Package ‘pedometrics’

June 19, 2022

Version 0.12.1

Date 2022-06-18

Title Miscellaneous Pedometric Tools

Description An R implementation of methods employed in the field of pedometrics, soil science discipline dedicated to studying the spatial, temporal, and spatio-temporal variation of soil using statistical and computational methods. The methods found here include the calibration of linear regression models using covariate selection strategies, computation of summary validation statistics for predictions, generation of summary plots, evaluation of the local quality of a geostatistical model of uncertainty, and so on. Other functions simply extend the functionalities of or facilitate the usage of functions from other packages that are commonly used for the analysis of soil data. Formerly available versions of suggested packages no longer available from CRAN can be obtained from the CRAN archive <https://cran.r-project.org/src/contrib/Archive/>.

URL https://github.com/Laboratorio-de-Pedometria/pedometrics-package

BugReports https://github.com/Laboratorio-de-Pedometria/pedometrics-package/issues

Depends R (>= 3.2.0)

Imports lattice, latticeExtra, Rcpp (>= 0.12.0)

LinkingTo Rcpp

Suggests car, fields, georob, grDevices, grid, gstat, knitr, MASS, methods, sp, SpatialTools

License GPL (>= 2)

Encoding UTF-8

Repository CRAN

RoxygenNote 7.2.0

SystemRequirements pandoc

Language en-US

NeedsCompilation yes

Author Alessandro Samuel-Rosa [aut, cre] (<https://orcid.org/0000-0003-0877-1320>), Lucia Helena Cunha dos Anjos [ths]

(<https://orcid.org/0000-0003-0063-3521>),
Gustavo Vasques [ths] (<https://orcid.org/0000-0001-9463-1898>),
Gerard B M Heuvelink [ths] (<https://orcid.org/0000-0003-0959-9358>),
Juan Carlos Ruiz Cuetos [ctb],
Maria Eugenia Polo Garcia [ctb],
Pablo Garcia Rodriguez [ctb],
Joshua French [ctb],
Ken Kleinman [ctb],
Dick Brus [ctb] (<https://orcid.org/0000-0003-2194-4783>),
Frank Harrell Jr [ctb],
Ruo Xu [ctb]

Maintainer Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

Date/Publication 2022-06-19 06:10:02 UTC

R topics documented:

`pedometrics-package` *Pedometric Tools and Techniques*

Description

This package contains many tools and techniques used in the field of pedometrics (see https://en.wikipedia.org/wiki/Pedometr for a definition of *pedometrics*). These tools and techniques were developed to fulfill the demands created by the PhD research project (2012-2016) entitled "Contribution to the Construction of Models for Predicting Soil Properties", developed by Alessandro Samuel-Rosa under the supervision of Dr Lúcia HC Anjos (Universidade Federal Rural do Rio de Janeiro, Brazil), Dr Gustavo M Vasques (Embrapa Solos, Brazil), and Dr Gerard B M Heuvelink (ISRIC - World Soil Information, the Netherlands). The project is/was funded by the CNPq Foundation (Process 140720/2012-0), Ministry of Science and Technology of Brazil, Brasília, DF, 70040-020, Brazil, phone +55 (61) 2022 6002, and the CAPES Foundation (Process ID BEX 11677/13-9), Ministry of Education of Brazil, Brasília, DF, 70040-020, Brazil, phone: +55 (61) 2022 6210.

Details

Several functions simply extend the functionalities of other functions commonly used for the analysis of pedometric data. It should be noted that changes are likely to occur quite often and the use of this package as a dependency for other packages is strongly discouraged.

Author(s)

Author and Maintainer: Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>.

---

adjR2 *Adjusted coefficient of determination*

Description

Calculates the adjusted coefficient of determination of a multiple linear regression model.

Usage

```
adjR2(r2, n, p)
```

Arguments

r2	Numeric vector with the coefficient of determination to be adjusted.
n	Numeric vector providing the number of observations used to fit the multiple linear regression model.
p	Numeric vector providing the number of parameters included in the multiple linear regression model.

Value

A numeric vector with the adjusted coefficient of determination.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

Coefficient of determination. Wikipedia, The Free Encyclopedia. Available at https://en.wikipedia.org/wiki/Coefficient_of_determination.

Examples

```
x <- adjR2(r2 = 0.95, n = 100, p = 80)
```

---

bbox2sp *Create a Spatial* object from a bounding box*

Description

Take the bounding box of a Spatial* object and create a SpatialPoints* or SpatialPolygons* object from it.

Usage

```
bbox2sp(obj, sp = "SpatialPolygons", keep.crs = TRUE)
```

Arguments

`obj`	Object of class Spatial*.
`sp`	Class of the resulting object with options `"SpatialPolygons"` (default), `"SpatialPoints"`, `"SpatialPointsDataFrame"`, and `"SpatialPolygonsDataFrame"`.
`keep.crs`	Logical for assigning the same coordinate reference system to the resulting Spatial* object. Defaults to `keep.crs = TRUE`.

Value

An object of class SpatialPoints* or SpatialPolygons*.

