrameStack from an array
- as.DataFrameStack: create a DataFrameStack from an array
- as.DataFrameStack.data.frame: create a DataFrameStack from an array
- DataFrameStack.array: create a DataFrameStack from an array
- as.DataFrameStack.array: create a DataFrameStack from an array
- DataFrameStack.list: create a DataFrameStack from a list
- as.DataFrameStack.list: create a DataFrameStack from an array

See Also

stackDim() to get or set the stacking dimension; noStackDim() to get (not set) the non-stacking dimension; as.array.DataFrameStack() and as.list.DataFrameStack() to convert the stack to an array or a list; dimnames.DataFrameStack() to get the dimension names; [[.DataFrameStack] to

extract rows of a stack; `getStackElement()` and `setStackElement` (same page as `getStackElement`) to extract/modify an element of the stack; `gmApply()` to apply any function to the stack, typically element-wise; and `swarmPlot()` to combine plot elements for each stack element into a single plot.

Examples

```
ll = lapply(1:3, function(i) as.data.frame(matrix(1:10+(i-1)*10, ncol=2)))
dfs = DataFrameStack(ll, stackDimName="rep")
dimnames(dfs)
df = as.data.frame(matrix(1:30, ncol=6))
dfs = DataFrameStack(df, dimnames = list(obs=1:5, vars=c("A","B"), rep=1:3), stackDim = "rep")
dimnames(dfs)
ar = array(1:30, dim = c(5,2,3), dimnames=list(obs=1:5, vars=c("A","B"), rep=1:3))
dfs = DataFrameStack(ar, stackDim="rep")
dimnames(dfs)
```

`dimnames.DataFrameStack`

Return the dimnames of a DataFrameStack

Description

Return the dimnames of a `DataFrameStack()`, i.e. the three dimensions

Usage

```
## S3 method for class 'DataFrameStack'
dimnames(x)

## S4 method for signature 'Spatial'
dimnames(x)
```

Arguments

x	a `DataFrameStack()` object

Value

a list (possibly named) with the element names of each of the three dimensions

Functions

- `dimnames,Spatial-method`: dimnames method for all Spatial*DataFrame objects of package sp which data slot contains a `DataFrameStack()`

DSpars *Create a parameter set specifying a direct sampling algorithm*

Description

Create a parameter set describing a direct sampling algorithm to multipoint simulation. All parameters except `nsim` are optional, as they have default values reasonable according to experience.

Usage

```
DSpars(nsim = 1, scanFraction = 0.25, patternSize = 10, gof = 0.05, ...)
```

Arguments

`nsim`	number of realisations desired (attention: current algorithm is slow, start with small values!)
`scanFraction`	maximum fraction of the training image to be scanned on each iteration
`patternSize`	number of observations used for conditioning the simulation
`gof`	maximum acceptance discrepance between a data event in the training image and the conditioning data event
`...`	further parameters, not used

Value

an S3-list of class "gmDirectSamplingParameters" containing the six elements given as arguments to the function. This is just a compact way to provide further functions such as `predict.gmSpatialModel()` with appropriate triggers for choosing a prediction method or another, in this case for triggering direct sampling.

Examples

```
(dsp = DSpars(nsim=100, scanFraction=75, patternSize=6, gof=0.05))
## then run predict(..., pars=dsp)
```

EmpiricalStructuralFunctionSpecification-class *Empirical structural function specification*

Description

Abstract class, containing any specification of an empirical variogram (or covariance function, or variations)

`fit_lmc.logratioVariogramAnisotropy` *Fit an LMC to an empirical variogram*

Description

Fit a linear model of coregionalisation to an empirical variogram

Usage

```
## S3 method for class 'logratioVariogramAnisotropy'
fit_lmc(v, g, model, ...)

fit_lmc(v, ...)

## S3 method for class 'gstatVariogram'
fit_lmc(v, g, model, ...)

## Default S3 method:
fit_lmc(v, g, model, ...)

## S3 method for class 'logratioVariogram'
fit_lmc(v, g, model, ...)
```

Arguments

`v`	empirical variogram
`g`	spatial data object, containing the original data
`model`	LMC or variogram model to fit
`...`	further parameters

Value

Method fit_lmc.gstatVariogram is a wrapper around `gstat::fit.lmc()`, that calls this function and gives the resulting model its appropriate class (c("variogramModelList", "list")). Method fit_lmc.default returns the fitted lmc (this function currently uses gstat as a calculation machine, but this behavior can change in the future)

Methods (by class)

- `logratioVariogramAnisotropy`: method for logratioVariogram with anisotropry
- `gstatVariogram`: wrapper around gstat::fit.lmc method
- `default`: flexible wrapper method for any class for which methods for `as.gstatVariogram()`, `as.gstat()` and `as.variogramModel()` exist. In the future there may be direct specialised implementations not depending on package gstat.

- `logratioVariogram`: method for logratioVariogram wrapping compositions::fit.lmc. In the future there may be direct specialised implementations, including anisotropy (not yet possible).

Examples

```
data("jura", package = "gstat")
X = jura.pred[,1:2]
Zc = jura.pred[,7:13]
gg = make.gmCompositionalGaussianSpatialModel(Zc, X, V="alr", formula = ~1)
vg = variogram(gg)
md = gstat::vgm(model="Sph", psill=1, nugget=1, range=1.5)
gg = fit_lmc(v=vg, g=gg, model=md)
variogramModelPlot(vg, model=gg)
```

`getMask` *Get the mask info out of a spatial data object*

Description

Retrieve the mask information from an object (if present). See `constructMask()` for examples.

Usage

```
getMask(x)

## Default S3 method:
getMask(x)

## S3 method for class 'SpatialPixelsDataFrame'
getMask(x)

## S3 method for class 'SpatialPixels'
getMask(x)

## S3 method for class 'SpatialPointsDataFrame'
getMask(x)
```

Arguments

x	a masked object

Value

The retrieved mask information from x, an object of class "mask"

Methods (by class)

- `default`: Get the mask info out of a spatial data object
- `SpatialPixelsDataFrame`: Get the mask info out of a SpatialPixelsDataFrame data object
- `SpatialPixels`: Get the mask info out of a SpatialPixels object
- `SpatialPointsDataFrame`: Get the mask info out of a SpatialPointsDataFrame data object

See Also

Other masking functions: `constructMask()`, `print.mask()`, `setMask()`, `unmask()`

---

getStackElement *Set or get the i-th data frame of a data.frame stack*

Description

Set or get one element of the `DataFrameStack()`

Usage

```
getStackElement(x, i, ...)

## Default S3 method:
getStackElement(x, i, ...)

## S3 method for class 'list'
getStackElement(x, i, ...)

## S3 method for class 'DataFrameStack'
getStackElement(x, i, MARGIN = stackDim(x), ...)

## Default S3 method:
setStackElement(x, i, value, ...)

## S3 method for class 'data.frame'
setStackElement(x, i, value, ...)

## S3 method for class 'list'
setStackElement(x, i, value, ...)

## S3 method for class 'DataFrameStack'
setStackElement(x, i, value, MARGIN = stackDim(x), ...)
```

Arguments

x	container data, typically a DataFrameStack(), but it can also be certain sp::Spatial() object derivates of it
i	index (or name) of the element of the stack to extract or replace
...	extra arguments for generic functionality
MARGIN	which dimension is the stacking dimension? you seldom want to touch this!!
value	for the setting operation, the new data.frame to replace the selected one; note that the compatibility of the dimensions of value is only checked for setStackElement.DataFrameStack and its Spatial derivates; for other methods setStackElement can break the consistency of the stack!

Value

For the getters, the result is the data.frame of the stack asked for. For the setters the result is the original DataFrameStack with the corresponding element replaced. Spatial methods return the corresponding spatial object, ie. the spatial information of the stack is transferred to the extracted element.

Methods (by class)

- default: Set or get one element of the DataFrameStack()
- list: Set or get one element of the DataFrameStack()
- DataFrameStack: Set or get one element of the DataFrameStack()
- default: Set or get one element of the DataFrameStack()
- data.frame: Set one element of the DataFrameStack() in data.frame form
- list: Set get one element of a DataFrameStack() in list form
- DataFrameStack: Set one element of the DataFrameStack()

Examples

```
ar = array(1:30, dim = c(5,2,3), dimnames=list(obs=1:5, vars=c("A","B"), rep=1:3))
dfs = DataFrameStack(ar, stackDim="rep")
dfs
stackDim(dfs)
getStackElement(dfs, 1)
```

---

getTellus *Download the Tellus survey data set (NI)*

Description

Download the A-soil geochemistry data of the Tellus survey (Northern Ireland), and if desired produce a training image of the geochemical subcomposition Mg-Al-Ca-Fe-Rest.

Usage

```
getTellus(wd = ".", destfile = "TellusASoil.RData", TI = FALSE, cleanup = TRUE)
```

Arguments

wd	working directory, where intermediate files and output will be produced
destfile	file name where to put the Tellus data (including the extension ".Rdata"!)
TI	either a logical (should the training image be created? defaults to FALSE) or else the file name where to put the created training image (including the extension ".Rdata")
cleanup	should the downloaded excel file be removed? defaults to TRUE

Details

The function is provided due to conflicting licenses. You download the data from the server https://www.bgs.ac.uk/gsni/tellus/index.html at your own accord. Actually a visit to the project data webpage is highly recommended, in particular for learning about the QA/QC process of the data acquisition, other data sources available and how to use this wealth of data.

Value

TRUE if everything went OK. The function is actually called for the side effect of downloading data from the Tellus survey project website, so be careful to call it specifying the right directory at wd.

Geochemical soil A survey

The function will download an excel file to your working directory, do some manipulations, save the resulting data.frame in a "TellusASoil.RData" file (or any other name you provide on the destfile argument) and eventually, remove the excel file (if you left cleanup=TRUE). This data set has 6862 obervations and 58 variables:

Sample Sample ID

EASTING X coordinate of each point

NORTHING Y coordinate of each point

Flag a flag marking some data as a study subset, fixed for reproducibility

Ag Silver concentration in ppm

Cd Cadmium concentration in ppm

In Indium concentration in ppm

Sn Tin concentration in ppm

Sb Antimony concentration in ppm

Te Tellurium concentration in ppm

I Indim concentration in ppm

Cs Caesium concentration in ppm

Ba Barium concentration in ppm

La Lanthanum concentration in ppm

Ce Cerium concentration in ppm

Na2O Sodium oxide in percent

MgO Magnesium oxide in percent

Al2O3 Aluminium oxide in percent

SiO2 Silicium oxide in percent

P2O5 Phosphorous(V) oxide in percent

SO3 Sulphur(III) oxide in percent

K2O Potasium oxide in percent

CaO Calcium oxide in percent

TiO2 Titanium oxide in percent

MnO Manganese oxide in percent

Fe2O3 Total Iron(III) oxide in percent

Cl Chlorine concentration in ppm

Sc Scandium concentration in ppm

V Vanadium concentration in ppm

Cr Chromium concentration in ppm

Co Cobalt concentration in ppm

Ni Nickel concentration in ppm

Cu Copper concentration in ppm

Zn Zinc concentration in ppm

Ga Callium concentration in ppm

Ge Germanium concentration in ppm

As Arsenic concentration in ppm

Se Selenium concentration in ppm

Br Bromine concentration in ppm

Rb Rubidium concentration in ppm

Sr Strontium concentration in ppm

Y Yttrium concentration in ppm

Zr Zirconium concentration in ppm

Nb Niobium concentration in ppm

Mo Molybdenum concentration in ppm

Nd Neodymium concentration in ppm

Sm Samarium concentration in ppm

Yb Ytterbium concentration in ppm

Hf Hafnium concentration in ppm

Ta Tantalum concentration in ppm

W Tungsten concentration in ppm

Tl Thallium concentration in ppm

Pb Lead concentration in ppm

Bi Bismuth concentration in ppm

Th Thorium concentration in ppm

U Uranium concentration in ppm

pH soil acidity

LOI loss on ignition in percent

Training image

Additionally, if you give TI!=FALSE, the function will produce an additional file "Tellus_TI.RData" (if TI=TRUE, or any other filename that you specify on the argument TI) with a data.frame with 13287 rows and 8 columns:

EASTING X coordinate of each cell point

NORTHING Y coordinate of each cell point

MgO Magnesia proportion

Al2O3 Alumina proportion

CaO Calcium oxide proportion

Fe2O3 Iron oxide proportion

Rest Residual complementing the sum to 1

Mask Indicator of grid point outside the boundary of the country. NOTE: to use it with setMask() you will need to invert it using !as.logical(TellusTI$Mask)

This is a migrated version of the data set to a regular grid, ideal for illustrating and testing multiple point geostatistical algorithms.

References

https://www.bgs.ac.uk/gsni/tellus/index.html

Examples

```
## Not run:
 getwd()
 getTellus(TI=TRUE, cleanup=TRUE)
 dir(pattern="Tellus")

## End(Not run)
```

gmApply *Apply Functions Over Array or DataFrameStack Margins*

Description

Returns a vector or array or list of values obtained by applying a function to the margins of an array or matrix. Method `gmApply.default()` is just a wrapper on `base::apply()`. Method `gmApply()` reimplements the functionality with future access to parallel computing and appropriate default values for the MARGIN. ALWAYS use named arguments here!

Usage

```
gmApply(X, ...)

## Default S3 method:
gmApply(X, MARGIN, FUN, ...)

## S3 method for class 'DataFrameStack'
gmApply(X, MARGIN = stackDim(X), FUN, ..., .parallel = FALSE)
```

Arguments

`X`	a `DataFrameStack()` object (see `base::apply()` for other options)
`...`	further arguments to `FUN`
`MARGIN`	a name or an index of the dimension along which should the calculations be done; defaults to the stacking dimension of the `DataFrameStack()`, i.e. to the output of `stackDim(X)`
`FUN`	function to apply; the default behaviour being that this function is applied to each element of the stack `X`
`.parallel`	currently ignored

Value

In principle, if `MARGIN==stackDim(X)` (the default), the oputput is a list with the result of using `FUN` on each element of the stack. If `FUN` returns a matrix or a data.frame assimilable to one element of the stack, a transformation of this output to a DataFrameStack is attempted.

For `X` non-DataFrameStack or `MARGIN!=stackDim(X)` see `base::apply()`.

Methods (by class)

- `default`: wrapper around `base::apply()`
- `DataFrameStack`: Apply Functions Over DataFrameStack Margins

Examples

```
dm = list(point=1:100, var=LETTERS[1:2], rep=paste("r",1:5, sep=""))
ar = array(rnorm(1000), dim=c(100,2,5), dimnames = dm)
dfs = DataFrameStack(ar, stackDim="rep")
gmApply(dfs, FUN=colMeans)
rs = gmApply(dfs, FUN=function(x) x+1)
class(rs)
getStackElement(rs,1)
getStackElement(dfs,1)
```

`gmGaussianMethodParameters-class`
parameters for Spatial Gaussian methods of any kind

Description

abstract class, containing any parameter specification for a spatial algorithm for interpolation, simulation or validation making use of Gaussian assumptions

---

`gmGaussianSimulationAlgorithm-class`
parameters for Gaussian Simulation methods

Description

abstract class, containing any parameter specification of a spatial simulation algorithm exploiting a Gaussian two-point model structure

---

`gmMPSParameters-class` *parameters for Multiple-Point Statistics methods*

Description

abstract class, containing any parameter specification of a spatial multipoint algorithm

---

`gmNeighbourhoodSpecification-class`
Neighbourhood description

Description

abstract class, containing any specification of a spatial neighbourhood

gmSimulationAlgorithm-class

Parameter specification for a spatial simulation algorithm

Description

abstract class, containing any parameter specification for a spatial simulation algorithm

gmSpatialDataContainer-class

General description of a spatial data container

Description

abstract class, containing any specification of a spatial data container

Details

setClassUnion(name="gmTrainingImage", members=c("SpatialGridDataFrame", "SpatialPixelsDataFrame")
)

gmSpatialMethodParameters-class

Parameter specification for any spatial method

Description

abstract class, containing any parameter specification for any spatial method. Members of this class are gmNeighbourhoodSpecification gmMPSParameters and gmValidationStrategy.

`gmSpatialModel-class` *Conditional spatial model data container*

Description

This class is devised to contain a conditional spatial model, with: some conditioning data (a `sp::SpatialPointsDataFrame()`), an unconditional geospatial model (a structure with e.g. a training image; or the information defining a Gaussian random field); and eventually some extra method parameters. The class extends `sp::SpatialPointsDataFrame()` and has therefore its slots, plus `model` (for the unconditional model) and `parameters` (for the extra method information)

Usage

```
## S4 method for signature 'gmSpatialModel'
variogram(object, methodPars = NULL, ...)

## S4 method for signature 'gmSpatialModel'
logratioVariogram(data, ..., azimuth = 0, azimuth.tol = 180/length(azimuth))

## S4 method for signature 'gmSpatialModel'
as.gstat(object, ...)
```

Arguments

`object`	a gmSpatialModel object containing spatial data.
`methodPars`	(currently ignored)
`...`	further parameters to `gstat::variogram()`
`data`	the data container (see gmSpatialModel for details)
`azimuth`	which direction, or directions, are desired (in case of directional variogram)
`azimuth.tol`	which tolerance sould be used for directional variograms?

