Package: geoRcb (via r-universe)

October 27, 2024

Title An Extension of Package geoR that Works with Cost-Based Distances

Version 1.7-7

Date 2015-08-07

Encoding UTF-8

Description Mainly, three functions are adapted in order to admit custom distance matrices as additional arguments. Namely variog, to produce empirical variograms; likfit, to fit theoretical variograms and krige.conv, to preform kriging.

Depends R (>= 3.0.0), geoR (>= 1.7.4.1)

Imports gdistance

Suggests sp, raster, rgdal, testthat, knitr, rmarkdown

License GPL (>= 3) | file LICENSE

URL `https://github.com/famuvie/geoRcb`

BugReports `https://github.com/famuvie/geoRcb/issues`

LazyData true

VignetteBuilder knitr

Repository https://famuvie.r-universe.dev

RemoteUrl https://github.com/famuvie/geoRcb

RemoteRef HEAD

RemoteSha 6b3591efbf208ee7ac8ed2c79dd62e3ef75ce8af

Contents

---

distmatGen *Compute cost-based distances*

Description

Generate a cost-based distance matrix among observations and/or between observations and prediction locations.

Usage

```
distmatGen(pts, condsurf, ret = c("both", "obs", "loc"), directions = 16,
  silent = FALSE)
```

Arguments

pts	A SpatialPoints[DataFrame] with a defined projection or a two-column data.frame/matrix of coordinates
condsurf	raster. Conductivity surface.
ret	specify whether to return distances between obs-obs ("obs"), obs-loc ("loc"), or both ("both") (default)
directions	See transition.
silent	logical. If TRUE avoids any warnings or messages.

Details

Use the centroids of the conductivity or conductivity surface raster cells as prediction locations.

Examples

```
data(noise)
if (require(raster)) {
  r <- raster(extent(c(0,3,0,3)), nrows = 3, ncols = 3)
  wall.idx <- 4:5
  values(r)[-wall.idx] <- 1
  obs <- coordinates(r)[-wall.idx, ]

  ddm <- distmatGen(obs, r, ret = "obs", directions = 8)

  par(mfrow = c(2, 1))
  plot(r)
  text(obs[, 1], obs[, 2],
       col = c('red', rep('black', 6)))

  plot(dist(obs)[1:6], ddm[-1, 1],
       type = 'n',
       main = 'Distances to point 1',
       xlab = 'Euclidean distance',
       ylab = 'Cost-based distance')
  text(dist(obs)[1:6], ddm[-1, 1],
       labels = 2:7)
  abline(0, 1)
}
```

`geoR_trf` *Default transition function in geoR*

Description

Default function to use with `transition`. Defined as the geometric mean.

Usage

```
geoR_trf(x)
```

Arguments

`x`	numeric vector of length 2 with cost/conductivity non-negative numbers.

Details

To use another function, override this one by defining it using the same name.

Value

cost/conductivity value of the transition between neighbouring cells.

Examples

```
geoR_trf(c(0, 1))

   # redefine function
   geoR_trf <- mean

   geoR_trf(c(0, 1))
```

`idx_isol` *Identify isolated locations*

Description

Defined as points whith cost distances to every other point of either 0 or Infinite.

Usage

```
idx_isol(x)
```

Details

param x distance matrix.

`krige.conv` *cost-based kriging*

Description

All the arguments work as in `krige.conv`, except the additional arguments `dd.dists.mat` and `dl.dists.mat`, which take matrices of distances between observation locations and between observations and prediction locations respectively

Usage

```
krige.conv(geodata, coords = geodata$coords, data = geodata$data, locations,
  borders, krige, output, dd.dists.mat, dl.dists.mat)
```