Dependencies

The **sp** package, provider of classes and methods for spatial data in R, is required for `bbox2sp()` to work. The development version of the **sp** package is available on https://github.com/edzer/sp/ while its old versions are available on the CRAN archive at https://cran.r-project.org/src/contrib/Archive/sp/.

Note

Some of the solutions used to build this function were found in the source code of the R-package **intamapInteractive**. As such, the authors of that package, Edzer Pebesma <edzer.pebesma@uni-muenster.de> and Jon Skoien <jon.skoien@gmail.com>, are entitled 'contributors' to the R-package **pedometrics**.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

Edzer Pebesma, Jon Skoien with contributions from Olivier Baume, A. Chorti, D.T. Hristopulos, S.J. Melles and G. Spiliopoulos (2013). *intamapInteractive: procedures for automated interpolation - methods only to be used interactively, not included in intamap package.* R package version 1.1-10. https://CRAN.R-project.org/package=intamapInteractive.

Examples

```
if (require(sp)) {
  data(meuse, package = "sp")
  sp::coordinates(meuse) <- ~ x + y
  bb <- bbox2sp(obj = meuse, keep.crs = FALSE)
}
```

`buildModelSeries` *Build a series of linear models using automated variable selection*

Description

Build a series of linear models with `stats::lm()` using one or more automated variable selection methods implemented in the functions `stepVIF()` and `MASS::stepAIC()`.

Usage

```
buildModelSeries(
  formula,
  data,
  vif = FALSE,
  vif.threshold = 10,
  vif.verbose = FALSE,
  aic = FALSE,
  aic.direction = "both",
  aic.trace = FALSE,
  aic.steps = 5000,
  ...
)

buildMS(
  formula,
  data,
  vif = FALSE,
  vif.threshold = 10,
  vif.verbose = FALSE,
  aic = FALSE,
  aic.direction = "both",
  aic.trace = FALSE,
  aic.steps = 5000,
  ...
)
```

Arguments

`formula`	A list containing one or several model formulas (a symbolic description of the model to be fitted).
`data`	Data frame containing the variables in the model formulas.
`vif`	Logical for performing backward variable selection using the Variance-Inflation Factor (VIF). Defaults to `vif = FALSE`.
`vif.threshold`	Numeric value setting the maximum acceptable VIF value. Defaults to `vif.threshold = 10`.
`vif.verbose`	Logical for printing iteration results of backward variable selection using the VIF. Defaults to `vif.verbose = FALSE`.
`aic`	Logical for performing variable selection using Akaike's Information Criterion (AIC). Defaults to `aic = FALSE`.
`aic.direction`	Character string setting the direction of variable selection when using AIC, with options `"both"` (default), `"forward"`, and `"backward"`.
`aic.trace`	Logical for printing iteration results of variable selection using the AIC. Defaults to `aic.trace = FALSE`.
`aic.steps`	Integer value setting the maximum number of steps to be considered for variable selection using the AIC. Defaults to `aic.steps = 5000`.
`...`	Further arguments passed to `MASS::stepAIC()`.

Details

`buildModelSeries()` was devised to deal with a list of linear model formulas. The main objective is to bring together several functions commonly used when building linear models, such as automated variable selection. In the current implementation, variable selection can be done using `stepVIF()` or `MASS::stepAIC()` or both. `stepVIF()` is a backward variable selection procedure, while `MASS::stepAIC()` supports backward, forward, and bidirectional variable selection. For more information about these functions, please visit their respective help pages.

An important feature of `buildModelSeries()` is that it records the initial number of candidate predictor variables and observations offered to the model, and adds this information as an attribute to the final selected model. Such feature was included because variable selection procedures result biased linear models (too optimistic), and the effective number of degrees of freedom is close to the number of candidate predictor variables initially offered to the model (Harrell, 2001). With the initial number of candidate predictor variables and observations offered to the model, one can calculate penalized or adjusted measures of model performance. For models built using `buildModelSeries()`, this can be done using `statsModelSeries()`.

Some important details should be clear when using `buildModelSeries()`:

- this function was originally devised to deal with a list of formulas, but can also be used with a single formula;
- in the current implementation, `stepVIF()` runs before `MASS::stepAIC()`;
- function arguments imported from `MASS::stepAIC()` and `stepVIF()` were named as in the original functions, and received a prefix (`aic` or `vif`) to help the user identifying which function is affected by a given argument without having to go check the documentation.

Value

A list containing the fitted linear models.

TODO

Add option to set the order in which `MASS::stepAIC()` and `stepVIF()` are run.

Dependencies

The **MASS** package, provider of support functions and datasets for Venables and Ripley's Modern Applied Statistics with S, is required for `buildModelSeries()` to work. The development version of the **MASS** package is available on `https://www.stats.ox.ac.uk/pub/MASS4/` while its old versions are available on the CRAN archive at `https://cran.r-project.org/src/contrib/Archive/MASS/`.

Author(s)

Alessandro Samuel-Rosa <`alessandrosamuelrosa@gmail.com`>

References

Harrell, F. E. (2001) *Regression modelling strategies: with applications to linear models, logistic regression, and survival analysis.* First edition. New York: Springer.