Value

You will seldom create the spatial model directly. Use instead the creators make.gm* linked below

Methods (by generic)

- `variogram`: Compute a variogram, see `variogram_gmSpatialModel()` for details
- `logratioVariogram`: S4 wrapper method around `logratioVariogram()` for `gmSpatialModel` objects
- `as.gstat`: convert from gmSpatialModel to gstat; see `as.gstat()` for details

Slots

`data` a data.frame (or class extending it) containing the conditional data

`coords` a matrix or dataframe of 2-3 columns containing the sampling locations of the conditional data

`coords.nrs` see `sp::SpatialPointsDataFrame()`

`bbox` see `sp::SpatialPointsDataFrame()`

`proj4string` see `sp::SpatialPointsDataFrame()`

`model` gmUnconditionalSpatialModel. Some unconditional geospatial model. It can be NULL.

`parameters` gmSpatialMethodParameters. Some method parameters. It can be NULL

See Also

Other gmSpatialModel: `as.gmSpatialModel()`, `make.gmCompositionalGaussianSpatialModel()`, `make.gmCompositionalMPSSpatialModel()`, `make.gmMultivariateGaussianSpatialModel()`, `predict.gmSpatialModel()`

Examples

```
data("jura", package="gstat")
library(sp)
X = jura.pred[,1:2]
Zc = jura.pred[,7:13]
spdf = sp::SpatialPointsDataFrame(coords=X, data=Zc)
new("gmSpatialModel", spdf)
make.gmCompositionalGaussianSpatialModel(data=Zc, coords=X, V="alr")
```

---

gmTrainingImage-class *MPS training image class*

Description

abstract class, containing any specification of a multiple-point training image

---

gmUnconditionalSpatialModel-class *General description of a spatial model*

Description

abstract class, containing any specification of an unconditional spatial model

gmValidationStrategy-class *Validation strategy description*

Description

abstract class, containing any specification of a validation strategy for spatial models

GridOrNothing-class *Superclass for grid or nothing*

Description

Superclass for slots containing a grid topology or being empty

gsi.calcCgram *Compute covariance matrix oout of locations*

Description

internal function to compute the variogram model for all combinations of two

Usage

```
gsi.calcCgram(X, Y, vgram, ijEqual = FALSE)
```

Arguments

X	matrix, coordinates of the first set of locations
Y	matrix, coordinates of the first set of locations (often Y=X, and ijEqual=TR)
vgram	covariogram model, of format "gmCgram"
ijEqual	logical, are X==Y? if you put TRUE, the function will only compute half of the points, so be sure that this is right!

Value

a matrix of elements of the covariance, in NxM blocks of DxD elements (being N and M resp. the nr. of rows of X and Y, and D the nr of variables)

`gsi.Cokriging` *Cokriging of all sorts, internal function*

Description

internal function to compute cokriging (simple, ordinary, universal, with trend)

Usage

```
gsi.Cokriging(
  Xout,
  Xin,
  Zin,
  vgram,
  Fin = rep(1, nrow(Xin)),
  Fout = rep(1, nrow(Xout)),
  krigVar = FALSE,
  tol = 1e-15
)
```

Arguments

`Xout`	(M,g)-matrix with coordinates of desired interpolation locations
`Xin`	(N,g)-matrix with coordinates of conditioning locations
`Zin`	(N,D)-matrix with condtioning observations
`vgram`	D-dimensional variogram model of class "gmCgram"
`Fin`	either NULL or a (N,p)-matrix with the base functions of the trend evaluated at the conditioning locations (defaults to a vector of N 1's)
`Fout`	either NULL or a (m,p)-matrix with the base functions of the trend evaluated at the conditioning locations (defaults to a vector of M 1's)
`krigVar`	logical or NA, should kriging variances be returned? FALSE=no, TRUE=give point-wise cokriging covariance; NA=return the whole covariance matrix of all points
`tol`	tolerance for the inversion of the cokriging matrix (in case of near-singularity)

Value

A (M,D)-matrix of predictions, eventually with an attribute "krigVar" containing the output defined by argument `krigVar`

gsi.CondTurningBands *Internal function, conditional turning bands realisations*

Description

Internal function to compute conditional turning bands simulations

Usage

```
gsi.CondTurningBands(
  Xout,
  Xin,
  Zin,
  vgram,
  nbands = 400,
  tol = 1e-15,
  nsim = NULL
)
```

Arguments

Xout	matrix of coordinates of locations where realisations are desired
Xin	matrix of coordinates of locations of conditioning points
Zin	matrix of variables at the conditioning locations
vgram	covariogram model, of format "gmCgram"
nbands	number of bands to use
tol	tolerance for the inversion of the cokriging matrix (in case of near-singularity)
nsim	number of realisations to return

Value

an array with (npoint, nvar, nsim)-elements, being npoint=nrow(X) and nvar = nr of variables in vgram

gsi.DS *Workhorse function for direct sampling*

Description

This function implements in R the direct sampling algorithm

Usage

```
gsi.DS(
  n,
  f,
  t,
  n_realiz,
  dim_TI,
  dim_SimGrid,
  TI_input,
  SimGrid_input,
  ivars_TI = 3:ncol(TI_input),
  SimGrid_mask = ncol(SimGrid_input),
  invertMask = TRUE
)
```

Arguments

`n`	size of the conditioning data event (integer)
`f`	fraction of the training image to scan (numeric between 0 and 1)
`t`	maximal acceptable discrepance between conditioning data event and TI event (numeric between 0 and 1)
`n_realiz`	number of simulations desired
`dim_TI`	dimensions of the grid of the training image (ie. either (n_x, n_y) for dimension $k = 2$ or (n_x, n_y, n_z) for dimension $k = 3$)
`dim_SimGrid`	dimensions of the simulation grid (ie. either (m_x, m_y) or (m_x, m_y, m_z))
`TI_input`	training image, as a matrix of $(n_x \cdot n_y \cdot n_z, k + D)$ elements; WITH NAMED COLUMNS and including spatial coordinates
`SimGrid_input`	simulation grid with conditioning data, as a matrix of $(m_x \cdot m_y \cdot m_z, k + D)$ elements; with same columns as `TI_input`
`ivars_TI`	which colnames of `TI_input` and `SimGrid_input` identify variables to consider in the data event
`SimGrid_mask`	either a logical vector of length $m_x \cdot m_y \cdot m_z$, or else a column name of `SimGrid_input` giving a logical column
`invertMask`	logical, does `SimGrid_mask` identify with TRUE the data OUTSIDE the simulation area?

Value

A `sp::SpatialPixelsDataFrame()` or `sp::SpatialGridDataFrame()`, depending on whether the whole grid is simulated. The '@data' slot of these objects contains a `DataFrameStack()` with the stacking dimension running through the realisations. It is safer to use this functionality through the interface `make.gmCompositionalMPSSpatialModel()`, then request a direct simulation with `DSpars()` and finally run it with `predict.gmSpatialModel()`.

Author(s)

Hassan Talebi (copyright holder), Raimon Tolosana-Delgado

Examples

```
## training image:
x = 1:10
y = 1:7
xy_TI = expand.grid(x=x, y=y)
TI_input = cbind(xy_TI, t(apply(xy_TI, 1, function(x) c(sum(x), abs(x[2]-x[1]))+rnorm(2, sd=0.01))))
colnames(TI_input) = c("x", "y", "V1", "V2")
o1 = image_cokriged(TI_input, ivar="V1")
o2 = image_cokriged(TI_input, ivar="V2")
## simulation grid:
SimGrid = TI_input
SimGrid$mask = with(SimGrid, x==1 | x==10 | y==1 | y==7)
tk = SimGrid$mask
tk[sample(70, 50)] = TRUE
SimGrid[tk,3:4]=NA
image_cokriged(SimGrid, ivar="V1", breaks=o1$breaks, col=o1$col)
image_cokriged(SimGrid, ivar="V2", breaks=o2$breaks, col=o2$col)
image_cokriged(SimGrid, ivar="mask", breaks=c(-0.0001, 0.5, 1.001))
res = gsi.DS(n=5, f=0.75, t=0.05, n_realiz=2, dim_TI=c(10,7),  dim_SimGrid=c(10,7),
       TI_input=as.matrix(TI_input), SimGrid_input=as.matrix(SimGrid),
       ivars_TI = c("V1", "V2"), SimGrid_mask="mask", invertMask=TRUE)
image_cokriged(cbind(xy_TI, getStackElement(res,1)), ivar="V1", breaks=o1$breaks, col=o1$col)
image_cokriged(cbind(xy_TI, getStackElement(res,2)), ivar="V1", breaks=o1$breaks, col=o1$col)
image_cokriged(cbind(xy_TI, getStackElement(res,1)), ivar="V2", breaks=o2$breaks, col=o2$col)
image_cokriged(cbind(xy_TI, getStackElement(res,2)), ivar="V2", breaks=o2$breaks, col=o2$col)
```

---

gsi.EVario2D *Empirical variogram or covariance function in 2D*

Description

compute the empirical variogram or covariance function in a 2D case study

Usage

```
gsi.EVario2D(
  X,
  Z,
  Ff = rep(1, nrow(X)),
  maxdist = max(dist(X[sample(nrow(X), min(nrow(X), 1000)), ]))/2,
  lagNr = 15,
  lags = seq(from = 0, to = maxdist, length.out = lagNr + 1),
  azimuthNr = 4,
  azimuths = seq(from = 0, to = 180, length.out = azimuthNr + 1)[1:azimuthNr],
  maxbreadth = Inf,
  minpairs = 10,
  cov = FALSE
)
```

Arguments

X	matrix of Nx2 columns with the geographic coordinates
Z	matrix or data.frame of data with dimension (N,Dv)
Ff	for variogram, matrix of basis functions with nrow(Ff)=N (can be a N-vector of 1s); for covariance function, a (N,Dv)-matrix or a Dv-vector giving the mean values
maxdist	maximum lag distance to consider
lagNr	number of lags to consider
lags	if maxdist and lagNr are not specified, either: (a) a matrix of 2 columns giving minimal and maximal lag distance defining the lag classes to consider, or (b) a vector of lag breaks
azimuthNr	number of azimuths to consider
azimuths	if azimuthNr is not specified, either: (a) a matrix of 2 columns giving minimal and maximal azimuth defining the azimuth classes to consider, or (b) a vector of azimuth breaks
maxbreadth	maximal breadth (in lag units) orthogonal to the lag direction
minpairs	minimal number of pairs falling in each class to consider the calculation sufficient; defaults to 10
cov	boolean, is covariance (TRUE) or variogram (FALSE) desired? defaults to variogram

Value

An empirical variogram for the provided data. NOTE: avoid using directly gsi.* functions! They represent either internal functions, or preliminary, not fully-tested functions. Use `variogram` instead.

See Also

Other gmEVario functions: `as.gmEVario()`, `ndirections()`, `plot.gmEVario()`, `variogramModelPlot()`

Examples

```
library(gstat)
data("jura", package = "gstat")
X = as.matrix(jura.pred[,1:2])
Z = as.matrix(jura.pred[,c("Zn","Cd","Pb")])
vge = gsi.EVario2D(X,Z)
dim(vge)
dimnames(vge)
class(vge["gamma",1])
dim(vge["gamma",1][[1]])
vge["npairs",1]
vge["lags",1]
```

gsi.gstatCokriging2compo *Reorganisation of cokriged compositions*

Description

Produce compositional predictions out of a gstat::gstat() prediction

Usage

```
gsi.gstatCokriging2compo(COKresult, ...)

## Default S3 method:
gsi.gstatCokriging2compo(COKresult, ...)

## S3 method for class 'data.frame'
gsi.gstatCokriging2compo(
  COKresult,
  V = NULL,
  orignames = NULL,
  tol = 1e-12,
  nscore = FALSE,
  gg = NULL,
  ...
)

## Default S3 method:
gsi.gstatCokriging2rmult(COKresult, ...)

## S3 method for class 'data.frame'
gsi.gstatCokriging2rmult(COKresult, nscore = FALSE, gg = NULL, ...)
```

Arguments

COKresult	output of a gstat::predict.gstat() cokriging, typically of class "data.frame", sp::SpatialPointsDataFrame(), sp::SpatialGridDataFrame() or sp::SpatialPixelsDataFrame
...	further arguments needed for nscore (**deprecated**)
V	string or matrix describing which logratio was applied ("ilr", "alr", or a matrix computing the ilr corrdinates; clr is not allowed!)
orignames	names of the original components (optional, but recommended)
tol	for generalized inversion of the matrix (**rarely touched!**)
nscore	boolean, were the data normal score-transformed? (**deprecated**)
gg	in the case that normal score transformation was applied, provide the gstat object! (**deprecated**)

Value

an (N,D)-object of class `c("spatialGridAcomp","acomp")` with the predictions, together with an extra attribute "krigVar" containing the cokriging covariance matrices in an (N, D, D)-array; here N=number of interpolated locations, D=number of original components of the composition

Methods (by class)

- `default`: Reorganisation of cokriged compositions
- `data.frame`: Reorganisation of cokriged compositions
- `default`: Reorganisation of cokriged multivariate data
- `data.frame`: Reorganisation of cokriged multivariate data

See Also

`image_cokriged.spatialGridRmult()` for an example

---

gsi.orig *extract information about the original data, if available*

Description

originally implemented in package:compositions

Usage

```
gsi.orig(x, y = NULL)
```

Arguments

x	an rmult object
y	possibly another rmult object

Value

The original untransformed data (with a compositional class), resp the TRANSPOSED INVERSE matrix, i.e. the one to recover the original components.

gsi.produceV *Create a matrix of logcontrasts and name prefix*

Description

Given a representation specification for compositions, this function creates the matrix of logcontrasts and provides a suitable prefix name for naming variables.

Usage

```
gsi.produceV(
  V = NULL,
  D = nrow(V),
  orignames = rownames(V),
  giveInv = FALSE,
  prefix = NULL
)
```

Arguments

V	either a matrix of logcontrasts or, most commonly, one of "clr", "ilr" or "alr"
D	the number of components of the composition represented
orignames	the names of the components
giveInv	logical, is the inverse logcontrast matrix desired?
prefix	the desired prefix name, if this is wished to be forced.

Value

A list with at least two elements

1. V containing the final matrix of logcontrasts
2. prefix containing the final prefix for names of transformed variables
3. W eventually, the (transposed, generalised) inverse of V, if giveInv=TRUE

Examples

```
gsi.produceV("alr", D=3)
gsi.produceV("ilr", D=3, orignames = c("Ca", "K", "Na"))
gsi.produceV("alr", D=3, orignames = c("Ca", "K", "Na"), giveInv = TRUE)
```

gsi.TurningBands *Internal function, unconditional turning bands realisations*

Description

Internal function to compute unconditional turning bands simulations

Usage

```
gsi.TurningBands(X, vgram, nBands, nsim = NULL)
```

Arguments

X	matrix of coordinates of locations where realisations are desired
vgram	covariogram model, of format "gmCgram"
nBands	number of bands to use
nsim	number of realisations to return

Value

an array with (npoint, nvar, nsim)-elements, being npoint=nrow(X) and nvar = nr of variables in vgram

has.missings.data.frame *Check presence of missings check presence of missings in a data.frame*

Description

Check presence of missings check presence of missings in a data.frame

Usage

```
## S3 method for class 'data.frame'
has.missings(x, mc = NULL, ...)
```

Arguments

x	data, of class data.frame
mc	not used
...	not used

Value

A single true/false response about the presence of any missing value on the whole data set

Examples

```
library(compositions)
data(Windarling)
has.missings(Windarling)
head(Windarling)
Windarling[1,1] = NA
head(Windarling)
has.missings(Windarling)
```

`image.logratioVariogramAnisotropy`
Plot variogram maps for anisotropic logratio variograms

Description

Image method to obtain variogram maps for anisotropic logratio variograms

Usage

```
## S3 method for class 'logratioVariogramAnisotropy'
image(
  x,
  jointColor = FALSE,
  breaks = NULL,
  probs = seq(0, 1, 0.1),
  col = spectralcolors,
  ...
)
```

Arguments

`x`	object of class c("logratioVariogramAnisotropy", "logratioVariogram")
`jointColor`	logical, should all variogram maps share the same color scale?
`breaks`	breaks to use in the color scale
`probs`	alternatively to explicit `breaks`, these probabilities allow to ask for some equally probable breaks
`col`	either a color palette, or else a vector of colors to use, of length `length(breaks)-1`
`...`	additional arguments for generic functionality (currently ignored)

Value

This function is called for its effect of producing a figure. Additionally, the graphical parameters active *prior to calling this function* are returned invisibly.

Examples

```
data("jura", package="gstat")
X = jura.pred[,1:2]
Zc = compositions::acomp(jura.pred[,7:9])
vg = logratioVariogram(data=Zc, loc=X, azimuth=c(0:18)*10,
                           azimuth.tol=22.5)
image(vg)
image(vg, jointColor=TRUE)
```

`image.mask` *Image method for mask objects*

Description

Plot a mask of a 2D grid; see `constructMask()` for an example

Usage

```
## S3 method for class 'mask'
image(x, col = c(NA, 2), ...)
```

Arguments

`x`	a mask
`col`	a two-color vector for the color (oustide, inside) the mask
`...`	ignored

Value

nothing

`image_cokriged` *Plot an image of gridded data*

Description

Plot an image of one variable (possibly, one realisation) of output of cokriging or cosimulation functions.