Arguments

`geodata`	a list containing elements `coords` and `data` as described next. Typically an object of the `class` `"geodata"` - a **geoR** data-set. If not provided the arguments `coords` and `data` must be provided instead.
`coords`	an $n \times 2$ matrix or data-frame with the 2-D coordinates of the n data locations. By default it takes the component `coords` of the argument `geodata`, if provided.
`data`	a vector with *n* data values. By default it takes the component `data` of the argument `geodata`, if provided.
`locations`	an $N \times 2$ matrix or data-frame with the 2-D coordinates of the N prediction locations, or a list for which the first two components are used. Input is internally checked by the function `check.locations`.
`borders`	optional. By default reads the element `borders` from the `geodata` object, if present. Setting to `NULL` prevents this behavior. If a two column matrix defining a polygon is provided the prediction is performed only at locations inside this polygon.

`krige`	a list defining the model components and the type of kriging. It can take an output to a call to `krige.control` or a list with elements as for the arguments in `krige.control`. Default values are assumed for arguments or list elements not provided. See the description of arguments in 'krige.control' below.
`output`	a list specifying output options. It can take an output to a call to `output.control` or a list with elements as for the arguments in `output.control`. Default values are assumed for arguments not provided. See documentation for `output.control` for further details.
`dd.dists.mat`	n x n symmetric matrix with cost-based distances between observations
`dl.dists.mat`	m x n matrix with cost-based distances from each observation to each one of the m prediction locations

Examples

```
## geodata structure with transformed covariates
data(noise)
if (require(sp)) {
  covarnames=sapply(1:3, function(x) paste("d2TV", x, sep=""))
  obs.df <- data.frame(Leq=obs$Leq,
                       1/(1+(as.data.frame(obs)[covarnames]/20)^2))
  obs.gd <- as.geodata(cbind(coordinates(obs), obs.df),
                       data.col="Leq",
                     covar.col=c('d2TV1','d2TV2','d2TV3'))
  trend=~d2TV1*(d2TV2+d2TV3)

  loc1.df <- as.data.frame(1/(1+(as.data.frame(loc)[covarnames]/20)^2))

  ## fitting variogram models
  vgmdl.std  <- likfit(geodata = obs.gd, trend=trend,
                     ini = c(8,300), cov.model = "matern")
  vgmdl.dmat <- likfit(geodata = obs.gd, trend=trend,
                       ini = c(8,300), cov.model = "matern",
                       dists.mat=dd.distmat)


  # With trend, Euclidean distances
  # NOTE: The Euclidean prediction is done with cost-based covariates
  KC.std = krige.control(trend.d=trend,
                         trend.l=~loc1.df$d2TV1*(loc1.df$d2TV2+loc1.df$d2TV3),
                         obj.model=vgmdl.std)
  kc1.std<-krige.conv(obs.gd,locations=coordinates(loc), krige=KC.std)

  # With trend, Cost-based distances
  KC = krige.control(trend.d=trend,
                     trend.l=~loc1.df$d2TV1*(loc1.df$d2TV2+loc1.df$d2TV3),
                     obj.model=vgmdl.dmat)
  kc1<-krige.conv(obs.gd,locations=coordinates(loc), krige=KC,
                  dd.dists.mat=dd.distmat, dl.dists.mat=dl.distmat)
}
```

likfit *Fit a cost-based variogram model*

Description

All the arguments work as in likfit, except the additional argument dists.mat, which takes a symmetric matrix of distances between observation locations

Usage

```
likfit(geodata, coords = geodata$coords, data = geodata$data,
  trend = "cte", ini.cov.pars, fix.nugget = FALSE, nugget = 0,
  fix.kappa = TRUE, kappa = 0.5, fix.lambda = TRUE, lambda = 1,
  fix.psiA = TRUE, psiA = 0, fix.psiR = TRUE, psiR = 1, cov.model,
  realisations, lik.method = "ML", components = TRUE, nospatial = TRUE,
  limits = pars.limits(), dists.mat, print.pars = FALSE, messages, ...)
```