Venables, W. N. and Ripley, B. D. (2002) *Modern applied statistics with S.* Fourth edition. New York: Springer.

A. Samuel-Rosa, G. B. M. Heuvelink, G. de Mattos Vasques, and L. H. C. dos Anjos, Do more detailed environmental covariates deliver more accurate soil maps?, *Geoderma*, vol. 243–244, pp. 214–227, May 2015, doi: 10.1016/j.geoderma.2014.12.017.

See Also

`stepVIF()`, `statsMS()`

Examples

```
if (interactive()) {
  # based on the second example of MASS::stepAIC()
  library("MASS")
  cpus1 <- cpus
  for(v in names(cpus)[2:7])
    cpus1[[v]] <- cut(cpus[[v]], unique(stats::quantile(cpus[[v]])),
                      include.lowest = TRUE)
  cpus0 <- cpus1[, 2:8]  # excludes names, authors' predictions
  cpus.samp <- sample(1:209, 100)
  cpus.form <- list(formula(log10(perf) ~ syct + mmin + mmax + cach + chmin +
                    chmax + perf),
                    formula(log10(perf) ~ syct + mmin + cach + chmin + chmax),
                    formula(log10(perf) ~ mmax + cach + chmin + chmax + perf))
  data <- cpus1[cpus.samp,2:8]
  cpus.ms <- buildModelSeries(cpus.form, data, vif = TRUE, aic = TRUE)
}
```

checkGMU — *Evaluation of geostatistical models of uncertainty*

Description

Evaluate the local quality of a geostatistical model of uncertainty (GMU) using summary measures and graphical displays.

Usage

```
checkGMU(
  observed,
  simulated,
  pi = seq(0.01, 0.99, 0.01),
  symmetric = TRUE,
  plotit = TRUE
)
```

Arguments

`observed`	Vector of observed values at the validation points. See 'Details' for more information.
`simulated`	Data frame or matrix with simulated values (columns) for each validation point (rows). See 'Details' for more information.
`pi`	Vector defining the width of the series of probability intervals. Defaults to `pi = seq(0.01, 0.99, 0.01)`. See 'Details' for more information.
`symmetric`	Logical for choosing the type of probability interval. Defaults to `symmetric = TRUE`. See 'Details' for more information.
`plotit`	Logical for plotting the results. Defaults to `plotit = TRUE`.

Details

There is no standard way of evaluating the local quality of a GMU. The collection of summary measures and graphical displays presented here is far from being comprehensive. A few definitions are given bellow.

Error statistics: Error statistics measure how well the GMU predicts the measured values at the validation points. Four error statistics are presented:

Mean error (ME) Measures the bias of the predictions of the GMU, being defined as the mean of the differences between the average of the simulated values and the observed values, i.e. the average of all simulations is taken as the predicted value.

Mean squared error (MSE) Measures the accuracy of the predictions of the GMU, being defined as the mean of the squared differences between the average of the simulated values and the observed values.

Scaled root mean squared error (SRMSE) Measures how well the GMU estimate of the prediction error variance (PEV) approximates the observed prediction error variance, where the first is given by the variance of the simulated values, while the second is given by the squared differences between the average of the simulated values, i.e. the squared error (SE). The SRMSE is computed as the average of SE / PEV, where SRMSE > 1 indicates underestimation, while SRMSE < 1 indicates overestimation.

Pearson correlation coefficient Measures how close the GMU predictions are to the observed values. A scatter plot of the observed values versus the average of the simulated values can be used to check for possible unwanted outliers and non-linearities. The square of the Pearson correlation coefficient measures the fraction of the overall spread of observed values that is explained by the GMU, that is, the amount of variance explained (AVE), also known as coefficient of determination or ratio of scatter.

Coverage probabilities: The coverage probability of an interval is given by the number of times that that interval contains its parameter over several replications of an experiment. For example, consider the interquartile range $IQR = Q3 - Q1$ of a Gaussian distributed variable with mean equal to zero and variance equal to one. The nominal coverage probability of the IQR is 0.5, i.e. two quarters of the data fall within the IQR. Suppose we generate a Gaussian distributed *random* variable with the same mean and variance and count the number of values that fall within the IQR defined above: about 0.5 of its values will fall within the IQR. If we continue generating Gaussian distributed *random* variables with the same mean and variance, on average, 0.5 of the values will fall in that interval.

Coverage probabilities are very useful to evaluate the local quality of a GMU: the closer the observed coverage probabilities of a sequence of probability intervals (PI) are to the nominal coverage probabilities of those PIs, the better the modeling of the local uncertainty.

Two types of PIs can be used here: symmetric, median-centered PIs, and left-bounded PIs. Papritz & Dubois (1999) recommend using left-bounded PIs because they are better at evidencing deviations for both large and small PIs. The authors also point that the coverage probabilities of the symmetric, median-centered PIs can be read from the coverage probability plots produced using left-bounded PIs.