Usage

```
image_cokriged(x, ...)

## Default S3 method:
image_cokriged(
  x,
  ivar = 3,
  breaks = quantile(as.data.frame(x)[, ivar], probs = c(0:10)/10, na.rm = TRUE),
  col = spectralcolors(length(breaks) - 1),
  legendPropSpace = 0.2,
  legendPos = "top",
  main = ifelse(is.character(ivar), ivar, colnames(x)[ivar]),
  ...
)

## S3 method for class 'spatialGridRmult'
image_cokriged(
  x,
  ivar = 1,
  isim = NULL,
  breaks = 10,
  mask = attr(x, "mask"),
  col = spectralcolors(length(breaks) - 1),
  legendPropSpace = 0.2,
  legendPos = "top",
 main = ifelse(is.character(ivar), ivar, dimnames(x)[[length(dimnames(x))]][ivar]),
  ...
)

## S3 method for class 'spatialGridAcomp'
image_cokriged(
  x,
  ivar = 1,
  isim = NULL,
  breaks = 10,
  mask = attr(x, "mask"),
  col = spectralcolors(length(breaks) - 1),
  legendPropSpace = 0.2,
  legendPos = "top",
 main = ifelse(is.character(ivar), ivar, dimnames(x)[[length(dimnames(x))]][ivar]),
  ...
)
```

Arguments

x	object with the interpolated / simulated data; currently there are methods for "spatialGridAcomp" and "spatialGridRmult", but the default method is able to deal with "SpatialPointsDataFrame", "SpatialPixelsDataFrame" and "Spatial-

	GridDataFrame" objects, and with the "data.frame" output of `gstat::predict.gstat()` and `predict.gmSpatialModel()`
`...`	generic functionality, currently ignored
`ivar`	which variable do you want to plot?
`breaks`	either the approximate number of breaks, or the vector of exact breaks to use for the plotting regions of the chosen variable
`col`	vector of colors to use for the image
`legendPropSpace`	which proportion of surface of the device should be used for the legend? trial and error might be necessary to adjust this to your needs
`legendPos`	where do you want your legend? one of c("top","left","right","bottom")
`main`	main title for the plot
`isim`	in case of simulated output, which simulation?
`mask`	optional mask object if `x` is of class "spatialGridAcomp" or "spatialGridRmult", and they have been masked (see `setMask()`)

Value

Invisibly, a list with elements `breaks` and `col` containing the breaks and hexadecimal colors finally used for the z-values of the image. Particularly useful for plotting other plotting elements on the same color scale.

Methods (by class)

- `default`: Plot an image of gridded data
- `spatialGridRmult`: method for spatialGridRmult objects
- `spatialGridAcomp`: method for spatialGridAcomp objects

Examples

```
## Not run:
getTellus(cleanup=TRUE, TI=TRUE)
load("Tellus_TI.RData")
head(Tellus_TI)
coords = as.matrix(Tellus_TI[,1:2])
compo = compositions::acomp(Tellus_TI[,3:7])
dt = spatialGridAcomp(coords=coords, compo=compo)
image_cokriged(dt, ivar="MgO") # equi-spaced
image_cokriged(dt, ivar="MgO", breaks = NULL) # equi-probable

## End(Not run)
```

is.anisotropySpecification *Check for any anisotropy class*

Description

Check that an object contains a valid specification of anisotropy, in any form

Usage

```
is.anisotropySpecification(x)
```

Arguments

x	object to check

Value

a logical, TRUE if the object is an anisotropy specification; FALSE otherwise

Examples

```
a =  anis2D.par2A(0.5, 30)
a
is.anisotropySpecification(a)
```

is.isotropic *Check for anisotropy of a theoretical variogram*

Description

Checks for anisotropy of a theoretical variogram or covariance function model

Usage

```
is.isotropic(x, tol = 1e-10, ...)
```

Arguments

x	variogram or covariance model object
tol	tolerance for
...	extra arguments for generic functionality

Value

Generic function. Returns of boolean answering the question of the name, or NA if object x does not contain a known theoretical variogram

KrigingNeighbourhood *Create a parameter set of local for neighbourhood specification.*

Description

Create a parameter set describing a kriging neighbourhood (local or global) for cokriging and cokriging based simulation. This heavily relies on the definitions of `gstat::gstat()`. All parameters are optional, as their default amounts to a global neihghbourhood.

Usage

```
KrigingNeighbourhood(
  nmax = Inf,
  nmin = 0,
  omax = 0,
  maxdist = Inf,
  force = FALSE,
  anisotropy = NULL,
  ...
)
```

Arguments

`nmax`	maximum number of data points per cokriging system
`nmin`	minimum number of data points per cokriging system
`omax`	maximum number of data points per cokriging system per quadrant/octant
`maxdist`	maximum radius of the search neighborhood
`force`	logical; if less than `nmin` points are found inside `maxdist` radius, keep searching.
`anisotropy`	currently ignored; in the future, argument to specify anisotropic search areas.
`...`	further arguments, currently ignored

Value

an S3-list of class "gmKrigingNeighbourhood" containing the six elements given as arguments to the function. This is just a compact way to provide further functions such as `predict.gmSpatialModel()` with appropriate triggers for choosing a prediction method or another, in this case for triggering cokriging (if alone) or eventually sequential simulation (see `SequentialSimulation()`).

Examples

```
data("jura", package="gstat")
X = jura.pred[,1:2]
summary(X)
Zc = jura.pred[,7:10]
ng_global = KrigingNeighbourhood()
ng_local = KrigingNeighbourhood(maxdist=1, nmin=4,
```

```
                                    omax=5, force=TRUE)
ng_local
ng_global
make.gmCompositionalGaussianSpatialModel(data = Zc, coords = X,
                                         V = "alr", ng = ng_local)
```

`LeaveOneOut` *Specify the leave-one-out strategy for validation of a spatial model*

Description

Function to specify the leave-one-out strategy for validation of a spatial model

Usage

```
LeaveOneOut()
```

Value

an object of class c("LeaveOneOut", "NfoldCrossValidation") to be used in a call to `validate()`

See Also

Other validation functions: `NfoldCrossValidation`, `validate()`

Examples

```
LeaveOneOut()
```

`length.gmCgram` *Length, and number of columns or rows*

Description

Provide number of structures, and nr of variables of an LMC of class gmCgram

Usage

```
## S3 method for class 'gmCgram'
length(x)
```

Arguments

`x`	gmCgram object

Value

`length` returns the number of structures (nugget not counted), while `ncol` and `nrow` return these values for the nugget (assuming that they will be also valid for the sill).

See Also

Other gmCgram functions: `[.gmCgram()`, `[[.gmCgram()`, `as.function.gmCgram()`, `as.gmCgram.variogramModelList()`, `ndirections()`, `plot.gmCgram()`, `variogramModelPlot()`

Examples

```
utils::data("variogramModels")
v1 = setCgram(type=vg.Gau, sill=diag(3)+0.5, anisRanges = 2*diag(c(3,0.5)))
v2 = setCgram(type=vg.Exp, sill=0.3*diag(3), anisRanges = 0.5*diag(2))
vm = v1+v2
length(vm)
ncol(vm)
nrow(vm)
```

LMCAnisCompo *Create a anisotropic model for regionalized compositions*

Description

Creates a (potentially anisotropic) variogram model for variation-variograms

Usage

```
LMCAnisCompo(
  Z,
  models = c("nugget", "sph", "sph"),
  azimuths = rep(0, length(models)),
  ranges = matrix(1, nrow = length(models), ncol = G),
  sillarray = NULL,
  V = NULL,
  tol = 1e-12,
  G = 2
)
```

Arguments

`Z`	compositional data set, used to derive the compositional dimension and colnames
`models`	string (or vector of strings) specifying which reference model(s) to use
`azimuths`	typically a vector providing, for each model, the direction of maximal continuity
`ranges`	typically a `G`-column matrix providing the minimal and maximal ranges, with one row per model (with `G` specified below)

sillarray	array of sills for each model. It can be null (to be estimated in the future). If specified, provide an appropriate (x,x,K)-array, where K is the number of models given, and x is explained below.
V	depending on what do you give as sillarray, you may need to provide the matrix of logcontrasts, or a string indicating whether the sills are represented in "alr" or "ilr"
tol	tolerance for determination of positive definiteness
G	one of c(1, 2, 3) identifying if we are in 1D, 2D or 3D cases

Value

an object of class "LMCAnisCompo" with all provided information appropriately structured, ready to be used for fitting, plotting or prediction.

Examples

```
data("jura", package="gstat")
Zc = compositions::acomp(jura.pred[,7:9])
LMCAnisCompo(Zc, models=c("nugget", "sph"), azimuths=c(0,45))
```

---

logratioVariogram *Empirical logratio variogram calculation*

Description

Calculation of an empirical logratio variogram (aka variation-variogram)

Usage

```
logratioVariogram(data, ...)
```

Arguments

data	a data container (of class "gmSpatialModel") or a composition (of class "acomp")
...	extra arguments for generic functionality

Value

The output of this function depends on the number of azimuth values provided: if it is one single value (or if you explicitly call logratioVariogram.default) the result is a list of class "logratioVariogram" with the following elements

- vg: A nbins x D x D array containing the logratio variograms
- h: A nbins x D x D array containing the mean distance the value is computed on.
- n: A nbins x D x D array containing the number of nonmissing pairs used for the corresponding value. If azimuth is a vector of directions, then the result is a matrix-like list of "logratioVariogram" objects. Each element of the mother list (or column of the matrix) is the variogram of one direction. The output has a compound class c("logratioVariogramAnisotropy", "logratioVariogram").

Examples

```
data("jura", package="gstat")
X = jura.pred[,1:2]
Zc = compositions::acomp(jura.pred[,7:13])
vg = logratioVariogram(data=Zc, loc=X)
class(vg)
summary(vg)
vg = logratioVariogram(data=Zc, loc=X, azimuth=c(0,90))
class(vg)
summary(vg)
```

---

logratioVariogram,acomp-method

Logratio variogram of a compositional data

Description

gmGeostats reimplementation of the compositions::logratioVariogram function

Usage

```
## S4 method for signature 'acomp'
logratioVariogram(
  data = comp,
  loc,
  ...,
  azimuth = 0,
  azimuth.tol = 180/length(azimuth),
  comp = NULL
)
```

Arguments

data	a data container (of class "gmSpatialModel") or a composition (of class "acomp")
loc	if data is a composition (or if comp is provided), spatial coordinates of the sampling locations
...	extra arguments for generic functionality
azimuth	which direction, or directions, are desired (in case of directional variogram)
azimuth.tol	which tolerance sould be used for directional variograms?
comp	an alias for data, provided for back-compatibility with compositions::logratioVariogram()

Value

a "logratioVariogram" object, or a "logratioVariogramAnisotropy" object if you provide more than one azimuth. See logratioVariogram() for details and examples.

`logratioVariogram_gmSpatialModel` *Variogram method for gmSpatialModel objects*

Description

Compute the empirical variogram of the conditioning data contained in a gmSpatialModel object

Usage

```
logratioVariogram_gmSpatialModel(
  data,
  ...,
  azimuth = 0,
  azimuth.tol = 180/length(azimuth)
)

variogram_gmSpatialModel(object, methodPars = NULL, ...)
```

Arguments

`data`	the data container (see gmSpatialModel for details)
`...`	further parameters to `gstat::variogram()`
`azimuth`	which direction, or directions, are desired (in case of directional variogram)
`azimuth.tol`	which tolerance sould be used for directional variograms?
`object`	a gmSpatialModel object containing spatial data.
`methodPars`	(currently ignored)

Value

Currently the function is just a convenience wrapper on the variogram calculation functionalities of package "gstat", and returns objects of class "`gstatVariogram`". Check the help of `gstat::variogram` for further information. In the near future, methods will be created, which will depend on the properties of the two arguments provided, `object` and `methodPars`.

Functions

- `logratioVariogram_gmSpatialModel`: logratio variogram method (see `logratioVariogram()` for details)

`Maf` *Generalised diagonalisations Calculate several generalized diagonalisations out of a data set and its empirical variogram*

Description

Generalised diagonalisations Calculate several generalized diagonalisations out of a data set and its empirical variogram

Usage

```
Maf(x, ...)

## S3 method for class 'data.frame'
Maf(x, vg, i = 2, ...)

## S3 method for class 'rmult'
Maf(x, vg, i = 2, ...)

## S3 method for class 'aplus'
Maf(x, vg, i = 2, ...)

## S3 method for class 'rplus'
Maf(x, vg, i = 2, ...)

## S3 method for class 'ccomp'
Maf(x, vg, i = 2, ...)

## S3 method for class 'rcomp'
Maf(x, vg, i = 2, ...)

## S3 method for class 'acomp'
Maf(x, vg, i = 2, ...)

UWEDGE(x, ...)

## Default S3 method:
UWEDGE(x, ...)

## S3 method for class 'acomp'
UWEDGE(x, vg, i = NULL, ...)

## S3 method for class 'rcomp'
UWEDGE(x, vg, i = NULL, ...)

RJD(x, ...)
```

```
## Default S3 method:
RJD(x, ...)

## S3 method for class 'acomp'
RJD(x, vg, i = NULL, ...)

## S3 method for class 'rcomp'
RJD(x, vg, i = NULL, ...)
```

Arguments

x	a data set, typically of class "data.frame" or of a compositional class
...	generic functionality arguments
vg	empirical variogram, of a kind fitting to the data provided
i	a slicer for the variogram, typically this will be one or more indices of the lag distance to take. %For other options see codegetEmpVariogramSlice.

Value

An object extending `c("princomp.CLASSOF(x)","princomp")` with classes "genDiag", and an extra class argument depending on the diagonalisation method chosen.

Function `Maf` results carry the extra class "maf", and they correspond to minimum/maximum autocorrelation factors (MAF) as proposed by Switzer and Green (1984). In this case, the slicer is typically just the index of one lag distance (defaults to i=2). MAF provides the analytical solution to the joint diagonalisation of two matrices, the covariance of increments provided by the slicing and the conventional covariance matrix (of the idt transformed values, if appropriate). Resulting factors are ordered in decreasing order of spatial continuity, which might produce surprising scree-plots for those who are used to see a screeplot of a principal component analysis.

Function `UWEDGE` (Uniformly Weighted Exhaustive Diagonalization with Gauss iterations; Tichavsky and Yeredor, 2009) results carry the extra class "uwedge". The function is a wrapper on `jointDiag::uwedge` from package `jointDiag` (Gouy-Pailler, 2017). In this case the slicer is typically just a subset of indices of lag distances to consider (defaults to the nearest indexes to mininum, maximum and mean lag distances of the variogram). UWEDGE iteratively seeks for a pair of matrices (a mixing and a demixing matrices) diagonalises the set of matrices $M_1, M_2, \ldots, M_K$ given, by minimizing the target quantity

$$Q_{uwedge}(A, W) = \sum_{k=1}^{K} Tr[E_k^t \cdot E_k],$$

where $E_k = (W^t \cdot M_k \cdot W - A \cdot \Lambda_k \cdot A^t)$ and $\Lambda_k = diag(W^t \cdot M_k \cdot W)$ is the resulting diagonalized version of each matrix. Obtained factors are ordered in decreasing order of spatial continuity, which might produce surprising scree-plots for those who are used to see a screeplot of a principal component analysis.

Function `RJD` results carry the extra class "rjd". The function is a wrapper on `JADE::rjd` (Miettinen, Nordhausen and Taskinen, 2017), implementing the Rotational joint diagonalisation method (Cardoso and Souloumiac, 1996). In this case the slicer is typically just a subset of indices of lag distances to consider (defaults to the nearest indexes to mininum, maximum and mean lag distances). This algorithm also served for quasi-diagonalising a set of matrices as in UWEDGE, just

that in this case the quantity to minimise is the sum of squares of all off-diagonal elements of $A^t \cdot M_k \cdot A$ for all $k = 1, 2, \ldots K$.

All these functions produce output mimicking `princomp`, i.e. with elements

sdev contrary to the output in PCA, this contains the square root of the metric variance of the predictions obtained for each individual factor; this is the quantity needed for `screeplot` to create plots of explained variance by factor

loadings matrix of contributions of each (cdt-transformed) original variable to the new factors

center center of the data set (eventually, represented through `cdt`), in compositional methods

scale the scalings applied to each original variable

n.obs number of observations

scores the scores of the supplied data on the new factors

call the call to the function (attention: it actually may come much later)

and additionally some of the following arguments, in different order

invLoadings matrix of contributions of each factor onto each original variable

Center compositional methods return here the cdt-backtransformed center

InvLoadings compositional methods return here the clr-backtransformed inverse loadings, so that each column of this matrix can be understood as a composition on itself

DownInvLoadings compositional methods return here the clr-backtransformed "minus inverse loadings", so that each column of this matrix can be understood as a composition on itself; details in `princomp.acomp`

C1, C2 Maf returns the two matrices that were diagonalised

eigenvalues Maf returns the generalized eigenvalues of the diagonalisation of C1 and C2

gof UWEDGE returns the values of the goodness of fit criterion across sweeps

diagonalized RJD returns the diagonalized matrices, in an array of (K,D,D)-dimensions, being D the number of variables in `x`

type a string describing which package and which function was used as a workhorse for the calculation

NOTE: if the arguments you provide to RJD and UWEDGE are not of the appropriate type (i.e. data.frames or equivalent) the default method of these functions will just attempt to call the basis functions JADE:rjd and JointDiag:uwedge respectively. This will be the case if you provide `x` as a "`matrix`", or as an "`array`". In those cases, the output will NOT be structured as an extension to princomp results; instead they will be native output from those functions.