Arguments

geodata	a list containing elements coords and data as described next. Typically an object of the class "geodata". If not provided the arguments coords and data must be provided instead.
coords	an $n \times 2$ matrix where each row has the 2-D coordinates of the n data locations. By default it takes the component coords of the argument geodata, if provided.
data	a vector with n data values. By default it takes the component data of the argument geodata, if provided.
trend	specifies the mean part of the model. See documentation of trend.spatial for further details. Defaults to "cte".
ini.cov.pars	initial values for the covariance parameters: σ^2 (partial sill) and ϕ (range parameter). Typically a vector with two components. However a matrix can be used to provide several initial values. See DETAILS below.
fix.nugget	logical, indicating whether the parameter τ^2 (nugget variance) should be regarded as fixed (fix.nugget = TRUE) or should be estimated (fix.nugget = FALSE). Defaults to FALSE.
nugget	value of the nugget parameter. Regarded as a fixed value if fix.nugget = TRUE otherwise as the initial value for the minimisation algorithm. Defaults to zero.
fix.kappa	logical, indicating whether the extra parameter κ should be regarded as fixed (fix.kappa = TRUE) or should be estimated (fix.kappa = FALSE). Defaults to TRUE.
kappa	value of the extra parameter κ. Regarded as a fixed value if fix.kappa = TRUE otherwise as the initial value for the minimisation algorithm. Defaults to 0.5. This parameter is valid only if the covariance function is one of: "matern", "powered.exponential", "cauchy" or "gneiting.matern". For more details on covariance functions see documentation for cov.spatial.

`fix.lambda`	logical, indicating whether the Box-Cox transformation parameter λ should be regarded as fixed (`fix.lambda = TRUE`) or should be be estimated (`fix.lambda = FALSE`). Defaults to `TRUE`.
`lambda`	value of the Box-Cox transformation parameter λ. Regarded as a fixed value if `fix.lambda = TRUE` otherwise as the initial value for the minimisation algorithm. Defaults to 1. Two particular cases are $\lambda = 1$ indicating no transformation and $\lambda = 0$ indicating log-transformation.
`fix.psiA`	logical, indicating whether the anisotropy angle parameter ψ_R should be regarded as fixed (`fix.psiA = TRUE`) or should be estimated (`fix.psiA = FALSE`). Defaults to `TRUE`.
`psiA`	value (in radians) for the anisotropy angle parameter ψ_A. Regarded as a fixed value if `fix.psiA = TRUE` otherwise as the initial value for the minimisation algorithm. Defaults to 0. See `coords.aniso` for further details on anisotropy correction.
`fix.psiR`	logical, indicating whether the anisotropy ratio parameter ψ_R should be regarded as fixed (`fix.psiR = TRUE`) or should be estimated (`fix.psiR = FALSE`). Defaults to `TRUE`.
`psiR`	value, always greater than 1, for the anisotropy ratio parameter ψ_R. Regarded as a fixed value if `fix.psiR = TRUE` otherwise as the initial value for the minimisation algorithm. Defaults to 1. See `coords.aniso` for further details on anisotropy correction.
`cov.model`	a string specifying the model for the correlation function. For further details see documentation for `cov.spatial`. Reads values from an `variomodel` object passed to `ini.cov.pars` if any, otherwise defaults to the *exponential* model.
`realisations`	optional. Logical or a vector indicating the number of replication for each datum. For further information see `DETAILS` below and documentation for `as.geodata`.
`lik.method`	(formely method.lik) options are `"ML"` for maximum likelihood and `"REML"` for restricted maximum likelihood. Defaults to `"ML"`.
`components`	an $n \times 3$ data-frame with fitted values for the three model components: trend, spatial and residuals. See the section `DETAILS` below for the model specification.
`nospatial`	logical. If `TRUE` parameter estimates for the model without spatial component are included in the output.
`limits`	values defining lower and upper limits for the model parameters used in the numerical minimisation. The auxiliary function `pars.limits` is called to set the limits. See also **Limits** in DETAILS below.
`dists.mat`	n x n symmetric matrix with cost-based distances between observations
`print.pars`	logical. If `TRUE` the parameters and the value of the negative log-likelihood (up to a constant) are printed each time the function to be minimised is called.
`messages`	logical. Indicates whether status messages should be printed on the screen (or output device) while the function is running.
`...`	additional parameters to be passed to the minimisation function. Typically arguments of the type `control()` which controls the behavior of the minimisation algorithm. For further details see documentation for the minimisation function `optim`.