In both cases, the PIs are computed at each validation location using the quantiles of the conditional cumulative distribution function (ccdf) defined by the set of realizations at that validation location. For a sequence of PIs of increasing width, we check which of them contains the observed value at all validation locations. We then average the results over all validation locations to compute the proportion of PIs (with the same width) that contains the observed value: this gives the coverage probability of the PIs.

Deutsch (1997) proposed three summary measures of the coverage probabilities to assess the local *goodness* of a GMU: accuracy (\$A\$), precision (\$P\$), and goodness (\$G\$). According to Deutsch (1997), a GMU can be considered “good” if it is both accurate and precise. Although easy to compute, these measures seem not to have been explored by many geostatisticians, except for the studies developed by Pierre Goovaerts and his later software implementation (Goovaerts, 2009). Richmond (2001) suggests that they should not be used as the only measures of the local quality of a GMU.

Accuracy An accurate GMU is that for which the proportion p^* of true values falling within the $p$ PI is equal to or larger than the nominal probability $p$, that is, when $p^* \geq p$. In the coverage probability plot, a GMU will be more accurate when all points are on or above the 1:1 line. The range of $A$ goes from 0 (lest accurate) to 1 (most accurate).

Precision The *precision*, $P$, is defined only for an accurate GMU, and measures how close p^* is to $p$. The range of $P$ goes from 0 (lest precise) to 1 (most precise). Thus, a GMU will be more accurate when all points in the PI-width plot are on or above the 1:1 line.

Goodness The *goodness*, $G$, is a measure of the departure of the points from the 1:1 line in the coverage probability plot. $G$ ranges from 0 (minimum goodness) to 1 (maximum goodness), the maximum $G$ being achieved when $p^* = p$, that is, all points in both coverage probability and interval width plots are exactly on the 1:1 line.

It is worth noting that the coverage probability and PI-width plots are relevant mainly to GMU created using *conditional simulations*, that is, simulations that are locally conditioned to the data observed at the validation locations. Conditioning the simulations locally serves the purposes of honoring the available data and reducing the variance of the output realizations. This is why one would like to find the points falling above the 1:1 line in both coverage probability and PI-width plots. For *unconditional simulations*, that is, simulations that are only globally conditioned to the histogram (and variogram) of the data observed at the validation locations, one would expect to find that, over a large number of simulations, the whole set of possible values (i.e. the global histogram) can be generated at any node of the simulation grid. In other words, it is expected to find all points on the 1:1 line in both coverage probability and PI-width plots. Deviations from the 1:1 line could then be used as evidence of problems in the simulation.

Value

A `list` of summary measures and plots of the coverage probability and width of probability intervals.

Note

Comments by Pierre Goovaerts `<pierre.goovaerts@biomedware.com>` were important to describe how to use the coverage probability and PI-width plots when a GMU is created using unconditional simulations.

Author(s)

Alessandro Samuel-Rosa `<alessandrosamuelrosa@gmail.com>`

References

Deutsch, C. Direct assessment of local accuracy and precision. Baafi, E. Y. & Schofield, N. A. (Eds.) *Geostatistics Wollongong '96*. Dordrecht: Kinwer Academic Publishers, v. I, p. 115-125, 1997.

Papritz, A. & Dubois, J. R. Mapping heavy metals in soil by (non-)linear kriging: an empirical validation. Gómez-Hernández, J.; Soares, A. & Froidevaux, R. (Eds.) *geoENV II – Geostatistics for Environmental Applications*. Springer, p. 429-440, 1999.

Goovaerts, P. Geostatistical modelling of uncertainty in soil science. *Geoderma*. v. 103, p. 3 - 26, 2001.

Goovaerts, P. AUTO-IK: a 2D indicator kriging program for the automated non-parametric modeling of local uncertainty in earth sciences. *Computers & Geosciences*. v. 35, p. 1255-1270, 2009.

Richmond, A. J. Maximum profitability with minimum risk and effort. Xie, H.; Wang, Y. & Jiang, Y. (Eds.) *Proceedings 29th APCOM*. Lisse: A. A. Balkema, p. 45-50, 2001.

Ripley, B. D. *Stochastic simulation*. New York: John Wiley & Sons, p. 237, 1987.

Examples

```
if (interactive()) {
  set.seed(2001)
  observed <- round(rnorm(100), 3)
  simulated <- t(
    sapply(1:length(observed), function (i) round(rnorm(100), 3)))
  resa <- checkGMU(observed, simulated, symmetric = T)
  resb <- checkGMU(observed, simulated, symmetric = F)
  resa$error; resb$error
  resa$goodness; resb$goodness
}
```

---

cont2cat *Categorize/stratify numerical variable(s)*

Description

Create break points, compute strata proportions, and stratify numerical variable(s) to create categorical variable(s).

Usage

```
cont2cat(x, breaks, integer = FALSE)

breakPoints(x, n, type = "area", prop = FALSE)

stratify(x, n, type = "area", integer = FALSE)
```

Arguments

x	Vector, data frame or matrix with data on the numerical variable(s) to be categorized/stratified.
breaks	Vector or list containing the lower and upper limits that should be used to break the numerical variable(s) into categories. See 'Details' for more information.