Functions

- `Maf`: Generalised diagonalisations
- `Maf.rmult`: Generalised diagonalisations
- `Maf.aplus`: Generalised diagonalisations
- `Maf.rplus`: Generalised diagonalisations
- `Maf.ccomp`: Generalised diagonalisations

- `Maf.rcomp`: Generalised diagonalisations
- `Maf.acomp`: Generalised diagonalisations
- `UWEDGE`: Generalised diagonalisations
- `UWEDGE.default`: Generalised diagonalisations
- `UWEDGE.acomp`: Generalised diagonalisations
- `UWEDGE.rcomp`: Generalised diagonalisations
- `RJD`: Generalised diagonalisations
- `RJD.default`: Generalised diagonalisations
- `RJD.acomp`: Generalised diagonalisations
- `RJD.rcomp`: Generalised diagonalisations

References

Cardoso, J. K. and Souloumiac A. 1996. Jacobi angles for simultaneous diagonalization. SIAM Journal of Matrix Analysis and Applications 17(1), 161-164.

Gouy-Pailler C., 2017. jointDiag: Joint approximate diagonalization of a set of square matrices. R package version 0.3. https://CRAN.R-project.org/package=jointDiag

Miettinen J., Nordhausen K., and Taskinen, S., 2017. Blind source separation based on Joint diagonalization in R: The packages JADE and BSSasymp. Journal of Statistical Software 76(2), 1-31.

Switzer P. and Green A.A., 1984. Min/Max autocorrelation factors for multivariate spatial imaging, Stanford University, Palo Alto, USA, 14pp.

Tichavsky, P. and Yeredor, A., 2009. Fast approximate joint diagonalization incorporating weight matrices. IEEE Transactions on Signal Processing 57, 878 – 891.

See Also

Other generalised Diagonalisations: `coloredBiplot.genDiag()`, `predict.genDiag()`

Examples

```
require("magrittr")
require("gstat")
require("compositions")
data("jura", package="gstat")
gs = gstat(id="Cd", formula=log(Cd)~1, locations=~Xloc+Yloc, data=jura.pred) %>%
gstat(id="Co", formula=log(Cd)~1, locations=~Xloc+Yloc, data=jura.pred) %>%
gstat(id="Cr", formula=log(Cr)~1, locations=~Xloc+Yloc, data=jura.pred) %>%
gstat(id="Cu", formula=log(Cu)~1, locations=~Xloc+Yloc, data=jura.pred) %>%
gstat(id="Ni", formula=log(Ni)~1, locations=~Xloc+Yloc, data=jura.pred) %>%
gstat(id="Pb", formula=log(Pb)~1, locations=~Xloc+Yloc, data=jura.pred) %>%
gstat(id="Zn", formula=log(Zn)~1, locations=~Xloc+Yloc, data=jura.pred)
vg = variogram(gs)
mf = Maf(aplus(jura.pred[, -(1:6)]), vg=vg)
mf
mf$loadings
biplot(mf)
```

`make.gmCompositionalGaussianSpatialModel` *Construct a Gaussian gmSpatialModel for regionalized compositions*

Description

Construct a regionalized compositional data container to be used for Gaussian-based geostatistics: variogram modelling, cokriging and simulation.

Usage

```
make.gmCompositionalGaussianSpatialModel(
  data,
  coords = attr(data, "coords"),
  V = "ilr",
  prefix = NULL,
  model = NULL,
  beta = model$beta,
  formula = model$formula,
  ng = NULL,
  nmax = ng$nmax,
  nmin = ng$nmin,
  omax = ng$omax,
  maxdist = ng$maxdist,
  force = ng$force
)
```

Arguments

`data`	either a `compositions::acomp()` compositional data set, or else a `sp::SpatialPointsDataFrame()` containing it
`coords`	the coordinates of the sampling locations, if no SpatialPointsDataFrame was provided
`V`	optionally, a matrix of logcontrasts, or else one of the following strings: "alr", "ilr" or "clr"; to produce a plot of the empirical variogram in the corresponding representation; default to variation-variograms
`prefix`	the desired prefix name for the logratio variables, if this is wished to be forced; otherwise derived from `V`
`model`	a variogram model, of any relevant class
`beta`	(see `formula`) the coefficients of dependence of the mean of the random field, if these are known; e.g. if `formula=~1` constant mean, and the mean is indeed known, `beta` would be a compositional mean; seldom used directly
`formula`	a formula without left-hand-side term, e.g. `~1` or `~Easting+Northing`, specifying what do we know of the dependence of the mean of the random field; this follows the same ideas than in `gstat::gstat()`

ng	optional neighborhood information, typically created with KrigingNeighbourhood()
nmax	optional, neighborhood description: maximum number of data points per cokriging system
nmin	optional, neighborhood description: minimum number of data points per cokriging system
omax	optional, neighborhood description: maximum number of data points per cokriging system per quadrant/octant
maxdist	optional, neighborhood description: maximum radius of the search neighborhood
force	optional logical, neighborhood description: if not nmin points are found inside maxdist radius, keep searching. This and all preceding arguments for neighborhood definition are borrowed from gstat::gstat()

Value

A "gmSpatialModel" object with all information provided appropriately structured. See gmSpatialModel.

See Also

SequentialSimulation(), TurningBands() or CholeskyDecomposition() for specifying the exact simulation method and its parameters, predict.gmSpatialModel() for running predictions or simulations

Other gmSpatialModel: as.gmSpatialModel(), gmSpatialModel-class, make.gmCompositionalMPSSpatialModel(), make.gmMultivariateGaussianSpatialModel(), predict.gmSpatialModel()

Examples

```
data("jura", package="gstat")
X = jura.pred[1:20,1:2]
Zc = compositions::acomp(jura.pred[1:20,7:13])
make.gmCompositionalGaussianSpatialModel(data=Zc, coords=X, V="alr")
```

---

make.gmCompositionalMPSSpatialModel

Construct a Multi-Point gmSpatialModel for regionalized compositions

---

Description

Construct a regionalized compositional data container to be used for multipoint geostatistics.

Usage

```
make.gmCompositionalMPSSpatialModel(
  data,
  coords = attr(data, "coords"),
  V = "ilr",
  prefix = NULL,
  model = NULL
)
```

Arguments

data	either a compositions::acomp() compositional data set, or else a sp::SpatialPointsDataFrame() containing it
coords	the coordinates of the sampling locations, if no SpatialPointsDataFrame was provided
V	optionally, a matrix of logcontrasts, or else one of the following strings: "alr", "ilr" or "clr"; to produce a plot of the empirical variogram in the corresponding representation; default to variation-variograms
prefix	the desired prefix name for the logratio variables, if this is wished to be forced; otherwise derived from V
model	a training image, of any appropriate class (typically a sp::SpatialGridDataFrame() or sp::SpatialPixelsDataFrame())

Value

A "gmSpatialModel" object with all information provided appropriately structured. See gmSpatialModel.

See Also

DirectSamplingParameters() for specifying a direct simulation method parameters, predict.gmSpatialModel() for running the simulation

Other gmSpatialModel: as.gmSpatialModel(), gmSpatialModel-class, make.gmCompositionalGaussianSpatialMode
make.gmMultivariateGaussianSpatialModel(), predict.gmSpatialModel()

---

make.gmMultivariateGaussianSpatialModel

Construct a Gaussian gmSpatialModel for regionalized multivariate data

---

Description

Construct a regionalized multivariate data container to be used for Gaussian-based geostatistics: variogram modelling, cokriging and simulation.

Usage

```
make.gmMultivariateGaussianSpatialModel(
  data,
  coords = attr(data, "coords"),
  model = NULL,
  beta = model$beta,
  formula = model$formula,
  ng = NULL,
  nmax = ng$nmax,
  nmin = ng$nmin,
  omax = ng$omax,
  maxdist = ng$maxdist,
  force = ng$force
)
```

Arguments

data	either a data set of any data.frame similar class, or else a sp::SpatialPointsDataFrame() containing it
coords	the coordinates of the sampling locations, if no SpatialPointsDataFrame was provided
model	a variogram model, of any relevant class
beta	(see formula) the coefficients of dependence of the mean of the random field, if these are known; e.g. if formula=~1 constant mean, and the mean is indeed known, beta would be a compositional mean; seldom used directly
formula	a formula without left-hand-side term, e.g. ~1 or ~Easting+Northing, specifying what do we know of the dependence of the mean of the random field; this follows the same ideas than in gstat::gstat()
ng	optional neighborhood information, typically created with KrigingNeighbourhood()
nmax	optional, neighborhood description: maximum number of data points per cokriging system
nmin	optional, neighborhood description: minimum number of data points per cokriging system
omax	optional, neighborhood description: maximum number of data points per cokriging system per quadrant/octant
maxdist	optional, neighborhood description: maximum radius of the search neighborhood
force	optional logical, neighborhood description: if not nmin points are found inside maxdist radius, keep searching. This and all preceding arguments for neighborhood definition are borrowed from gstat::gstat()

Value

A "gmSpatialModel" object with all information provided appropriately structured. See gmSpatialModel.

See Also

SequentialSimulation(), TurningBands() or CholeskyDecomposition() for specifying the exact simulation method and its parameters, predict.gmSpatialModel() for running predictions or simulations

Other gmSpatialModel: as.gmSpatialModel(), gmSpatialModel-class, make.gmCompositionalGaussianSpatialMode
make.gmCompositionalMPSSpatialModel(), predict.gmSpatialModel()

Examples

```
data("jura", package="gstat")
X = jura.pred[,1:2]
Zc = jura.pred[,7:13]
make.gmMultivariateGaussianSpatialModel(data=Zc, coords=X)
```

---

mean.accuracy *Mean accuracy*

Description

Mean method for accuracy curves, after Deutsch (1997)

Usage

```
## S3 method for class 'accuracy'
mean(x, ...)
```

Arguments

x	accuracy curve obtained with accuracy()
...	generic functionality, not used

Value

The value of the mean accuracy after Deutsch (1997); details can be found in precision().

See Also

Other accuracy functions: accuracy(), plot.accuracy(), precision(), validate(), xvErrorMeasures()

mean.spatialDecorrelationMeasure

Average measures of spatial decorrelation

Description

Compute average measres of spatial decorrelation out of the result of a call to `spatialDecorrelation()`

Usage

```
## S3 method for class 'spatialDecorrelationMeasure'
mean(x, ...)
```

Arguments

`x`	the outcome of a call to `spatialDecorrelation()`
`...`	further arguments to mean

Value

a mean for each measure available on `x`, i.e. this can be a vector. The output is named.

See Also

`spatialDecorrelation()` for an example

ModelStructuralFunctionSpecification-class

Structural function model specification

Description

Abstract class, containing any specification of a variogram (or covariance) model

ndirections *Number of directions of an empirical variogram*

Description

Returns the number of directions at which an empirical variogram was computed

Usage

```
ndirections(x)

## Default S3 method:
ndirections(x)

## S3 method for class 'azimuth'
ndirections(x)

## S3 method for class 'azimuthInterval'
ndirections(x)

## S3 method for class 'logratioVariogramAnisotropy'
ndirections(x)

## S3 method for class 'logratioVariogram'
ndirections(x)

## S3 method for class 'gmEVario'
ndirections(x)

## S3 method for class 'gstatVariogram'
ndirections(x)
```

Arguments

x	empirical variogram object

Value

Generic function. It provides the number of directions at which an empirical variogram was computed

Methods (by class)

- `default`: generic method
- `azimuth`: method for objects of class "azimuth" (vectors of single angles)
- `azimuthInterval`: method for objects of class "azimuthInterval" (data.frames of intervals for angles)

- `logratioVariogramAnisotropy`: method for empirical logratio variograms with anisotropy
- `logratioVariogram`: method for empirical logratio variograms without anisotropy
- `gmEVario`: method for empirical gmGeostats variograms
- `gstatVariogram`: method for empirical gstat variograms

See Also

`logratioVariogram()`, `gstat::variogram()`

Other gmEVario functions: `as.gmEVario()`, `gsi.EVario2D()`, `plot.gmEVario()`, `variogramModelPlot()`

Other gmCgram functions: `[.gmCgram()`, `[[.gmCgram()`, `as.function.gmCgram()`, `as.gmCgram.variogramModelList()`, `length.gmCgram()`, `plot.gmCgram()`, `variogramModelPlot()`

NfoldCrossValidation *Specify a strategy for validation of a spatial model*

Description

Specify a strategy to validate a spatial model. Currently only leave-one-out and n-fold cross-validation are available, each specified by its own function. Leave-one-out takes no parameter.

Usage

```
NfoldCrossValidation(nfolds = 2, doAll = TRUE, ...)
```

Arguments

`nfolds`	Either, one integer between 2 and the number of hard conditioning data, specifying how many groups do you want to split the data available; or else a factor specifying these groups
`doAll`	boolean; should each group be used once for validating the model constructed with the remaining groups; else, only the first group will be used for validation, and the other will be used for training.
`...`	ignored

Value

An object, a list with an appropriate class, controlling the strategy specified. This can be of class "NfoldCrossValidation" or of class c("LeaveOneOut", "NfoldCrossValidation").

See Also

Other validation functions: `LeaveOneOut`, `validate()`

Examples

```
NfoldCrossValidation(nfolds=5, doAll=FALSE)
```

NGSAustralia *National Geochemical Survey of Australia: soil data*

Description

A dataset containing the geochemical composition of soil samples covering approx 2/3 of Australia, at a spatial resolution of 1 sample/5000 km^2. The original data is published by Geosciences Australia under the Creative Commons CC4 Licence + Attribution (CC-BY-SA-4.0). The provided data set has additional information and some pre-treatment with respect to the one available in the references below. These changes:

Usage

```
NGSAustralia
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 5259 rows and 76 columns.

Details

- Concentrations are all reported as mg/kg, also known as parts per million (ppm). Values below detection limit are reported as minus the detection limit, as standardised in package "compositions".
- The samples were complemented with information about their belonging to one of the major crustal blocks (MCB) of Australia.
- Easting and Northing coordinates were computed using the Lambert conformal conic projection of Australia (earth ellipsoid GRS80; standard parallels: 18°S and 36ºS latitude; central meridian: 134ºE longitude).

#' @format A tibble, a data set of compound class c("tbl_df", "tbl", "data.frame") with 5259 observations and 76 variables:

ORDER Entry order

SITEID ID of the sampling site

DATE SAMPLED Timestamp of the sampling

EAST Easting coordinate of the sampling site

NORTH Northing coordinate of the sampling site

LATITUDE Latitude of the sampling site

LONGITUDE Longitude of the sampling site

STATE State of the sampling site, one of: "NWS" (New South Wales), "NT" (Northern Territory), "QLD" (Qeensland), "SA" (Southern Australia), "TAS" (Tasmania), "VIC" (Victoria) or "WA" (Western Australia)

REGION One of the three geo-regions of the data set: "EAST", "WEST" or "EUCLA"

DUPLICATE CODE Marker for duplicate samples, for quality control

SAMPLEID ID of the sample

GRAIN SIZE Particle size of the soil sample, one of "<2 mm" (coarse) or "<75 µm" (fine)

DEPTH Sampling depth, one of: "TOS" (top soil) or "BOS" (bottom soil)

CODE A combination of Grain Size and DEPTH, one of "Tc" "Tf" "Bc" "Bf", standing for TOS coarse, TOS fine, BOS coarse and BOS fine, respectively

Ag ICP-MS mg/kg 0.03 concemtration of Silver in that sample, analysed with inductive coupled plasma mass spectrometry (ICP-MS), with a detection limit of 0.03ppm

Al XRF mg/kg 26 concentration of Aluminium, analysed with X-Ray Fluorescence (XRF), detection limit 26 ppm

As ICP-MS mg/kg 0.4 concentration of Arsenic measured with ICP-MS , detection limit: 0.4 mg/kg

Au FA mg/kg 0.001 concentration of Gold measured with fire assay (FA) , detection limit: 0.001 mg/kg

Ba ICP-MS mg/kg 0.5 concentration of Barium measured with ICP-MS , detection limit: 0.5 mg/kg

Be ICP-MS mg/kg 1.1 concentration of Berillium measured with ICP-MS , detection limit: 1.1 mg/kg

Bi ICP-MS mg/kg 0.02 concentration of Bismuth measured with ICP-MS , detection limit: 0.02 mg/kg

Ca XRF mg/kg 14 concentration of Calcium measured with XRF , detection limit: 14 mg/kg

Cd ICP-MS mg/kg 0.1 concentration of Cadmium measured with ICP-MS , detection limit: 0.1 mg/kg

Ce ICP-MS mg/kg 0.03 concentration of Cerium measured with ICP-MS , detection limit: 0.03 mg/kg

Cl XRF mg/kg 10 concentration of Chlorine measured with XRF , detection limit: 10 mg/kg

Co ICP-MS mg/kg 0.1 concentration of Cobalt measured with ICP-MS , detection limit: 0.1 mg/kg

Cr ICP-MS mg/kg 0.5 concentration of Cromium measured with ICP-MS , detection limit: 0.5 mg/kg

Cs ICP-MS mg/kg 0.1 concentration of Caesium measured with ICP-MS , detection limit: 0.1 mg/kg

Cu ICP-MS mg/kg 0.2 concentration of Copper measured with ICP-MS , detection limit: 0.2 mg/kg

Dy ICP-MS mg/kg 0.1 concentration of Dysprosium measured with ICP-MS , detection limit: 0.1 mg/kg

Er ICP-MS mg/kg 0.03 concentration of Erbium measured with ICP-MS , detection limit: 0.03 mg/kg

Eu ICP-MS mg/kg 0.03 concentration of Europium measured with ICP-MS , detection limit: 0.03 mg/kg

F ISE mg/kg 20 concentration of Fluoride measured with Ion Selective Electrode (ISE), detection limit: 20 mg/kg

FeT XRF mg/kg 35 concentration of total Iron measured with XRF , detection limit: 35 mg/kg

Ga ICP-MS mg/kg 0.1 concentration of Gallium measured with ICP-MS , detection limit: 0.1 mg/kg

Gd ICP-MS mg/kg 0.03 concentration of Gadolinium measured with ICP-MS , detection limit: 0.03 mg/kg

Ge ICP-MS mg/kg 0.04 concentration of Germanium measured with ICP-MS , detection limit: 0.04 mg/kg

Hf ICP-MS mg/kg 0.04 concentration of Hafnium measured with ICP-MS , detection limit: 0.04 mg/kg

Ho ICP-MS mg/kg 0.02 concentration of Holmium measured with ICP-MS , detection limit: 0.02 mg/kg

K XRF mg/kg 42 concentration of Potassium measured with XRF , detection limit: 42 mg/kg

La ICP-MS mg/kg 0.1 concentration of Lantanum measured with ICP-MS , detection limit: 0.1 mg/kg

Lu ICP-MS mg/kg 0.02 concentration of Lutetium measured with ICP-MS , detection limit: 0.02 mg/kg

Mg XRF mg/kg 60 concentration of Magnesium measured with XRF , detection limit: 60 mg/kg

Mn XRF mg/kg 39 concentration of Manganese measured with XRF , detection limit: 39 mg/kg

Mo ICP-MS mg/kg 0.3 concentration of Molybdenum measured with ICP-MS , detection limit: 0.3 mg/kg

Na XRF mg/kg 74 concentration of Sodium measured with XRF , detection limit: 74 mg/kg

Nb ICP-MS mg/kg 0.03 concentration of Niobium measured with ICP-MS , detection limit: 0.03 mg/kg

Nd ICP-MS mg/kg 0.1 concentration of Neodymium measured with ICP-MS , detection limit: 0.1 mg/kg

Ni ICP-MS mg/kg 0.5 concentration of Nickel measured with ICP-MS , detection limit: 0.5 mg/kg

P XRF mg/kg 22 concentration of Phosphorus measured with XRF , detection limit: 22 mg/kg

Pb ICP-MS mg/kg 0.1 concentration of Lead measured with ICP-MS , detection limit: 0.1 mg/kg

Pd FA mg/kg 0.001 concentration of Palladium measured with FA , detection limit: 0.001 mg/kg

Pr ICP-MS mg/kg 0.02 concentration of Praseodymium measured with ICP-MS , detection limit: 0.02 mg/kg

Pt FA mg/kg 0.0005 concentration of Platinum measured with FA , detection limit: 0.0005 mg/kg

Rb ICP-MS mg/kg 0.2 concentration of Rubidium measured with ICP-MS , detection limit: 0.2 mg/kg

S XRF mg/kg 10 concentration of Sulphur measured with XRF , detection limit: 10 mg/kg

Sb ICP-MS mg/kg 0.4 concentration of Antimony measured with ICP-MS , detection limit: 0.4 mg/kg

Sc ICP-MS mg/kg 0.3 concentration of Scandium measured with ICP-MS , detection limit: 0.3 mg/kg

Si XRF mg/kg 47 concentration of Silicon measured with XRF , detection limit: 47 mg/kg

Sm ICP-MS mg/kg 0.04 concentration of Samarium measured with ICP-MS , detection limit: 0.04 mg/kg

Sn ICP-MS mg/kg 0.2 concentration of Tin measured with ICP-MS , detection limit: 0.2 mg/kg

Sr ICP-MS mg/kg 0.2 concentration of Strontium measured with ICP-MS , detection limit: 0.2 mg/kg

Ta ICP-MS mg/kg 0.02 concentration of Tantalum measured with ICP-MS , detection limit: 0.02 mg/kg

Tb ICP-MS mg/kg 0.02 concentration of Terbium measured with ICP-MS , detection limit: 0.02 mg/kg

Th ICP-MS mg/kg 0.02 concentration of Thorium measured with ICP-MS , detection limit: 0.02 mg/kg

Ti XRF mg/kg 30 concentration of Titanium measured with XRF , detection limit: 30 mg/kg

U ICP-MS mg/kg 0.02 concentration of Uranium measured with ICP-MS , detection limit: 0.02 mg/kg

V ICP-MS mg/kg 0.1 concentration of Vanadium measured with ICP-MS , detection limit: 0.1 mg/kg

W ICP-MS mg/kg 0.1 concentration of Wolframium measured with ICP-MS , detection limit: 0.1 mg/kg

Y ICP-MS mg/kg 0.05 concentration of Yttrium measured with ICP-MS , detection limit: 0.05 mg/kg

Yb ICP-MS mg/kg 0.04 concentration of Ytterbium measured with ICP-MS , detection limit: 0.04 mg/kg

Zn ICP-MS mg/kg 0.9 concentration of Zink measured with ICP-MS , detection limit: 0.9 mg/kg

Zr ICP-MS mg/kg 0.2 concentration of Zirconium measured with ICP-MS , detection limit: 0.2 mg/kg #'

LOI CALC mg/kg concentration of Loss-on-Ignition, measured by calcination

LOI CALC mg/kg concentration of Loss-On-Ignition measured with Calcination , detection limit: 0.2 mg/kg #'

LOI CALC mg/kg concentration of Loss-on-Ignition, measured by calcination

MRB obtained from simplifying the major crustal boundaries of Korsch2015a, Korsch2015b

License

CC BY-SA 4.0

Source

https://www.ga.gov.au/about/projects/resources/national-geochemical-survey

`noSpatCorr.test` *Test for lack of spatial correlation*

Description

Permutation test for checking lack of spatial correlation.

Usage

```
noSpatCorr.test(Z, ...)

## S3 method for class 'data.frame'
noSpatCorr.test(Z, X, ...)

## Default S3 method:
noSpatCorr.test(Z, ...)

## S3 method for class 'matrix'
noSpatCorr.test(
  Z,
  X,
  R = 299,
  maxlag0 = 0.1 * max(as.matrix(dist(X))),
  minlagInf = 0.25 * max(as.matrix(dist(X))),
  ...
)
```

Arguments

`Z`	matrix (or equivalent) of scaled observations
`...`	extra arguments for generic functionality
`X`	matrix (or equivalent) of sample location coordinates
`R`	number of realizations of the Monte Carlo test
`maxlag0`	maximum lag distance to consider in the short range covariance
`minlagInf`	minimum lag distance to consider in the long range covariance

Value

Produces a test of lack of spatial correlation by means of permutations. The test statistic is based on the smallest eigenvalue of the generalised eigenvalues of the matrices of covariance for short range and for long range.

Methods (by class)

- `data.frame`: Test for lack of spatial correlation
- `default`: Test for lack of spatial correlation, works only for Spatial objects with a "data" slot
- `matrix`: Test for lack of spatial correlation

Examples

```
data("jura", package="gstat")
X = jura.pred[, 1:2]
Z = data.frame(compositions::ilr(jura.pred[,-(1:6)]))
noSpatCorr.test(Z=Z, X=X)
# now destroy the spatial structure reshuffling the coordinates:
ip = sample(nrow(X))
noSpatCorr.test(Z=Z, X=X[ip,])
```

`pairsmap` *Multiple maps Matrix of maps showing different combinations of components of a composition, user defined*

Description

Multiple maps Matrix of maps showing different combinations of components of a composition, user defined

Usage

```
pairsmap(data, ...)

## S3 method for class 'SpatialPointsDataFrame'
pairsmap(data, ...)

## Default S3 method:
pairsmap(
  data,
  loc,
  colscale = rev(rainbow(10, start = 0, end = 4/6)),
  cexrange = c(0.1, 2),
  scale = rank,
  commonscale = FALSE,
 mfrow = c(floor(sqrt(ncol(data))), ceiling(ncol(data)/floor(sqrt(ncol(data))))),
  foregroundcolor = "black",
  closeplot = TRUE,
  ...
)
```

Arguments

`data`	data to represent; admits many data containing objects (matrix, data.frame, classes from package `compositions`) as well as their `Spatial` counterparts (in which case, `loc` is not necessary)
`...`	extra arguments for generic functionality

loc	matrix or data.frame of coordinates of the sample locations
colscale	set of colors to be used as colorscale (defauls to 10 colors between blue and red)
cexrange	symbol size min and max values (default to 0.1 to 2)
scale	function scaling the set of z-values of each map, defaults to rank
commonscale	logical, should all plots share a common z-scale? defaults to FALSE
mfrow	vector of two integers, giving the number of plots per row and per column of the matrix of plots to be created; defaults to something square-ish, somewhat wider than longer, and able to contain all plots
foregroundcolor	color to be used for the border of the symbol
closeplot	logical, should the plot be left open (FALSE) for further changes, or be frozen (TRUE)? defaults to TRUE

Value

The function is primarily called for producing a matrix of plots of each component of a multivariate data set, such as a compositional data set. Each plot represents a map whose symbols are colored and sized according to a z-scale controlled by one of the variables of the data set. It can be used virtually with any geometry, any kind of data (compositional, positive, raw, etc). This function returns invisibly the graphical parameters that were active prior to calling this function. This allows the user to add further stuff to the plots (mostly, using `par(mfg=c(i,j))` to plot on the diagram (i,j)), or else freeze the plot (by wrapping the call to `pwlrmap` on a call to `par`).

Methods (by class)

- `SpatialPointsDataFrame`: Multiple maps
- `default`: Multiple maps

Examples

```
data("Windarling")
library("compositions")
coords = as.matrix(Windarling[,c("Easting","Northing")])
compo = Windarling[,c("Fe","Al2O3","SiO2", "Mn", "P")]
compo$Rest = 100-rowSums(compo)
compo = acomp(compo)
pairsmap(data=clr(compo), loc=coords) # clr
pairsmap(data=compo, loc=coords) # closed data
```

---

plot.accuracy *Plot method for accuracy curves*

Description

Plot an accuracy curve out of the outcome of `accuracy()`.

Usage

```
## S3 method for class 'accuracy'
plot(
  x,
  xlim = c(0, 1),
  ylim = c(0, 1),
  xaxs = "i",
  yaxs = "i",
  type = "o",
  col = "red",
  asp = 1,
  xlab = "confidence",
  ylab = "coverage",
  pty = "s",
  main = "accuracy plot",
  colref = col[1],
  ...
)
```

Arguments

x	an accuracy() object
xlim	graphical parameters, see graphics::plot.default()
ylim	graphical parameters, see graphics::plot.default()
xaxs	graphical parameters, see graphics::plot.default()
yaxs	graphical parameters, see graphics::plot.default()
type	graphical parameters, see graphics::plot.default()
col	graphical parameters, see graphics::plot.default()
asp	graphical parameters, see graphics::plot.default()
xlab	graphical parameters, see graphics::plot.default()
ylab	graphical parameters, see graphics::plot.default()
pty	graphical parameters, see graphics::plot.default()
main	graphical parameters, see graphics::plot.default()
colref	color for the reference line 1:1
...	further graphical parameters to graphics::plot.default()

Value

Nothing, called to plot the accuracy curve $\{(p_i, \pi_i), i = 1, 2, \ldots, I\}$, where $\{p_i\}$ are a sequence of nominal confidence of prediction intervals and each π_i is the actual coverage of an interval with nominal confidence p_i.

See Also

Other accuracy functions: accuracy(), mean.accuracy(), precision(), validate(), xvErrorMeasures()

plot.gmCgram *Draw cuves for covariance/variogram models*

Description

Represent a gmCgram object as a matrix of lines in several plots

Usage

```
## S3 method for class 'gmCgram'
plot(
  x,
  xlim.up = NULL,
  xlim.lo = NULL,
  vdir.up = NULL,
  vdir.lo = NULL,
  xlength = 200,
  varnames = colnames(x$nugget),
  add = FALSE,
  commonAxis = TRUE,
  cov = TRUE,
  closeplot = TRUE,
  ...
)
```

Arguments

x	object to draw, of class gmCgram // curently only valid for symmetric functions
xlim.up	range of lag values to use in plots of the upper triangle
xlim.lo	range of lag values to use in plots of the lower triangle
vdir.up	geograohic directions to represent in the upper triangle
vdir.lo	geograohic directions to represent in the lower triangle
xlength	number of discretization points to use for the curves (defaults to 200)
varnames	string vector, variable names to use in the labelling of axes
add	logical, should a new plot be created or stuff be added to an existing one?
commonAxis	logical, is a common Y axis for all plots in a row desired?
cov	logical, should the covariance function (=TRUE) or the variogram (=FALSE) be plotted?
closeplot	logical, should the plot be left open (FALSE) for further changes, or be frozen (TRUE)? defaults to TRUE
...	further graphical parameters for the plotting function

Value

This function is called for its side effect of producing a plot: the plot will be open to further changes if you provide `closeplot=FALSE`. Additionally, the function invisibly returns the graphical parameters that were active before starting the plot. Hence, if you want to freeze a plot and not add anymore to it, you can do `par(plot(x,closeplot=FALSE,...))`, or `plot(x,closeplot=TRUE,...)`. If you want to further add stuff to it, better just call `plot(x,closeplot=FALSE,...)`. The difference is only relevant when working with the screen graphical device.

See Also

Other gmCgram functions: `[.gmCgram()`, `[[.gmCgram()`, `as.function.gmCgram()`, `as.gmCgram.variogramModelList()`, `length.gmCgram()`, `ndirections()`, `variogramModelPlot()`

Examples

```
utils::data("variogramModels")
v1 = setCgram(type=vg.Gau, sill=diag(3)-0.5, anisRanges = 2*diag(c(3,0.5)))
v2 = setCgram(type=vg.Exp, sill=0.3*diag(3), anisRanges = 0.5*diag(2))
vm = v1+v2
plot(vm)
plot(vm, cov=FALSE)
```

---

plot.gmEVario *Plot empirical variograms*

Description

Flexible plot of an empirical variogram of class gmEVario

Usage

```
## S3 method for class 'gmEVario'
plot(
  x,
  xlim.up = NULL,
  xlim.lo = NULL,
  vdir.up = NULL,
  vdir.lo = NULL,
  varnames = dimnames(x$gamma)[[2]],
  type = "o",
  add = FALSE,
  commonAxis = TRUE,
  cov = attr(x, "type") == "covariance",
  closeplot = TRUE,
  ...
)
```

Arguments

x	object to print, of class gmEVario
xlim.up	range of X values to be used for the diagrams of the upper triangle
xlim.lo	range of X values to be used for the diagrams of the lower triangle
vdir.up	in case of anisotropic variograms, indices of the directions to be plotted on the upper triangle
vdir.lo	..., indices of the directions to be plotted on the lower triangle
varnames	variable names to be used
type	string, controlling whether to plot lines, points, etc (see plot)
add	boolean, add stuff to an existing diagram?
commonAxis	boolean, should vertical axes be shared by all plots in a row?
cov	boolean, is this a covariance? (if FALSE, it is a variogram)
closeplot	logical, should the plot be left open (FALSE) for further changes, or be frozen (TRUE)? defaults to TRUE
...	further parameters to matplot

Value

invisibly, the graphical parameters active before calling the function. This is useful for freezing the plot if you provided closeplot=FALSE.

How to use arguments vdir.lo and vdir.up? Each empirical variogram x has been computed along certain distances, recorded in its attributes and retrievable with command ndirections.

See Also

Other gmEVario functions: as.gmEVario(), gsi.EVario2D(), ndirections(), variogramModelPlot()

Examples

```
library(gstat)
data("jura", package = "gstat")
X = as.matrix(jura.pred[,1:2])
Z = as.matrix(jura.pred[,c("Zn","Cd","Pb")])
vge = gsi.EVario2D(X,Z)
plot(vge)
plot(vge, pch=22, lty=1, bg="grey")
```

`plot.logratioVariogramAnisotropy` *Plot variogram lines of empirical directional logratio variograms*

Description

Plots an "logratioVariogramAnisotropy" object in a series of panels, with each direction represented as a broken line.

Usage

```
## S3 method for class 'logratioVariogramAnisotropy'
plot(
  x,
  azimuths = colnames(x),
  col = rev(rainbow(length(azimuths))),
  type = "o",
  V = NULL,
  lty = 1,
  pch = 1:length(azimuths),
  model = NULL,
  figsp = 0,
  closeplot = TRUE,
  ...
)
```

Arguments

`x`	logratio variogram with anisotropy, i.e. object of class c("logratioVariogramAnisotropy", "logratioVariogram")
`azimuths`	which directions do you want to plot? default: all directions available
`col`	colors to be used for plotting
`type`	type of representation, see `graphics::plot()`
`V`	optionally, a matrix of logcontrasts, or else one of the following strings: "alr", "ilr" or "clr"; to produce a plot of the empirical variogram in the corresponding representation; default to variation-variograms
`lty`	style of the lines, potentially different for each directions
`pch`	symbols for the points, potentially different for each directions
`model`	eventually, variogram model to plot on top of the empirical variogram
`figsp`	spacing between the several panels, if desired
`closeplot`	boolean, should the plotting par() be returned to the starting values? (defaults to TRUE; see `plot.gmCgram()` for details)
`...`	additional graphical arguments, to be passed to the underlying `graphics::matplot()` function

Value

Nothing. The function is called to create a plot.

Examples

```
data("jura", package="gstat")
X = jura.pred[,1:2]
Zc = compositions::acomp(jura.pred[,7:9])
vg = logratioVariogram(data=Zc, loc=X, azimuth=c(0:3)*45,
                         azimuth.tol=22.5)
plot(vg)
```

`plot.swarmPlot` *Plotting method for swarmPlot objects*

Description

Produces a plot for objects obtained by means of `swarmPlot()`. These are lists of two-element list, respectively giving the (x,y) coordinates of a series of points defining a curve. They are typically obtained out of a `DataFrameStack()`, and each curve represents a a certain relevant summary of one of the elements of the stack. All parameters of `graphics::plot.default()` are available, plus the following

Usage

```
## S3 method for class 'swarmPlot'
plot(
  x,
  type = "l",
  col = "grey",
  xlim = NULL,
  ylim = NULL,
  xlab = "x",
  ylab = "y",
  main = NULL,
  asp = NA,
  sub = NULL,
  log = "",
  ann = par("ann"),
  axes = TRUE,
  frame.plot = axes,
  bg = NA,
  pch = 1,
  cex = 1,
  lty = 1,
  lwd = 1,
  add = FALSE,
```

```
  ...
)
```

Arguments

x	swarmPlot object
type	see graphics::plot.default()
col	color to be used for all curves and points
xlim	limits of the X axis, see graphics::plot.default()
ylim	limits of the Y axis, see graphics::plot.default()
xlab	label for the X axis, see graphics::plot.default()
ylab	label for the Y axis, see graphics::plot.default()
main	main plot title, see graphics::plot.default()
asp	plot aspect ratio, see graphics::plot.default()
sub	plot subtitle, see graphics::plot.default()
log	log scale for any axis? see graphics::plot.default()
ann	plot annotation indications, see graphics::plot.default()
axes	should axes be set? see graphics::plot.default()
frame.plot	should the plot be framed? see graphics::plot.default()
bg	fill color for the data points if certain pch symbols are used
pch	symbol for the data points, see graphics::points()
cex	symbol size for the data points
lty	line style for the curves
lwd	line thickness for the curves
add	logical, add results to an existing plot?
...	further parameters to the plotting function

Value

Nothing. The function is called to make a plot.

Examples

```
dm = list(point=1:100, var=LETTERS[1:2], rep=paste("r",1:5, sep=""))
ar = array(rnorm(1000), dim=c(100,2,5), dimnames = dm)
dfs = DataFrameStack(ar, stackDim="rep")
rs = swarmPlot(dfs, PLOTFUN=function(x) density(x[,1]), .plotargs=FALSE)
plot(rs, col="yellow", lwd=3)
```

precision *Precision calculations*

Description

Precision calculations

Usage

```
precision(x, ...)