Examples

```
## geodata structure with transformed covariates
data(noise)
if (require(sp)) {
  covarnames=sapply(1:3, function(x) paste("d2TV", x, sep=""))
  obs.df <- data.frame(Leq=obs$Leq,
                       1/(1+(as.data.frame(obs)[covarnames]/20)^2))
  obs.gd <- as.geodata(cbind(coordinates(obs), obs.df),
                       data.col="Leq",
                       covar.col=c('d2TV1','d2TV2','d2TV3'))
  trend=~d2TV1*(d2TV2+d2TV3)

  ## compute euclidean and cost-based empirical variograms
  vg.std <- variog(obs.gd, trend=trend)
  vg.dmat <- variog(obs.gd, trend=trend, dists.mat=dd.distmat)

  ## fitting variogram models
  vgmdl.std  <- likfit(geodata = obs.gd, trend=trend,
                       ini = c(8,300), cov.model = "matern")
  vgmdl.dmat <- likfit(geodata = obs.gd, trend=trend,
                       ini = c(8,300), cov.model = "matern",
                       dists.mat=dd.distmat)

  ## Fitted parameters
  data.frame(
    parameters=c("tausq","sigmasq","phi"),
   Euclidean=c(round(vgmdl.std$tausq,2),round(vgmdl.std$sigmasq,2),round(vgmdl.std$phi,0)),
   Cost_based=c(round(vgmdl.dmat$tausq,2),round(vgmdl.dmat$sigmasq,2),round(vgmdl.dmat$phi,0)))

  ## practical range
  ## defined as the value for which the correlation function
  ## decays to 5% of its value at 0
  x=seq(0,800)
  y=cov.spatial(x,cov.pars=vgmdl.std$cov.pars)
  min(x[y<0.05*y[1]])    # 358
  y=cov.spatial(x,cov.pars=vgmdl.dmat$cov.pars)
  min(x[y<0.05*y[1]])    # 502
  # Note that the cost-based  analysis detects a
  # longer-ranged correlation structure

  ## plotting and comparing empirical variograms
  ## (with classical and robust estimation)
  ## and fitted variogram models
  op <- par(mfrow=c(2,2))
  u = 13  # binning
  for( est in c('classical','modulus') )
  {
    vg.std <- variog(obs.gd, trend=trend, estimator.type=est, uvec=u)
   vg.dmat <- variog(obs.gd, trend=trend, dists.mat=dd.distmat, estimator.type=est, uvec=u)
    plot(vg.std,pch=20, cex=1.2, max.dist=max(vg.dmat$u), ylim=c(0,20), col='gray')
    lines(vg.std,pch=20, col='gray')
    lines(vg.dmat,pch=20,cex=2,col='red')
    legend('topleft',c('Euclidean','Cost'),lty=1,lwd=2,col=c('gray','red'))
    title(paste('binning:',u,'   estimator:',est))

    plot(vg.std, pch=20, cex=1.2, max.dist=800, col='gray')   # empirical standard
    lines(vgmdl.std,lwd=2,col='gray')        # fitted model standard
   points(as.data.frame(vg.dmat[c(1,2)]),pch=20,cex=2,col='red')  # empirical cost-based
    lines(vgmdl.dmat,lwd=2,col='darkred')            # fitted model cost-based
    legend('topleft',c('Euclidean','Cost'),lty=1,lwd=2,col=c('gray','red'))
    title(paste('binning:',u,'   estimator:',est))
  }
  par(op)
}
```

`loglik.GRF` *Log-likelihood*

Description

Same as geoR's `loglik.GRF` but taking into account non-Euclidean distances if pertinent.