`integer`	Logical value indicating if the categorical variable(s) should be returned as `integers`. Defaults to `integer = FALSE`, i.e. the variable(s) will be returned as `factors`.
`n`	Integer value indicating the number of categories/strata that should be created.
`type`	Character value indicating the type of categories/strata that should be used, with options `"area"` (default), for equal-area, and `"range"`, for equal-range strata.
`prop`	Logical value indicating if the strata proportions should be returned? Defaults to `prop = FALSE`.

Details

Argument `breaks` must be a vector if `x` is a vector, but a list if `x` is a data frame or matrix. Using a list allows breaking each column of `x` into different number of categories.

Value

A vector, data frame, or matrix, depending on the class of `x`.

Dependencies

The **SpatialTools** package, provider of tools for spatial data analysis in R, is required for `breakPoints()` and `stratify()` to work. The development version of the **SpatialTools** package is available on `https://github.com/jfrench/SpatialTools` while its old versions are available on the CRAN archive at `https://cran.r-project.org/src/contrib/Archive/SpatialTools/`.

Reverse dependencies

The **spsann** package, provider of methods for the optimization of sample configurations using spatial simulated annealing in R, requires `breakPoints()`, `cont2cat()` and `stratify()` for some of its functions to work. The development version of the **spsann** package is available on `https://github.com/Laboratorio-de-Pedometria/spsann-package`.

Author(s)

Alessandro Samuel-Rosa <`alessandrosamuelrosa@gmail.com`>

References

B. Minasny and A. B. McBratney. A conditioned Latin hypercube method for sampling in the presence of ancillary information. *Computers & Geosciences*, vol. 32, no. 9, pp. 1378–1388, Nov. 2006, doi: 10.1016/j.cageo.2005.12.009.

T. Hengl, D. G. Rossiter, and A. Stein. Soil sampling strategies for spatial prediction by correlation with auxiliary maps. *Australian Journal of Soil Research*, vol. 41, no. 8, pp. 1403–1422, 2003, doi: 10.1071/SR03005.

Examples

```
if (require(SpatialTools)) {
  ## Compute the break points of marginal strata
  x <- data.frame(x = round(rnorm(10), 1), y = round(rlnorm(10), 1))
  x <- breakPoints(x = x, n = 4, type = "area", prop = TRUE)

  ## Convert numerical data into categorical data
  # Matrix
  x <- y <- c(1:10)
  x <- cbind(x, y)
  breaks <- list(c(1, 2, 4, 8, 10), c(1, 5, 10))
  y <- cont2cat(x, breaks)

  # Data frame
  x <- y <- c(1:10)
  x <- data.frame(x, y)
  breaks <- list(c(1, 2, 4, 8, 10), c(1, 5, 10))
  y <- cont2cat(x, breaks, integer = TRUE)

  # Vector
  x <- c(1:10)
  breaks <- c(1, 2, 4, 8, 10)
  y <- cont2cat(x, breaks, integer = TRUE)

  ## Stratification
  x <- data.frame(x = round(rlnorm(10), 1), y = round(rnorm(10), 1))
  x <- stratify(x = x, n = 4, type = "area", integer = TRUE)
  x
}
```

`coordenadas` *Defunct functions*

Description

The functions listed here are no longer part of the **pedometrics** package. If you need to use any of these functions, you can still find them at `https://github.com/samuel-rosa/ASRtools` or `https://cran.r-project.org/src/contrib/Archive/pedometrics/`.

Usage

```
coordenadas(...)

cdfPlot(...)

cdfStats(...)

cdfTable(...)
```

```
gcpDiff(...)

trend.terms(...)

trend.matrix(...)
```

Arguments

`...`	Not used.

Value

No return value, called for side effects.

Author(s)

Alessandro Samuel-Rosa <`alessandrosamuelrosa@gmail.com`>
Tony Olsen <`Olsen.Tony@epa.gov`>
Tom Kincaid <`Kincaid.Tom@epa.gov`>

`cramer` *Association between categorical variables*

Description

Compute the Cramer's V, a descriptive statistic that measures the association between categorical variables.

Usage

```
cramer(x)
```

Arguments

`x`	Data frame or matrix with a set of categorical variables.

Details

Any integer variable is internally converted to a factor.

Value

A matrix with the Cramer's V between the categorical variables.

Reverse dependencies

The **spsann** package, provider of methods for the optimization of sample configurations using spatial simulated annealing in R, requires `cramer()` for some of its functions to work. The development version of the **spsann** package is available on `https://github.com/Laboratorio-de-Pedometria/spsann-package`.

Note

The original code is available at https://sas-and-r.blogspot.com/, Example 8.39: calculating Cramer's V, posted by Ken Kleinman on Friday, June 3, 2011. As such, Ken Kleinman <Ken_Kleinman@hms.harvard.edu> is entitled a 'contributor' to the R-package **pedometrics**.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

Cramér, H. *Mathematical methods of statistics*. Princeton: Princeton University Press, p. 575, 1946.