## S3 method for class 'accuracy'
precision(x, ...)
```

Arguments

x	an object from which precision is to be computed
...	generic functionality, not used

Value

output depends on input and meaning of the function (the term precision is highly polysemic)

Methods (by class)

- accuracy: Compute precision and goodness for accuracy curves, after Deutsch (1997), using the accuracy curve obtained with accuracy(). This returns a named vector with two values, one for precision and one for goodness.

  Mean accuracy, precision and goodness were defined by Deutsch (1997) for an accuracy curve $\{(p_i, \pi_i), i = 1, 2, \ldots, I\}$, where $\{p_i\}$ are a sequence of nominal confidence of prediction intervals and each π_i is the actual coverage of an interval with nominal confidence p_i. Out of these values, the mean accuracy (see mean.accuracy()) is computed as

$$A = \int_0^1 I\{(\pi_i - p_i) > 0\} dp,$$

  where the indicator $I\{(\pi_i - p_i) > 0\}$ is 1 if the condition is satisfied and 0 otherwise. Out of it, the area above the 1:1 bisector and under the accuracy curve is the precision $P = 1 - 2\int_0^1 (\pi_i - p_i) \cdot I\{(\pi_i - p_i) > 0\} dp$, which only takes into account those points of the accuracy curve where $\pi_i > p_i$. To consider the whole curve, goodness can be used

$$G = 1 - \int_0^1 (\pi_i - p_i) \cdot (3 \cdot I\{(\pi_i - p_i) > 0\} - 2) dp.$$

See Also

Other accuracy functions: accuracy(), mean.accuracy(), plot.accuracy(), validate(), xvErrorMeasures()

predict.genDiag *Predict method for generalised diagonalisation objects*

Description

Predict method for generalised diagonalisation objects

Usage

```
## S3 method for class 'genDiag'
predict(object, newdata = NULL, ...)
```

Arguments

object	a generalized diagonalisation object, as obtained from a call to Maf, and on the same page, information on the other diagonalisation methods UWEDGE or RJD
newdata	a matrix or data.frame of factor scores to convert back to the original scale (default: the scores element from object)
...	not used, kept for generic compatibility

Value

A data set or compositional object of the nature of the original data used for creating the genDiag object.

See Also

Other generalised Diagonalisations: Maf(), coloredBiplot.genDiag()

Examples

```
data("jura", package="gstat")
juracomp = compositions::acomp(jura.pred[, -(1:6)])
lrvg = logratioVariogram(data=juracomp, loc=jura.pred[,1:2])
mf = Maf(juracomp, vg=lrvg)
mf
biplot(mf)
predict(mf)
unclass(predict(mf)) - unclass(juracomp) # predict recovers the original composition
```

predict.gmSpatialModel *Predict method for objects of class 'gmSpatialModel'*

Description

This is a one-entry function for several spatial prediction and simulation methods, for model objects of class gmSpatialModel. The several methods are chosen by means of pars objects of the appropriate class.

Usage

```
## S3 method for class 'gmSpatialModel'
predict(object, newdata = NULL, pars = object@parameters, ...)
```

Arguments

object	a complete "gmSpatialModel", containing conditioning data and unconditional model
newdata	a collection of locations where a prediction/simulation is desired; this is typically a sp::SpatialPoints(), a data.frame or similar of X-Y(-Z) coordinates; or perhaps for gridded data an object of class sp::GridTopology(), sp::SpatialGrid() or sp::SpatialPixels()
pars	parameters describing the method to use, *encloded in an object of appropriate class* (see below)
...	further parameters for generic functionality, currently ignored

Details

Package "gmGeostats" aims at providing a broad series of algorithms for geostatistical prediction and simulation. All can be accesses through this interface, provided that arguments object and pars are of the appropriate kind. In object, the most important criterion is the nature of its slot model. In pars its class counts: for the creation of informative parameters in the appropriate format and class, a series of accessory functions are provided as well.

Classical (gaussian-based two-point) geostatistics are obtained if object@model contains a covariance function, or a variogram model. Argument pars can be created with functions such as KrigingNeighbourhood(), SequentialSimulation(), TurningBands() or CholeskyDecomposition() to respectively trigger a cokriging, as sequential Gaussian simulation, a turning bands simulation, or a simulation via Cholesky decomposition. The kriging neighbourhood can as well be incorporated in the "gmSpatialModel" object directly, or even be nested in a "SequentialSimulation" parameter object.

Conversely, to run a multipoint geostatistics algorithm, the first condition is that object@model contains a training image. Additionally, pars must describe the characteristics of the algorithm to use. Currently, only direct sampling is available: it can be obtained by providing some parameter object created with a call to DirectSamplingParameters(). Currently it is also necessary that newdata is a gridded set of locations.

Value

Depending on the nature of newdata, the result will be a data container of the same kind, extended with the predictions or simulations. For instance, if we want to obtain predictions on the locations of a "SpatialPoints", the result will be a sp::SpatialPointsDataFrame(); if we want to obtain simulations on the coordinates provided by a "data.frame", the result will be a DataFrameStack() with the spatial coordinates stored as an extra attribute; or if the input for a simulation is a masked grid of class sp::SpatialPixels(), the result will be of class sp::SpatialPixelsDataFrame() which data slot will be a DataFrameStack.

See Also

Other gmSpatialModel: as.gmSpatialModel(), gmSpatialModel-class, make.gmCompositionalGaussianSpatialMode
make.gmCompositionalMPSSpatialModel(), make.gmMultivariateGaussianSpatialModel()

---

predict.LMCAnisCompo *Compute model variogram values Evaluate the variogram model provided at some lag vectors*

Description

Compute model variogram values

Evaluate the variogram model provided at some lag vectors

Usage

```
## S3 method for class 'LMCAnisCompo'
predict(object, newdata, ...)
```

Arguments

object	variogram model
newdata	matrix or data.frame of lag vectors
...	extra arguments for generic compatbility

Value

an array of dimension (nr of lags, D, D) with D the number of variables in the model.

Examples

```
data("jura", package="gstat")
Zc = compositions::acomp(jura.pred[,7:9])
lrmd = LMCAnisCompo(Zc, models=c("nugget", "sph"), azimuths=c(0,45))
predict(lrmd, outer(0:5, c(0,1)))
```

print.mask *Print method for mask objects*

Description

Print method for mask objects. See `constructMask()` for examples. If you want to see the whole content of the mask, then use `unclass(...)`

Usage

```
## S3 method for class 'mask'
print(x, ...)
```

Arguments

`x`	mask to print
`...`	ignored

Value

the summary of number of nodes inside/outside the mask

See Also

Other masking functions: `constructMask()`, `getMask()`, `setMask()`, `unmask()`

pwlrmap *Compositional maps, pairwise logratios Matrix of maps showing different combinations of components of a composition, in pairwise logratios*

Description

Compositional maps, pairwise logratios Matrix of maps showing different combinations of components of a composition, in pairwise logratios

Usage

```
pwlrmap(
  loc,
  comp,
  colscale = rev(rainbow(10, start = 0, end = 4/6)),
  cexrange = c(0.1, 2),
  scale = rank,
  commonscale = FALSE,
  foregroundcolor = "black",
  closeplot = TRUE
)
```

Arguments

loc	matrix or data.frame of coordinates of the sample locations
comp	composition observed at every location, can be a matrix, a data.frame or of one of the classes compositions::acomp or compositions::aplus
colscale	set of colors to be used as colorscale (defauls to 10 colors between blue and red)
cexrange	symbol size min and max values (default to 0.1 to 2)
scale	function scaling the set of z-values of each map, defaults to rank
commonscale	logical, should all plots share a common z-scale? defaults to FALSE
foregroundcolor	color to be used for the border of the symbol
closeplot	logical, should the plot be left open (FALSE) for further changes, or be frozen (TRUE)? defaults to TRUE

Value

The function is primarily called for producing a matrix of (D,D) plots of the D-part compositional samples, where at each plot we represent a map whose symbols are colored and sized according to a z-scale controlled by a different logratio. For each plot, this is the logratio of the row variable by the column variable. However, in case that closeplot=FALSE, this function returns invisibly the graphical parameters that were active prior to calling this function. This allows the user to add further stuff to the plots (mostly, using par(mfg=c(i,j)) to plot on the diagram (i,j)), or manually freeze the plot (by wrapping the call to pwlrmap on a call to par).

Examples

```
data("Windarling")
coords = as.matrix(Windarling[,c("Easting","Northing")])
compo = Windarling[,c("Fe","Al2O3","SiO2", "Mn", "P")]
compo$Rest = 100-rowSums(compo)
compo = compositions::acomp(compo)
# in quantiles (default, ranking controls color and size)
pwlrmap(coords, compo)

# in logratios (I=identity)
pwlrmap(coords, compo, scale=I)
# in ratios (i.e., apply exp)
pwlrmap(coords, compo, scale=exp)
# use only color, no change in symbol size
pwlrmap(coords, compo, cexrange=c(1,1))
# change all
pwlrmap(coords, compo, commonscale=TRUE, cexrange=c(1.2,1.2),
                    colscale=rev(rainbow(40, start=0, end=4/6)))
```

SequentialSimulation *Create a parameter set specifying a gaussian sequential simulation algorithm*

Description

Create a parameter set describing a sequential simulation algorithm to two-point simulation, mostly for covariance or variogram-based gaussian random fields.

Usage

```
SequentialSimulation(nsim = 1, ng = NULL, rank = Inf, debug.level = 1, ...)
```

Arguments

nsim	number of realisations desired
ng	a neighbourhood specification, as obtained with function KrigingNeighbourhood()
rank	currently ignored (future functionality: obtain a reduced-rank simulation)
debug.level	degree of verbosity of results; negative values produce a progress bar; values can be extracted from gstat::predict.gstat()
...	further parameters, currently ignored

Value

an S3-list of class "gmSequentialSimulation" containing the four elements given as arguments to the function. This is just a compact way to provide further functions such as predict.gmSpatialModel() with appropriate triggers for choosing a prediction method or another, in this case for triggering sequential Gaussian simulation.

Examples

```
data("jura", package="gstat")
X = jura.pred[,1:2]
summary(X)
Zc = jura.pred[,7:10]
ng_local = KrigingNeighbourhood(maxdist=1, nmin=4, omax=5, force=TRUE)
(sgs_local = SequentialSimulation(nsim=100, ng=ng_local, debug.level=-1))
## then run predict(..., pars=sgs_local)
```

`setCgram` *gmGeostats Variogram models set up a D-variate variogram models*

Description

gmGeostats Variogram models set up a D-variate variogram models

Usage

```
setCgram(type, nugget = sill * 0, sill, anisRanges, extraPar = 0)
```

Arguments

`type`	constant indicating the model of correlation function
`nugget`	(DxD)-matrix for the nugget effect
`sill`	(DxD)-matrix for the partial sills of the correlation function
`anisRanges`	2x2 or 3x3 matrix of ranges (see details)
`extraPar`	for certain correlation functions, extra parameters (smoothness, period, etc)

Details

The argument `type` must be an integer indicating the model to be used. Some constants are available to make reading code more understandable. That is, you can either write `1`, `vg.Sph` or `vg.Spherical`, they will all work and produce a spherical model. The same applies for the following models: `vg.Gauss=0`; `vg.Exp=vg.Exponential=2`. These constants are available after calling `data("variogramModels")`. No other model is currently available, but this data object will be regularly updated. The constant vector `gsi.validModels` contains all currently valid models.

Argument `anisRange` expects a matrix M such that

$$h^2 = (\mathbf{x}_i - \mathbf{x}_j) \cdot M^{-1} \cdot (\mathbf{x}_i - \mathbf{x}_j)^t$$

is the (square of) the lag distance to be fed into the correlation function.

Value

an object of class "gmCgram" containing the linear model of corregionalization of the nugget and the structure given.

Examples

```
utils::data("variogramModels") # shortcut for all model constants
v1 = setCgram(type=vg.Gau, sill=diag(2), anisRanges = 3*diag(c(3,1)))
v2 = setCgram(type=vg.Exp, sill=0.3*diag(2), anisRanges = 0.5*diag(2))
vm = v1+v2
plot(vm)
```

setGridOrder *Set or get the ordering of a grid*

Description

Specify or retrieve the ordering in which a grid is stored in a vector (or matrix).

Usage

```
setGridOrder(x, refpoint, cycle)

setGridOrder_sp(x, G = 2)

setGridOrder_array(x, G = 2)

gridOrder_sp(G = 2)

gridOrder_gstat(G = 2)

gridOrder_array(G = 2)

gridOrder_GSLib(G = 2)
```

Arguments

x	a data container for the elements of the grid; the grid order is stored as an attribute to it
refpoint	a string specifying which point of the grid corresponds to the first element of x; see below
cycle	a permutation of the integers 1:G (see below)
G	number of geographic dimensions of the setting, typically G=2 or G=3

Details

A "gridOrder" attribute is a list consisting of two named elements:

refpoint one of "topleft", "bottomleft", "topright" or "bottomright" in 2D, or also of "topleftsurf", "bottomleftsurf", "toprightsurf", "bottomrightsurf", "topleftdeep","bottomleftdeep", "toprightdeep" or "bottomrightdeep" in 3D ("deep" is accessory, i.e. "topleft"== "topleftdeep"), indicating the location on the grid of the first point of the object x

cycle a permutation of 1:G indicating in which order run the dymensions, from faster to slower

Thus, a conventional ordering of a (nX*nY)-element vector into a matrix to plot with graphics::image() corresponds to an refpoint="bottomleft" and cycle=1:2, i.e. start with the lower left corner and run first by rows (eastwards), then by columns (northwards). This is constructed by gridOrder_array(G), and can be directly set to an object x by setGridOrder_array(x,G); gridOrder_GSLib is an alias for gridOrder_array.

The grids from package "sp" (and many other in R), on the contrary follow the convention `refpoint="topleft"` and `cycle=1:2`, i.e. start with the upper left corner and run first by rows (eastwards), then by columns (**southwards**). This is constructed by `gridOrder_sp(G)`, and can be directly set to an object `x` by `setGridOrder_sp(x,G)`; `gridOrder_gstat` is an alias for `gridOrder_sp`.

Value

`setGridOrder(x,...)` returns the object `x` with the grid order description attached as an attribute "gridOrder"; `getGridOrder(x)` retrieves this attribute and returns it.

Functions

- `setGridOrder_sp`: Set or get the ordering of a grid
- `setGridOrder_array`: Set or get the ordering of a grid
- `gridOrder_sp`: Set or get the ordering of a grid
- `gridOrder_gstat`: Set or get the ordering of a grid
- `gridOrder_array`: Set or get the ordering of a grid
- `gridOrder_GSLib`: Set or get the ordering of a grid

See Also

`sortDataInGrid()` for ways of reordering a grid

Examples

```
gt = sp::GridTopology(cellcentre.offset=c(1,11), cellsize=c(1,1), cells.dim=c(5,3))
sp::coordinates(sp::SpatialGrid(grid=gt))
gridOrder_sp(2)
```

---

setMask *Set a mask on an object*

Description

Set a mask on an object See `constructMask()` for examples on how to construct masks.

Usage

```
setMask(x, ...)

## Default S3 method:
setMask(x, mask, coordinates = 1:2, ...)

## S3 method for class 'data.frame'
setMask(x, mask, coordinates = 1:2, ...)

## S3 method for class 'DataFrameStack'
```

```
setMask(x, mask, coordinates = attr(x, "coordinates"), ...)