Usage

```
loglik.GRF(geodata, coords = geodata$coords, data = geodata$data,
  obj.model = NULL, cov.model = "exp", cov.pars, nugget = 0,
  kappa = 0.5, lambda = 1, psiR = 1, psiA = 0, trend = "cte",
  method.lik = "ML", compute.dists = TRUE, realisations = NULL)
```

Arguments

`geodata`	a list containing elements `coords` and `data` as described next. Typically an object of the class `"geodata"` - a **geoR** data-set. If not provided the arguments `coords` and `data` must be provided instead.
`coords`	an $n \times 2$ matrix, each row containing Euclidean coordinates of the n data locations. By default it takes the element `coords` of the argument `geodata`.
`data`	a vector with data values. By default it takes the element `data` of the argument `geodata`.
`obj.model`	a object of the class `variomodel` with a fitted model. Tipically an output of `likfit` or `variofit`.
`cov.model`	a string specifying the model for the correlation function. For further details see documentation for `cov.spatial`.
`cov.pars`	a vector with 2 elements with values of the covariance parameters σ^2 (partial sill) and ϕ (range parameter).
`nugget`	value of the nugget parameter. Defaults to 0.
`kappa`	value of the smoothness parameter. Defaults to 0.5.
`lambda`	value of the Box-Cox tranformation parameter. Defaults to 1.

`psiR`	value of the anisotropy ratio parameter. Defaults to 1, corresponding to isotropy.
`psiA`	value (in radians) of the anisotropy rotation parameter. Defaults to zero.
`trend`	specifies the mean part of the model. The options are: `"cte"` (constant mean), `"1st"` (a first order polynomial on the coordinates), `"2nd"` (a second order polynomial on the coordinates), or a formula of the type `~X` where `X` is a matrix with the covariates (external trend). Defaults to `"cte"`.
`method.lik`	options are `"ML"` for likelihood and `"REML"` for restricted likelihood. Defaults to `"ML"`.
`compute.dists`	for internal use with other function. Don't change the default unless you know what you are doing.
`realisations`	optional. A vector indicating replication number for each data. For more details see `as.geodata`.

See Also

`loglik.GRF`

`noise` *Noise data*

Description

Noise measurements, cartography and cost-based distance matrices from a pilot-study in Valencia, Spain (see reference).

References

Antonio López-Quílez and Facundo Muñoz (2009). Geostatistical computing of acoustic maps in the presence of barriers. *Mathematical and Computer Modelling* **50**(5-6): 929–938. DOI:10.1016/j.mcm.2009.05.021

Examples

```
data(noise)
 if (require(sp)) {
   plot(malilla)
   points(obs, pch=19, col='red')

   ## geodata structure with transformed covariates
   covarnames=sapply(1:3, function(x) paste("d2TV", x, sep=""))
   obs1.df <- as.data.frame(cbind(Leq=obs$Leq,
                         1/(1+(as.data.frame(obs)[covarnames]/20)^2)))
   obs.gd <- as.geodata(cbind(coordinates(obs), obs1.df),
                     data.col="Leq",
                     covar.col=c('d2TV1','d2TV2','d2TV3'))
   plot(obs.gd)
 }
```

varcov.spatial *Computes Covariance Matrix and Related Results*

Description

Same as geoR's `varcov.spatial`, but taking into account non-Euclidean distances if pertinent.