Everitt, B. S. *The Cambridge dictionary of statistics*. Cambridge: Cambridge University Press, p. 432, 2006.

Examples

```
if (interactive()) {
  data(meuse, package = "sp")
  str(meuse)
  test <- cramer(meuse[, c("ffreq", "soil", "lime", "landuse")])
}
```

---

gcpVector *Calculate module and azimuth*

Description

Calculate the module and azimuth of the difference on x and y coordinates between two sets of ground control points (GCP). It is suited to perform calculations for topographical coordinates only. The origin is set in the y coordinate, and rotation performed clockwise.

Usage

```
gcpVector(dx, dy)
```

Arguments

dx	Numeric vector containing the difference on the 'x' coordinate between two sets of GCP.
dy	Numeric vector containing the difference on the 'y' coordinate between two sets of GCP.

Value

A data frame containing the module, its square and azimuth. These three columns are named 'module', 'sq.module' and 'azimuth'.

Note

This function was adapted from package's **VecStatGraphs2D** function `LoadData()`.

Author(s)

Juan Carlos Ruiz Cuetos <`bilba_t@hotmail.com`>
Maria Eugenia Polo Garcia <`mepolo@unex.es`>
Pablo Garcia Rodriguez <`pablogr@unex.es`>
Alessandro Samuel-Rosa <`alessandrosamuelrosa@gmail.com`>

References

Ruiz-Cuetos J.C., Polo M.E. and Rodriguez P.G. (2012). *VecStatGraphs2D: Vector analysis using graphical and analytical methods in 2D*. R package version 1.6. `https://CRAN.R-project.org/package=VecStatGraphs2D`.

Examples

```
x <- gcpVector(dx = rnorm(3, 5, 10), dy = rnorm(3, 5, 10))
```

`isNumint` *Tests for data types*

Description

Evaluate the data type contained in an object.

Usage

```
isNumint(x)

allNumint(x)

anyNumint(x)

whichNumint(x)

allInteger(x)

anyInteger(x)

whichInteger(x)

allFactor(x)

anyFactor(x)
```

```
whichFactor(x)

allNumeric(x)

anyNumeric(x)

whichNumeric(x)

uniqueClass(x)
```

Arguments

x	Object to be tested.

Value

TRUE or FALSE depending on whether x contains a given data type.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

See Also

base::is.numeric(), base::is.integer(), base::is.factor().

Examples

```
# Vector of integers
x <- 1:10
isNumint(x) # FALSE

# Vector of numeric integers
x <- as.numeric(x)
isNumint(x) # TRUE

# Vector of numeric values
x <- c(1.1, 1, 1, 1, 2)
isNumint(x) # FALSE
allNumint(x) # FALSE
anyNumint(x) # TRUE
whichNumint(x)

# Single numeric integer
isNumint(1) # TRUE

# Single numeric value
isNumint(1.1) # FALSE
```

`plotCor` *Correlation plot*

Description

Plotting correlation matrices.

Usage

```
plotCor(r, r2, col, breaks, col.names, ...)
```

Arguments

`r`	A square matrix with correlation values.
`r2`	(optional) A second square matrix with correlation values.
`col`	(optional) Color table to use for `image` – see `graphics::image()` for details. The default is a colorblind-friendly palette created using the **RColorBrewer** palette `"RdBu"`.
`breaks`	(optional) Break points in sorted order to indicate the intervals for assigning the colors. See `fields::image.plot()` for more details.
`col.names`	(optional) Character vector with short (up to 5 characters) column names.
`...`	(optional) Additional parameters passed to plotting functions.

Details

A correlation plot in an alternative way of showing the strength of the empirical correlations between variables. This is done by using a diverging color palette, where the darker the color, the stronger the absolute correlation value.

`plotCor()` can also be used to compare correlations between the same variables at different points in time or space or for different observations. This is done by passing two square correlation matrices using arguments `r` and `r2`. The lower triangle of the resulting correlation plot will contain correlations from `r`, correlations from `r2` will be in the upper triangle, and the diagonal will be empty.

Value

A correlation plot.

Dependencies

The **fields** package, provider of tools for spatial data in R, is required for `plotCor()` to work. The development version of the **fields** package is available on `https://github.com/dnychka/fieldsRPackage` while its old versions are available on the CRAN archive at `https://cran.r-project.org/src/contrib/Archive/fields/`.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

Neuwirth E (2022). *RColorBrewer: ColorBrewer Palettes*. R package version 1.1-3, https://CRAN.R-project.org/package=RColorBrewer.

Examples

```
if (all(c(require(sp), require(fields)))) {
  data(meuse, package = "sp")
  cols <- c("cadmium", "copper", "lead", "zinc", "elev", "dist", "om")

  # A single correlation matrix
  r <- cor(meuse[1:20, cols], use = "complete")
  r <- round(r, 2)
  plotCor(r)

  # Two correlation matrices: r2 goes in the upper triangle
  r2 <- cor(meuse[21:40, cols], use = "complete")
  r2 <- round(r2, 2)
  plotCor(r, r2)
}
```

---

plotESDA *Plots for exploratory spatial data analysis (ESDA)*

Description

Create four plots for exploratory spatial data analysis (ESDA): histogram + density plot, bubble plot, variogram plot, and variogram map.