## S3 method for class 'SpatialGrid'
setMask(x, mask, ...)

## S3 method for class 'GridTopology'
setMask(x, mask, ...)

## S3 method for class 'SpatialPoints'
setMask(x, mask, ...)
```

Arguments

x	an object to mask (for set) or masked (for get)
...	extra arguments for generic compatibility
mask	the mask to impose on x
coordinates	for some of the methods, it is important to specify the names or indices of the columns containing the geographic coordinates (only setMask.data.frame) or else to specify the matrix of spatial coordinates (all setMask methods including it)

Value

The object x appropriately masked (for the setter methods).

Methods (by class)

- default: Set a mask on an object
- data.frame: Set a mask on a data.frame object
- DataFrameStack: Set a mask on a DataFrameStack object
- SpatialGrid: Set a mask on a SpatialGrid object
- GridTopology: Set a mask on a GridTopology object
- SpatialPoints: Set a mask on a SpatialPoints object

See Also

Other masking functions: constructMask(), getMask(), print.mask(), unmask()

`sortDataInGrid` *Reorder data in a grid*

Description

Reorder the data in a compact grid, changing between ordering specifications

Usage

```
sortDataInGrid(
  x,
  grid = attr(x, "grid"),
  orderIn = attr(x, "gridOrder"),
  orderOut = list(refpoint = "bottomleft", cycle = 1:2)
)
```

Arguments

`x`	gridded data
`grid`	grid topology underlying
`orderIn`	current ordering description (see `setGridOrder()`)
`orderOut`	desired output ordering description (see `setGridOrder()`)

Value

the data from `x` (typically a matrix), but reordered as `orderOut`

See Also

`setGridOrder()` for ways of specifying the grid ordering

Examples

```
## Not run:
getTellus(cleanup=TRUE, TI=TRUE)
load("Tellus_TI.RData")
coords = as.matrix(Tellus_TI[,1:2])
compo = compositions::acomp(Tellus_TI[,3:7])
dt = spatialGridAcomp(coords=coords, compo=compo)
image_cokriged(dt, ivar="MgO", breaks = NULL)
x = sort(unique(coords[,1]))
y = sort(unique(coords[,2]))
x0 = c(min(x), min(y))
Ax = c(mean(diff(x)), mean(diff(y)))
n = c(length(x), length(y))
gr = sp::GridTopology(cellcentre.offset=x0, cellsize=Ax, cells.dim=n)
dt0 = sortDataInGrid(Tellus_TI, grid=gr, orderIn=gridOrder_array(2),
                     orderOut=list(refpoint="bottomright", cycle=2:1))
```

```
coords = as.matrix(dt0[,1:2])
compo = compositions::acomp(dt0[,3:7])
spatialGridAcomp(coords=coords, compo=compo)

## End(Not run)
```

`spatialDecorrelation` *Compute diagonalisation measures*

Description

Compute one or more diagonalisation measures out of an empirical multivariate variogram.

Usage

```
spatialDecorrelation(vgemp, ...)

## S3 method for class 'gstatVariogram'
spatialDecorrelation(
  vgemp,
  vgemp0 = NULL,
  method = "add",
  quadratic = method[1] != "rdd",
  ...
)

## S3 method for class 'logratioVariogram'
spatialDecorrelation(vgemp, vgemp0 = NULL, method = "add", ...)

## S3 method for class 'gmEVario'
spatialDecorrelation(vgemp, vgemp0 = NULL, method = "add", ...)
```

Arguments

`vgemp`	the empirical variogram to qualify
`...`	ignored
`vgemp0`	optionally, a reference variogram (see below; necessary for `method="sde"`)
`method`	which quantities are desired? one or more of c("rdd", "add", "sde")
`quadratic`	should the quantities be computed for a variogram or for its square? see below

Details

The three measures provided are

absolute deviation from diagonality ("add") defined as the sum of all off-diagonal elements of the variogram, possibly squared ($p=2$ if `quadratic=TRUE` the default; otherwise $p=1$)

$$\zeta(h) = \sum_{k=1}^{n} \sum_{j \neq k}^{n} \gamma_{k,j}^{p}(h)$$

relative deviation from diagonality ("rdd") comparing the absolute sum of off-diagonal elements with the sum of the diagonal elements of the variogram, each possibly squared ($p=2$ if `quadratic=TRUE`; otherwise $p=1$ the default)

$$\tau(h) = \frac{\sum_{k=1}^{n} \sum_{j \neq k}^{n} |\gamma_{k,j}(h)|^p}{\sum_{k=1}^{n} |\gamma_{k,k}(h)|^p}$$

spatial diagonalisation efficiency ("sde") is the only one requiring `vgemp0`, because it compares an initial state with a diagonalised state of the variogram system

$$\kappa(h) = 1 - \frac{\sum_{k=1}^{n} \sum_{j \neq k}^{n} |\gamma_{k,j}(h)|^p}{\sum_{k=1}^{n} \sum_{j \neq k}^{n} |\gamma_{(0)k,j}(h)|^p}$$

The value of p is controlled by the first value of `method`. That is, the results with `method=c("rdd","add")` are not the same as those obtained with `method=c("add","rdd")`, as in the first case $p=1$ and in the second case $p=2$.

Value

an object of a similar nature to `vgemp`, but where the desired quantities are reported for each lag. This can then be plotted or averages be computed.

Methods (by class)

- `gstatVariogram`: Compute diagonalisation measures
- `logratioVariogram`: Compute diagonalisation measures
- `gmEVario`: Compute diagonalisation measures

Examples

```
data("jura", package="gstat")
X = jura.pred[, 1:2]
Z = jura.pred[,-(1:6)]
gm1 = make.gmCompositionalGaussianSpatialModel(data=Z, coords=X, V="alr")
vg1 = variogram(as.gstat(gm1))
(r1 = spatialDecorrelation(vg1, method=c("add", "rdd")))
plot(r1)
mean(r1)
require("compositions")
pc = princomp(acomp(Z))
v = pc$loadings
colnames(v)=paste("pc", 1:ncol(v), sep="")
gm2 = make.gmCompositionalGaussianSpatialModel(data=Z, coords=X, V=v, prefix="pc")
vg2 = variogram(as.gstat(gm2))
```

```
(r2 = spatialDecorrelation(vg2, method=c("add", "rdd")))
plot(r2)
mean(r2)
(r21 = spatialDecorrelation(vg2, vg1, method="sde") )
plot(r21)
mean(r21)
```

`spatialGridAcomp` *Construct a regionalized composition / reorder compositional simulations*

Description

Connect some coordinates to a composition (of hard data, of predictions or of simulations); currently, the coordinates are stored in an attribute and the dataset is given a complex S3 class. This functionality **will** change in the future, to make use of package "sp" classes.

Usage

```
spatialGridAcomp(coords, compo, dimcomp = 2, dimsim = NA)
```

Arguments

`coords`	coordinates of the locations
`compo`	(observed or predicted) compositional data set; or else array of simulated compositions
`dimcomp`	which of the dimensions of `compo` does correspond to the parts of the composiiton?
`dimsim`	if `compo` contains simulations, which of its dimensions does run across the realisations? leave it as NA if `compo` has observations or predictions.

Value

A (potentially transposed/aperm-ed) matrix of class c("spatialGridAcomp","acomp") with the coordinates in an extra attribute "coords".

See Also

`image_cokriged.spatialGridAcomp()` for an example; `gsi.gstatCokriging2compo()` to restructure the output from `gstat::predict.gstat()` confortably

spatialGridRmult *Construct a regionalized multivariate data*

Description

Connect some coordinates to a multivariate data set (of hard data, of predictions or of simulations); currently, the coordinates are stored in an attribute and the dataset is given a complex S3 class. This functionality **will** change in the future, to make use of package "sp" classes.

Usage

```
spatialGridRmult(coords, data, dimcomp = 2, dimsim = NA)
```

Arguments

coords	coordinates of the locations
data	(observed or predicted) rmult or matrix data set; or else array of simulated rmult /real-valued multivariate data
dimcomp	which of the dimensions of data does correspond to the variables?
dimsim	if data contains simulations, which of its dimensions does run across the realisations? leave it as NA if data has observations or predictions.

Value

A (potentially transposed/aperm-ed) matrix of class c("spatialGridAcomp","acomp") with the coordinates in an extra attribute "coords".

See Also

image_cokriged.spatialGridRmult() for an example; gsi.gstatCokriging2rmult() to restructure the output from gstat::predict.gstat() confortably

spectralcolors *Spectral colors palette based on the RColorBrewer::brewer.pal(11,"Spectral")*

Description

Spectral colors palette based on the RColorBrewer::brewer.pal(11,"Spectral")

Usage

```
spectralcolors(n)
```

Arguments

n	number of colors

Value

a palette, i.e. a list of colors, from dark blue to dark red over pale yellow.

Examples

```
(cls=spectralcolors(20))
```

---

sphTrans *Spherifying transform Compute a transformation that spherifies a certain data set*

Description

Spherifying transform Compute a transformation that spherifies a certain data set

Usage

```
sphTrans(Y, ...)

## Default S3 method:
sphTrans(Y, weights = NULL, p = 1:ncol(Y), ...)
```

Arguments

Y	data set defining the spherification
...	extra arguments for generic functionality
weights	weights to incorporate in the compuations, length=nrow(Y)
p	dimensions to be considered structural (useful for filtering noise)

Value

a function with arguments `(x,inv=FALSE)`, where `x` will be the data to apply the transformation to, and `inv=FALSE` will indicate if the direct or the inverse transformation is desired. This function applied to the same data returns a translated, rotated and scaled, so that the new scores are centered, have variance 1, and no correlation.

Methods (by class)

- `default`: Spherifying transform

Author(s)

K. Gerald van den Boogaart, Raimon Tolosana-Delgado

See Also

ana, anaBackward, sphTrans

Examples

```
library(compositions)
data("jura", package="gstat")
Y = acomp(jura.pred[,c(10,12,13)])
par(mfrow=c(1,1))
plot(Y)
sph = sphTrans(Y)
class(sph)
z = sph(Y)
plot(z)
cor(cbind(z, ilr(Y)))
colMeans(cbind(z, ilr(Y)))
```

`stackDim` *Get/set name/index of (non)stacking dimensions*

Description

Return (or set) the name or index of either the stacking dimension, or else of the non-stacking dimension (typically, the dimension runing through the variables)

Usage

```
stackDim(x, ...)

## S3 method for class 'DataFrameStack'
stackDim(x, ...)

noStackDim(x, ...)

## Default S3 method:
noStackDim(x, ...)

stackDim(x) <- value

## Default S3 replacement method:
stackDim(x) <- value
```

Arguments

`x`	a `DataFrameStack()` object, (only for `stackDim` it can also be a `Spatial` object which `data` slot is a `DataFrameStack`)
`...`	extra arguments for generic functionality
`value`	the name or the index to be considered as stacking dimension

Value

the index or the name of the asked dimension.

Functions

- `stackDim.DataFrameStack`: Get/set name/index of (non)stacking dimensions
- `noStackDim`: Get/set name/index of (non)stacking dimensions
- `noStackDim.default`: Get/set name/index of (non)stacking dimensions
- `stackDim<-`: Get/set name/index of (non)stacking dimensions
- `stackDim<-.default`: Get/set name/index of (non)stacking dimensions

Examples

```
ar = array(1:30, dim = c(5,2,3), dimnames=list(obs=1:5, vars=c("A","B"), rep=1:3))
dfs = DataFrameStack(ar, stackDim="rep")
dfs
stackDim(dfs)
noStackDim(dfs)
getStackElement(dfs, 1)
stackDim(dfs) <- "vars"
getStackElement(dfs, 1)
```

`stackDim,Spatial-method` *Get name/index of the stacking dimension of a Spatial object*

Description

Get name/index of the stacking dimension of a Spatial object which data slot is of class `DataFrameStack()`

Usage

```
## S4 method for signature 'Spatial'
stackDim(x)
```

Arguments

`x`	a `Spatial` object which `data` slot is a `DataFrameStack`

Value

see `stackDim()` for details

See Also

`stackDim()`

swarmPlot *Plot a swarm of calculated output through a DataFrameStack*

Description

Take a DataFrameStack() and apply a certain plotting function to each elements of the stack. The result (typically a curve per each stack element), may then be plotted all together

Usage

```
swarmPlot(
  X,
  MARGIN = stackDim(X),
  PLOTFUN,
  ...,
  .plotargs = list(type = "l"),
  .parallelBackend = NA
)
```

Arguments

X	a DataFrameStack() or anaologous object
MARGIN	which dimension defines the stack? it has a good default! change only if you know what you do
PLOTFUN	the elemental calculating function; this must take as input one element of the stack and return as output the (x,y)-coordinates of the calculated curve for that element, in a list of two elements
...	further parameters to PLOTFUN
.plotargs	either a logical, or else a list of graphical arguments to pass to plot.swarmPlot(); if .plotargs=FALSE no plot is produced; if .plotargs=TRUE a plot with default values is produced;
.parallelBackend	NA or a parallelization strategy; currently unstable for certain operations and platforms.

Value

Invisibly, this function returns a list of the evaluation of PLOTFUN on each element of the stack. If .plotargs other than FALSE, then the function calls plot.swarmPlot() to produce a plot.

Examples

```
dm = list(point=1:100, var=LETTERS[1:2], rep=paste("r",1:5, sep=""))
ar = array(rnorm(1000), dim=c(100,2,5), dimnames = dm)
dfs = DataFrameStack(ar, stackDim="rep")
swarmPlot(dfs, PLOTFUN=function(x) density(x[,1]))
```

swath *Swath plots*

Description

Plots of data vs. one spatial coordinate, with local average spline curve

Usage

```
swath(data, ...)

## Default S3 method:
swath(
  data,
  loc,
  pch = ".",
  withLoess = TRUE,
  commonScale = TRUE,
  xlab = deparse(substitute(loc)),
  ...,
  mfrow
)

## S3 method for class 'acomp'
swath(
  data,
  loc,
  pch = ".",
  withLoess = TRUE,
  commonScale = NA,
  xlab = deparse(substitute(loc)),
  ...
)

## S3 method for class 'ccomp'
swath(
  data,
  loc,
  pch = ".",
  withLoess = TRUE,
  commonScale = NA,
  xlab = deparse(substitute(loc)),
  ...
)

## S3 method for class 'rcomp'
swath(
```

```
  data,
  loc,
  pch = ".",
  withLoess = TRUE,
  commonScale = NA,
  xlab = deparse(substitute(loc)),
  ...
)
```

Arguments

data	data to be represented, compositional class, data.frame, or spatial data object (in which case, loc is a formula!)
...	further arguments to panel plots
loc	a numeric vector with the values for one coordinate
pch	symbol to be used for the individual points, defaults to "."
withLoess	either logical (should a loess line be added?) or a list of arguments to DescTools::lines.loess
commonScale	logical or NA: should all plots share the same vertical range? FALSE=no, TRUE=yes (default), for compositional data sets, the value NA (=all plots within a row) is also permitted and is actually the default
xlab	label for the common x axis (defaults to a deparsed version of loc)
mfrow	distribution of the several plots; it has a good internal default (not used for compositional classes)

Value

Nothing, as the function is primarily called to produce a plot

Methods (by class)

- default: swath plot default method
- acomp: Swath plots for acomp objects
- ccomp: Swath plots for ccomp objects
- rcomp: Swath plots for rcomp objects

Examples

```
data("Windarling")
library("sp")
compo = compositions::acomp(Windarling[,c("Fe","Al2O3","SiO2", "Mn", "P")])
Northing = Windarling$Northing
swath(compo, Northing)
wind.spdf = sp::SpatialPointsDataFrame(sp::SpatialPoints(Windarling[,6:7]),
     data=compo)
swath(wind.spdf, loc=Northing)
```

TurningBands — *Create a parameter set specifying a turning bands simulation algorithm*

Description

Create a parameter set describing a turning bands algorithm to two-point simulation, mostly for covariance or variogram-based gaussian random fields.

Usage

```
TurningBands(nsim = 1, nBands = 1000, ...)
```

Arguments

nsim	number of realisations desired
nBands	number of bands desired for the decomposition of the 2D or 3D space in individual signals
...	further parameters, currently ignored

Value

an S3-list of class "gmTurningBands" containing the few elements given as arguments to the function. This is just a compact way to provide further functions such as `predict.gmSpatialModel()` with appropriate triggers for choosing a prediction method or another, in this case for triggering turning bands simulation.

Examples

```
(tbs_local = TurningBands(nsim=100, nBands=300))
## then run predict(..., pars=tbs_local)
```

unmask — *Unmask a masked object*

Description

Unmask a masked object, i.e. recover the original grid and extend potential data containers associated to it with NAs. See examples in `constructMask()`

Usage

```
unmask(x, ...)