Usage

```
varcov.spatial(coords = NULL, dists.lowertri = NULL, cov.model = "matern",
  kappa = 0.5, nugget = 0, cov.pars = stop("no cov.pars argument"),
  inv = FALSE, det = FALSE, func.inv = c("cholesky", "eigen", "svd",
  "solve"), scaled = FALSE, only.decomposition = FALSE, sqrt.inv = FALSE,
  try.another.decomposition = TRUE, only.inv.lower.diag = FALSE, ...)
```

Arguments

`coords`	an $n \times 2$ matrix with the coordinates of the data locations. If not provided the argument `dists.lowertri` should be provided instead.
`dists.lowertri`	a vector with the lower triangle of the matrix of distances between pairs of data points. If not provided the argument `coords` should be provided instead.
`cov.model`	a string indicating the type of the correlation function. More details in the documentation for `cov.spatial`. Defaults are equivalent to the *exponential* model.
`kappa`	values of the additional smoothness parameter, only required by the following correlation functions: `"matern"`, `"powered.exponential"`, `"cauchy"` and `"gneiting.matern"`.
`nugget`	the value of the nugget parameter τ^2.
`cov.pars`	a vector with 2 elements or an $ns \times 2$ matrix with the covariance parameters. The first element (if a vector) or first column (if a matrix) corresponds to the variance parameter σ^2. second element or column corresponds to the correlation function parameter ϕ. If a matrix is provided each row corresponds to the parameters of one *spatial structure*. Models with several structures are also called *nested models* in the geostatistical literature.
`inv`	if `TRUE` the inverse of covariance matrix is returned. Defaults to `FALSE`.
`det`	if `TRUE` the logarithmic of the square root of the determinant of the covariance matrix is returned. Defaults to `FALSE`.
`func.inv`	algorithm used for the decomposition and inversion of the covariance matrix. Options are `"chol"` for Cholesky decomposition, `"svd"` for singular value decomposition and `"eigen"` for eigenvalues/eigenvectors decomposition. Defaults to `"chol"`.
`scaled`	logical indicating whether the covariance matrix should be scaled. If `TRUE` the partial sill parameter σ^2 is set to 1. Defaults to `FALSE`.
`only.decomposition`	logical. If `TRUE` only the square root of the covariance matrix is returned. Defaults to `FALSE`.

`sqrt.inv`	if `TRUE` the square root of the inverse of covariance matrix is returned. Defaults to `FALSE`.
`try.another.decomposition`	logical. If `TRUE` and the argument `func.inv` is one of `"cholesky"`, `"svd"` or `"solve"`, the matrix decomposition or inversion is tested and, if it fails, the argument `func.inv` is re-set to `"eigen"`.
`only.inv.lower.diag`	logical. If `TRUE` only the lower triangle and the diagonal of the inverse of the covariance matrix are returned. Defaults to `FALSE`.
`...`	for naw, only for internal usage.

See Also

`varcov.spatial`

`variog` *cost-based empirical variogram*

Description

All the arguments work as in `variog`, except the additional argument `dists.mat`, which takes a symmetric matrix of distances between observation locations

Usage

```
variog(geodata, coords = geodata$coords, data = geodata$data,
  uvec = "default", breaks = "default", trend = "cte", lambda = 1,
  option = c("bin", "cloud", "smooth"), estimator.type = c("classical",
  "modulus"), nugget.tolerance, max.dist, pairs.min = 2, bin.cloud = FALSE,
  direction = "omnidirectional", tolerance = pi/8,
  unit.angle = c("radians", "degrees"), angles = FALSE, dists.mat, messages,
  ...)
```

Arguments

`geodata`	a list containing element `coords` as described next. Typically an object of the class `"geodata"` - a **geoR** data-set. If not provided the arguments `coords` must be provided instead.
`coords`	an $n \times 2$ matrix containing coordinates of the n data locations in each row. Defaults to `geodata$coords`, if provided.
`data`	a vector or matrix with data values. If a matrix is provided, each column is regarded as one variable or realization. Defaults to `geodata$data`, if provided.
`uvec`	a vector with values used to define the variogram binning. Only used when `option = "bin"`. See `DETAILS` below for more details on how to specify the bins.