Usage

```
plotESDA(
  z,
  lat,
  lon,
  lags = NULL,
  cutoff = NULL,
  width = c(cutoff/20),
  leg.pos = "right"
)
```

Arguments

`z`	Vector of numeric values of the variable for with ESDA plots should be created.
`lat`	Vector of numeric values containing the y coordinate (latitude) of the point locations where the z variable was observed.
`lon`	Vector of numeric values containing the x coordinate (longitude) of the point locations where the z variable was observed.
`lags`	(optional) Numerical vector; upper boundaries of lag-distance classes. See argument `boundaries` of `gstat::variogram()` for more info.
`cutoff`	(optional) Integer value defining the spatial separation distance up to which point pairs are included in semi-variance estimates. Defaults to the length of the diagonal of the box spanning the data divided by three.
`width`	Integer value specifying the width of subsequent distance intervals into which data point pairs are grouped for semi-variance estimates. Defaults to `width = cutoff / 20`.
`leg.pos`	(optional) Character value indication the location of the legend of the bubble plot. Defaults to `leg.pos = "right"`.

Details

The user should visit the help pages of `gstat::variogram()`, `plotHD()`, `sp::bubble()` and `sp::spplot()` to obtain more details about the main functions used to built `plotESDA()`.

Value

Four plots: histogram and density plot, bubble plot, empirical variogram, and variogram map.

Dependencies

The **sp** package, provider of classes and methods for spatial data in R, is required for `plotESDA()` to work. The development version of the **sp** package is available on `https://github.com/edzer/sp/` while its old versions are available on the CRAN archive at `https://cran.r-project.org/src/contrib/Archive/sp/`.

The **gstat** package, provider of methods for spatial and spatio-temporal geostatistical modelling, prediction and simulation in R, is required for `plotESDA()` to work. The development version of the **sp** package is available on `https://github.com/r-spatial/gstat` while its old versions are available on the CRAN archive at `https://cran.r-project.org/src/contrib/Archive/gstat/`.

Author(s)

Alessandro Samuel-Rosa <`alessandrosamuelrosa@gmail.com`>

References

Cressie, N.A.C. (1993) *Statistics for Spatial Data*. New York: John Wiley and Sons, p.900, 1993.

Pebesma, E.J. (2004) Multivariable geostatistics in S: the gstat package. *Computers and Geosciences*, 30:683-691, 2004.

Webster, R., Oliver, M.A. *Geostatistics for environmental scientists*. Chichester: John Wiley and Sons, p.315, 2007.

See Also

`plotHD()`

Examples

```
if (all(require(sp), require(gstat))) {
  data(meuse, package = "sp")
  p <- plotESDA(z = meuse$zinc, lat = meuse$y, lon = meuse$x)
}
```

`plotHist` *Histogram and density plot*

Description

Plot a histogram and a density plot of a single variable using the R-package **lattice**.

Usage

```
plotHist(
  x,
  HD = "over",
  nint = 20,
  digits = 2,
  stats = TRUE,
  BoxCox = FALSE,
  col = c("lightgray", "black"),
  lwd = c(1, 1),
  lty = "dashed",
  xlim,
  ylim,
  ...
)

plotHD(
  x,
  HD = "over",
  nint = 20,
  digits = 2,
  stats = TRUE,
  BoxCox = FALSE,
  col = c("lightgray", "black"),
  lwd = c(1, 1),
  lty = "dashed",
  xlim,
  ylim,
  ...
)
```

Arguments

`x`	Vector of numeric values of the variable for which the histogram and density plot should be created.
`HD`	Character value indicating the type of plot to be created. Available options are `"over"`, to create a histogram superimposed by the theoretical density plot of a normally distributed variable, and `"stack"`, to create a histogram and an empirical density plot in separated panels. Defaults to `HD = "over"`.
`nint`	Integer specifying the number of histogram bins. Defaults to `nint = 20`.
`digits`	Integer indicating the number of decimal places to be used when printing the statistics of the variable `x`. Defaults to `digits = 2`.
`stats`	Logical to indicate if descriptive statistics of the variable `x` should be added to the plot. Available only when `HD = "over"`. The function tries to automatically find the best location to put the descriptive statistics given the shape of the histogram. Defaults to `stats = TRUE`.
`BoxCox`	Logical to indicate if the variable `x` should be transformed using the Box-Cox family of power transformations. The estimated lambda value of the Box-Cox transform is printed in the console. It is set to zero when negative. Defaults to `BoxCox = FALSE`.
`col`	Vector of two elements, the first indicating the color of the histogram, the second indicating the color of the density plot. Defaults to `col = c("lightgray", "black")`.
`lwd`	Vector of two elements, the first indicating the line width of the histogram, the second indicating the line width of the density plot. Defaults to `lwd = c(1, 1)`.
`lty`	Character value indicating the line type for the density plot. Defaults to `lty = "dashed"`.
`xlim`	Vector of two elements defining the limits of the x axis. The function automatically optimizes `xlim` based on the density plot.
`ylim`	Vector of two elements defining the limits of the y axis. The function automatically optimizes `ylim` based both histogram and density plot.
`...`	Other arguments that can be passed to **lattice** functions. There is no guarantee that they will work.