## S3 method for class 'data.frame'
unmask(
  x,
  mask = attr(x, "mask"),
  fullgrid = attr(mask, "fullgrid"),
  forceCheck = is(fullgrid, "GridTopology"),
  ...
)

## S3 method for class 'DataFrameStack'
unmask(
  x,
  mask = attr(x, "mask"),
  fullgrid = attr(mask, "fullgrid"),
  forceCheck = is(fullgrid, "GridTopology"),
  ...
)

## S3 method for class 'SpatialPixels'
unmask(
  x,
  mask = NULL,
  fullgrid = attr(mask, "fullgrid"),
  forceCheck = FALSE,
  ...
)

## S3 method for class 'SpatialPoints'
unmask(
  x,
  mask = attr(x@data, "mask"),
  fullgrid = attr(mask, "fullgrid"),
  forceCheck = FALSE,
  ...
)
```

Arguments

x	a masked object
...	arguments for generic functionality
mask	the mask; typically has good defaults
fullgrid	the full grid; typically has good defaults
forceCheck	if fullgrid is provided, should the coordinates provided in x and in fullgrid be cross-checked to ensure that they are given in compatible orders? See sortDataInGrid()

and setGridOrder() for controlling the ordering of vectors and grids.

Value

The original grid data and extend potential data containers associated to it with NAs. See examples in constructMask(). The nature of the output depends on the nature of x: a "data.frame" produced a "data.frame"; a "unmask.DataFrameStack" produces a "unmask.DataFrameStack"; a "SpatialPoints" produces a "SpatialPoints"; and finally a "SpatialPixels" produces either a "SpatialPixels" or a "SpatialGrid" (if it is full). Note that only in the case that class(x)=="SpatialPixels" is mask required, for the other methods all arguments have reasonable defaults.

Functions

- unmask: Unmask a masked object
- unmask.DataFrameStack: Unmask a masked object
- unmask.SpatialPixels: Unmask a masked object
- unmask.SpatialPoints: Unmask a masked object

See Also

Other masking functions: constructMask(), getMask(), print.mask(), setMask()

---

validate *Validate a spatial model*

Description

Validate a spatial model by predicting some values. Typically this will be a validation set, or else some subset of the conditing data.

Usage

```
validate(object, strategy, ...)

## S3 method for class 'LeaveOneOut'
validate(object, strategy, ...)

## S3 method for class 'NfoldCrossValidation'
validate(object, strategy, ...)
```

Arguments

object	spatial model object, typically of class gstat::gstat() or gmSpatialModel
strategy	which strategy to follow for the validation? see functions in 'see also' below.
...	generic parameters, ignored.

Value

A data frame of predictions (possibly with kriging variances and covariances, or equivalent uncertainty measures) for each element of the validation set

Methods (by class)

- `LeaveOneOut`: Validate a spatial model
- `NfoldCrossValidation`: Validate a spatial model

See Also

Other validation functions: `LeaveOneOut`, `NfoldCrossValidation`

Other accuracy functions: `accuracy()`, `mean.accuracy()`, `plot.accuracy()`, `precision()`, `xvErrorMeasures()`

Examples

```
data("Windarling")
X = Windarling[,c("Easting","Northing")]
Z = compositions::acomp(Windarling[,c(9:12,16)])
gm = make.gmCompositionalGaussianSpatialModel(data=Z, coords=X)
vg = variogram(gm)
md = gstat::vgm(range=30, model="Sph", nugget=1, psill=1)
gs = fit_lmc(v=vg, g=gm, model=md)
## Not run:  v1 = validate(gs, strategy=LeaveOneOut()) # quite slow
vs2 = NfoldCrossValidation(nfolds=sample(1:10, nrow(X), replace=TRUE))
vs2
## Not run:  v2 = validate(gs, strategy=vs2) # quite slow
```

`variogramModelPlot` *Quick plotting of empirical and theoretical variograms Quick and dirty plotting of empirical variograms/covariances with or without their models*

Description

Quick plotting of empirical and theoretical variograms Quick and dirty plotting of empirical variograms/covariances with or without their models

Usage

```
variogramModelPlot(vg, ...)

## S3 method for class 'gmEVario'
variogramModelPlot(
  vg,
  model = NULL,
  col = rev(rainbow(ndirections(vg))),
```

```
  commonAxis = FALSE,
  newfig = TRUE,
  closeplot = TRUE,
  ...
)
```

Arguments

vg	empirical variogram or covariance function
...	further parameters to underlying plot or matplot functions
model	optional, theoretical variogram or covariance function
col	colors to use for the several directional variograms
commonAxis	boolean, should all plots in a row share the same vertical axis?
newfig	boolean, should a new figure be created? otherwise user should ensure the device space is appropriately managed
closeplot	logical, should the plot be left open (FALSE) for further changes, or be frozen (TRUE)? defaults to TRUE

Value

The function is primarily called for producing a plot. However, it invisibly returns the graphical parameters active before the call occurred. This is useful for constructing complex diagrams, by adding layers of info. If you want to "freeze" your plot, embed your call in another call to par, e.g. `par(variogramModelPlot(...))`; if you want to leave the plot open for further changes give the extra argument `closeplot=FALSE`.

Functions

- `variogramModelPlot`: Quick plotting of empirical and theoretical variograms

See Also

logratioVariogram()

Other variogramModelPlot: variogramModelPlot.gstatVariogram(), variogramModelPlot.logratioVariogram()

Other gmEVario functions: as.gmEVario(), gsi.EVario2D(), ndirections(), plot.gmEVario()

Other gmCgram functions: [.gmCgram(), [[.gmCgram(), as.function.gmCgram(), as.gmCgram.variogramModelList(), length.gmCgram(), ndirections(), plot.gmCgram()

Examples

```
utils::data("variogramModels")
v1 = setCgram(type=vg.Gau, sill=diag(3)+0.5, anisRanges = 5e-1*diag(c(3,0.5)))
v2 = setCgram(type=vg.Exp, sill=0.3*diag(3), anisRanges = 5e-2*diag(2))
vm = v1+v2
plot(vm, closeplot=TRUE)
library(gstat)
data("jura", package = "gstat")
```

```
X = as.matrix(jura.pred[,1:2])
Z = as.matrix(jura.pred[,c("Zn","Cd","Pb")])
vge = gsi.EVario2D(X,Z)
variogramModelPlot(vge, vm)
```

`variogramModelPlot.gstatVariogram` *Quick plotting of empirical and theoretical variograms Quick and dirty plotting of empirical variograms/covariances with or without their models*

Description

Quick plotting of empirical and theoretical variograms Quick and dirty plotting of empirical variograms/covariances with or without their models

Usage

```
## S3 method for class 'gstatVariogram'
variogramModelPlot(
  vg,
  model = NULL,
  col = rev(rainbow(1 + length(unique(vg$dir.hor)))),
  commonAxis = FALSE,
  newfig = TRUE,
  closeplot = TRUE,
  ...
)
```

Arguments

`vg`	empirical variogram or covariance function
`model`	optional, theoretical variogram or covariance function
`col`	colors to use for the several directional variograms
`commonAxis`	boolean, should all plots in a row share the same vertical axis?
`newfig`	boolean, should a new figure be created? otherwise user should ensure the device space is appropriately managed
`closeplot`	logical, should the plot be left open (FALSE) for further changes, or be frozen (TRUE)? defaults to TRUE
`...`	further parameters to underlying plot or matplot functions

Value

The function is primarily called for producing a plot. However, it invisibly returns the graphical parameters active before the call occurred. This is useful for constructing complex diagrams, by giving argument `closeplot=FALSE` and then adding layers of information. If you want to "freeze" your plot, either give `closeplot=TRUE` or embed your call in another call to `par`, e.g. `par(variogramModelPlot(...))`.

See Also

`gstat::plot.gstatVariogram()`

Other variogramModelPlot: `variogramModelPlot.logratioVariogram()`, `variogramModelPlot()`

Examples

```
data("jura", package="gstat")
X = jura.pred[,1:2]
Zc = jura.pred[,7:13]
gg = make.gmCompositionalGaussianSpatialModel(Zc, X, V="alr", formula = ~1)
vg = variogram(gg)
md = gstat::vgm(model="Sph", psill=1, nugget=1, range=1.5)
gg = fit_lmc(v=vg, g=gg, model=md)
variogramModelPlot(vg, model=gg)
```

`variogramModelPlot.logratioVariogram`

Quick plotting of empirical and theoretical logratio variograms Quick and dirty plotting of empirical logratio variograms with or without their models

---

Description

Quick plotting of empirical and theoretical logratio variograms Quick and dirty plotting of empirical logratio variograms with or without their models

Usage

```
## S3 method for class 'logratioVariogram'
variogramModelPlot(vg, model = NULL, col = rev(rainbow(ndirections(vg))), ...)
```

Arguments

`vg`	empirical variogram or covariance function
`model`	optional, theoretical variogram or covariance function
`col`	colors to use for the several directional variograms
`...`	further parameters to `plot.logratioVariogramAnisotropy()`

Value

The function is primarily called for producing a plot. However, it invisibly returns the graphical parameters active before the call occurred. This is useful for constructing complex diagrams, by adding layers of info. If you want to "freeze" your plot, embed your call in another call to `par`, e.g. `par(variogramModelPlot(...))`.

See Also

Other variogramModelPlot: `variogramModelPlot.gstatVariogram()`, `variogramModelPlot()`

`Windarling` *Ore composition of a bench at a mine in Windarling, West Australia.*

Description

A dataset containing the geochemical composition (and some extra variables) of a grade control study of a mine bench at Windarling, West Australia.

Usage

```
Windarling
```

Format

A data.frame with 1600 rows and 16 columns:

Hole_id ID of the observation

Sample.West indicator for belonging to a subsample on the West sector

Sample.East indicator for belonging to a subsample on the East sector

West indicator for belonging to the Western wing of the bench

East indicator for belonging to the Eastern wing of the bench

Easting Easting (X) coordinate of the sample

Northing Northing (Y) coordinate of the sample

Lithotype factor, indicating material type of the sample, one of: basalt, goethite, magnetite, schist

Fe percent of Iron in the sample

P percent of Phosphorus in the sample

SiO2 percent of Silica in the sample

Al2O3 percent of Alumina in the sample

S percent of Sulphur in the sample

Mn percent of Magnanese in the sample

CL percent of Chlorine in the sample

LOI percent Loss-on-Ignition of the sample

License

CC BY-SA 4.0

Source

Ward (2015) Compositions, logratios and geostatistics: An application to iron ore. MSc Thesis Edith Cowan University; https://ro.ecu.edu.au/theses/1581/

`write.GSLib` *Write a regionalized data set in GSLIB format*

Description

Write a regionalized data set in plain text GSLIB format

Usage

```
write.GSLib(x, file, header = basename(file))
```

Arguments

`x`	regionalized data set
`file`	filename
`header`	the first line of text for the file, defaults to filename

Value

The status of closing the file, see `close` for details, although this is seldom problematic. This function is basically called for its side-effect of writing a data set in the simplified Geo-EAS format that is used in GSLIB.

See Also

http://www.gslib.com/gslib_help/format.html

Examples

```
data("jura", package="gstat")
## Not run: write.GSLib(jura.pred, file="jurapred.txt")
```

`xvErrorMeasures` *Cross-validation errror measures*

Description

Compute one or more error measures from cross-validation output

Usage

```
xvErrorMeasures(x, ...)

## S3 method for class 'data.frame'
xvErrorMeasures(
  x,
  observed = x$observed,
  output = "MSDR1",
  univariate = length(dim(observed)) == 0,
  ...
)

## S3 method for class 'DataFrameStack'
xvErrorMeasures(
  x,
  observed,
  output = "ME",
  univariate = length(dim(observed)) == 0,
  ...
)
```

Arguments

`x`	a dataset of predictions (if `x` is of class "data.frame") or simulations (if `x` is of class "DataFrameStack")
`...`	extra arguments for generic functionality
`observed`	a vector (if univariate) or a matrix/dataset of true values
`output`	which output do you want? a vector of one or several of c("ME","MSE","MSDR","MSDR1","MSDR2","M
`univariate`	logical control, typically you should not touch it

Details

"ME" stands for *mean error* (average of the differences between true values and predicted values), "MSE" stands for *mean square error* (average of the square differences between true values and predicted values), and "MSDR" for *mean squared deviation ratio* (average of the square between true values and predicted values each normalized by its kriging variance). These quantities are classically used in evaluating output results of validation excercices of one single variable. For multivariate cases, "ME" (a vector) and "MSE" (a scalar) work as well, while two different definitions of a multivariate mean squared deviation ratio can be given:

- "MSDR1" is the average Mahalanobis square error (see `accuracy()` for explanations)
- "MSDR2" is the average univariate "MSDR" over all variables.

Value

If just some of c("ME","MSE","MSDR","MSDR1","MSDR2") are requested, the output is a named vector with the desired quantities. If only "Mahalanobis" is requested, the output is a vector of Mahalanobis square errors. If you mix up things and ask for "Mahalanobis" and some of the quantities mentioned above, the result will be a named list with the requested quantities. (NOTE: some options are not available for `x` a "DataFrameStack")

Functions

- `xvErrorMeasures`: Cross-validation errror measures
- `xvErrorMeasures.DataFrameStack`: Cross-validation errror measures

See Also

Other accuracy functions: `accuracy()`, `mean.accuracy()`, `plot.accuracy()`, `precision()`, `validate()`

`[.DataFrameStack` *Extract rows of a DataFrameStack*

Description

Extract rows (and columns if you know what you are doing) from a stacked data frame

Usage

```
## S3 method for class 'DataFrameStack'
x[i = NULL, j = NULL, ..., drop = FALSE]
```

Arguments

`x`	`DataFrameStack()` object
`i`	row indices, names or logical vector of appropriate length (i.e. typically locations, observations, etc)
`j`	column indices, names or logical vector of appropriate length. DO NOT USE if you are not sure what you are doing. The result will be a conventional data.frame, probably with the stacking structure destroyed.
`...`	generic parameters, ignored.
`drop`	logical, if selection results in one single row or column selected, return a vector?

Value

the DataFrameStack or data.frame of the selected rows and columns.

Examples

```
ar = array(1:30, dim = c(5,2,3), dimnames=list(obs=1:5, vars=c("A","B"), rep=1:3))
dfs = DataFrameStack(ar, stackDim="rep")
dfs[1:2,]
stackDim(dfs[1:2,])
```

`[.gmCgram` *Subsetting of gmCgram variogram structures*

Description

Extraction of some variables of a gmCgram object

Usage

```
## S3 method for class 'gmCgram'
x[i, j = i, ...]
```

Arguments

`x`	`gmCgram` variogram object
`i`	row-indices of the variables to be kept/removed
`j`	column-indices of the variables to be kept/removed (if only `i` is specified, `j` will be taken as equal to `i`!)
`...`	extra arguments for generic functionality

Details

This function can be used to extract the model for a a subset of variables. If only `i` is specified, `j` will be taken as equal to `i`. If you want to select all `i`'s for certain `j`'s or vice versa, give `i=1:dim(x$nugget)[1]` and `j=` your desired indices, respectively `j=1:dim(x$nugget)[2]` and `i=` your desired indices; replace `x` by the object you are giving. If `i!=j`, the output will be a `c("gmXCgram","gmCgram")` object, otherwise it will be a regular class `"gmCgram"` object. If you want to extract "slots" or "elements" of the variogram, use the $-notation. If you want to extract variables of the variogram matrices, use the [-notation.

Value

a `gmCgram` variogram object with the desired variables only.

See Also

Other gmCgram functions: `[[.gmCgram()`, `as.function.gmCgram()`, `as.gmCgram.variogramModelList()`, `length.gmCgram()`, `ndirections()`, `plot.gmCgram()`, `variogramModelPlot()`

Examples

```
utils::data("variogramModels")
v1 = setCgram(type=vg.Gau, sill=diag(2), anisRanges = 3*diag(c(3,1)))
v2 = setCgram(type=vg.Exp, sill=0.3*diag(2), anisRanges = 0.5*diag(2))
vm = v1+v2
vm[1,1]
```

[.logratioVariogramAnisotropy

Subsetting of logratioVariogram objects

Description

Subsetting of logratioVariogram objects

Usage

```
## S3 method for class 'logratioVariogramAnisotropy'
x[i = NULL, j = NULL, ...]
```

Arguments

x	an object of class "logratioVariogram" or c("logratioVariogramAnisotropy", "logratioVariogram")
i	index or indexes of lags to be kept (if positive) or removed (if negative)
j	index or indexes of directions to be kept, only for objects of class c("logratioVariogramAnisotropy", "logratioVariogram")
...	extra arguments, ignored

Value

the selected variograms or lags, potentially of class "logratioVariogram" if only one direction is chosen

Examples

```
data("jura", package="gstat")
X = jura.pred[,1:2]
Zc = compositions::acomp(jura.pred[,7:9])
vg = logratioVariogram(data=Zc, loc=X)
vg[1]
vg = logratioVariogram(data=Zc, loc=X, azimuth=c(0,90))
vg[,1]
vg[1,1]
vg[1,]
```

`[[.gmCgram` *Subsetting of gmCgram variogram structures*

Description

Extraction or combination of nested structures of a gmCgram object

Usage

```
## S3 method for class 'gmCgram'
x[[i, ...]]
```

Arguments

`x`	`gmCgram` variogram object
`i`	indices of the structures that are desired to be kept (0=nugget) or removed (see details)
`...`	extra arguments for generic functionality

Details

This function can be used to: extract the nugget (i=0), extract some structures (i=indices of the structures, possibly including 0 for the nugget), or filter some structures out (i=negative indices of the structures to remove; nugget will always removed in this case). If you want to extract "slots" or "elements" of the variogram, use the $-notation. If you want to extract variables of the variogram matrices, use the `[`-notation. The contrary operation (adding structures together) is obtained by summing (+) two `gmCgram` objects.

Value

a `gmCgram` variogram object with the desired structures only.

See Also

Other gmCgram functions: `[.gmCgram()`, `as.function.gmCgram()`, `as.gmCgram.variogramModelList()`, `length.gmCgram()`, `ndirections()`, `plot.gmCgram()`, `variogramModelPlot()`

Examples

```
utils::data("variogramModels")
v1 = setCgram(type=vg.Gau, sill=diag(2), anisRanges = 3*diag(c(3,1)))
v2 = setCgram(type=vg.Exp, sill=0.3*diag(2), anisRanges = 0.5*diag(2))
vm = v1+v2
vm[[1]]
```

Index