`breaks`	a vector with values to define the variogram binning. Only used when `option = "bin"`. See `DETAILS` below for more details on how to specify the bins.
`trend`	specifies the mean part of the model. See documentation of `trend.spatial` for further details. Defaults to `"cte"`.
`lambda`	values of the Box-Cox transformation parameter. Defaults to 1 (no transformation). If another value is provided the variogram is computed after transforming the data. A case of particular interest is $\lambda = 0$ which corresponds to log-transformation.
`option`	defines the output type: the options `"bin"` returns values of binned variogram, `"cloud"` returns the variogram cloud and `"smooth"` returns the kernel smoothed variogram. Defaults to `"bin"`.
`estimator.type`	`"classical"` computes the classical method of moments estimator. `"modulus"` returns the variogram estimator suggested by Hawkins and Cressie (see Cressie, 1993, pg 75). Defaults to `"classical"`.
`nugget.tolerance`	a numeric value. Points which are separated by a distance less than this value are considered co-located. Defaults to zero.
`max.dist`	a numerical value defining the maximum distance for the variogram. Pairs of locations separated for distance larger than this value are ignored for the variogram calculation. If not provided defaults takes the maximum distance among all pairs of data locations.
`pairs.min`	a integer number defining the minimum numbers of pairs for the bins. For `option = "bin"`, bins with number of pairs smaller than this value are ignored. Defaults to `NULL`.
`bin.cloud`	logical. If `TRUE` and `option = "bin"` the cloud values for each class are included in the output. Defaults to `FALSE`.
`direction`	a numerical value for the directional (azimuth) angle. This used to specify directional variograms. Default defines the omnidirectional variogram. The value must be in the interval $[0, \pi]$ radians ($[0, 180]$ degrees).
`tolerance`	numerical value for the tolerance angle, when computing directional variograms. The value must be in the interval $[0, \pi/2]$ radians ($[0, 90]$ degrees). Defaults to $\pi/8$.
`unit.angle`	defines the unit for the specification of angles in the two previous arguments. Options are `"radians"` and `"degrees"`, with default to `"radians"`.
`angles`	Logical with default to `FALSE`. If `TRUE` the function also returns the angles between the pairs of points (unimplemented).
`dists.mat`	n x n symmetric matrix with cost-based distances between observations
`messages`	logical. Indicates whether status messages should be printed on the screen (or output device) while the function is running.
`...`	arguments to be passed to the function `ksmooth`, if `option = "smooth"`.

Examples

```
## geodata structure with transformed covariates
```

```
   data(noise)
if (require(sp)) {
   covarnames=sapply(1:3, function(x) paste("d2TV", x, sep=""))
   obs.df <- data.frame(Leq=obs$Leq,
                        1/(1+(as.data.frame(obs)[covarnames]/20)^2))
   obs.gd <- as.geodata(cbind(coordinates(obs), obs.df),
                        data.col="Leq",
                        covar.col=c('d2TV1','d2TV2','d2TV3'))

   ## compute euclidean and cost-based empirical variograms
   vg.std <- variog(obs.gd, trend=~d2TV1*(d2TV2+d2TV3))
   vg.dmat <- variog(obs.gd, trend=~d2TV1*(d2TV2+d2TV3), dists.mat=dd.distmat)

   ## plot and compare empirical variograms
   plot(vg.std, type = 'l', col = 'darkred',
        main = 'Euclidean (red) vs. cost-based (gray) empirical variograms')
   lines(vg.dmat, type = 'l', col = 'darkgray')
}
```

`vecdist` *Vector of reciprocal distances*

Description

If there is a custom definicion of distances, it uses it. Otherwise, it computes the reciprocal distances of points in `x`.

Usage

```
vecdist(x)
```

Arguments

`x`	a `data.frame` with planar coordinates

Details

If there exists a matrix object named `.personal.definition.of.distances` in the global environment, the function returns the lower triangular part of it as a vector, regardless of `x`. Otherwise, the Euclidean distances between points in `x` are returned as a vector.

Index