Details

The user should visit the help pages of `lattice::histogram()`, `lattice::densityplot()`, `lattice::panel.mathdensit` `car::powerTransform()`, and `car::bcPower()` to obtain more details about the main functions used to built `plotHD()`.

Value

An object of class "trellis". The lattice::update.trellis() method can be used to update components of the object and the lattice::print.trellis() print method (usually called by default) will plot it on an appropriate plotting device.

Dependencies

The **car** package, provider of functions to accompany Fox and Weisberg's An R Companion to Applied Regression, is required for plotHist() to work. The development version of the **car** package is available on https://r-forge.r-project.org/projects/car/ while its old versions are available on the CRAN archive at https://cran.r-project.org/src/contrib/Archive/car/.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

Sarkar, Deepayan (2008) *Lattice: Multivariate Data Visualization with R*, Springer. http://lmdvr.r-forge.r-project.org/

See Also

lattice::histogram(), lattice::densityplot(), lattice::panel.mathdensity()

Examples

```
if (all(c(require(car), require(lattice), require(latticeExtra)))) {
  x <- rnorm(100, 10, 2)
  p1 <- plotHist(x, HD = "stack")
  p2 <- plotHist(x, HD = "over")
}
```

---

plotModelSeries *Model series plot*

Description

Produce a graphical output to examine the effect of using different model specifications (design) on the predictive performance of these models (a model series). Devised to access the results of buildModelSeries() and statsMS(), but can be easily adapted to work with any model structure and performance measure.

Usage

```
plotModelSeries(
  obj,
  grid,
  line,
  ind,
  type = c("b", "g"),
  pch = c(20, 2),
  size = 0.5,
  arrange = "desc",
  color = NULL,
  xlim = NULL,
  ylab = NULL,
  xlab = NULL,
  at = NULL,
  ...
)

plotMS(
  obj,
  grid,
  line,
  ind,
  type = c("b", "g"),
  pch = c(20, 2),
  size = 0.5,
  arrange = "desc",
  color = NULL,
  xlim = NULL,
  ylab = NULL,
  xlab = NULL,
  at = NULL,
  ...
)
```

Arguments

`obj`	Object of class `data.frame`, generally returned by `statsMS()`, containing: 1. a series of performance statistics of several models, and 2. the design information of each model. See ‘Details’ for more information.
`grid`	Vector of integer values or character strings indicating the columns of the `data.frame` containing the design data which will be gridded using the function `lattice::levelplot()`. See ‘Details’ for more information.
`line`	Character string or integer value indicating which of the performance statistics (usually calculated by `statsMS()`) should be plotted using the function `lattice::xyplot()`. See ‘Details’ for more information.

`ind`	Integer value indicating for which group of models the mean rank is to be calculated. See 'Details' for more information.
`type`	Vector of character strings indicating some of the effects to be used when plotting the performance statistics using `lattice::xyplot()`. Defaults to `type = c("b", "g")`. See `lattice::panel.xyplot()` for more information on how to set this argument.
`pch`	Vector with two integer values specifying the symbols to be used to plot points. The first sets the symbol used to plot the performance statistic, while the second sets the symbol used to plot the mean rank of the indicator set using argument `ind`. Defaults to `pch = c(20, 2)`. See `graphics::points()` for possible values and their interpretation.
`size`	Numeric value specifying the size of the symbols used for plotting the mean rank of the indicator set using argument `ind`. Defaults to `size = 0.5`. See `grid::grid.points()` for more information.
`arrange`	Character string indicating how the model series should be arranged, which can be in ascending (`"asc"`) or descending (`"desc"`, default) order.
`color`	Vector defining the colors to be used in the grid produced by function `lattice::levelplot()`. If `color = NULL`, defaults to `color = cm.colors(n)`, where `n` is the number of unique values in the columns defined by argument `grid`. See `grDevices::cm.colors()` to see how to use other color palettes.
`xlim`	Numeric vector of length 2, giving the x coordinates range. If `xlim = NULL` (which is the recommended value), defaults to `xlim = c(0.5, dim(obj)[1] + 0.5)`. This is, so far, the optimum range for adequate plotting.
`ylab`	Character vector of length 2, giving the y-axis labels. When `obj` is a `data.frame` returned by `statsMS()`, and the performance statistic passed to argument `line` is one of those calculated by `statsMS()` (`"candidates"`, `"df"`, `"aic"`, `"rmse"`, `"nrmse"`, `"r2"`, `"adj_r2"`, or `"ADJ_r2"`), the function tries to automatically identify the correct `ylab`.
`xlab`	Character vector of unit length, the x-axis label. Defaults `xlab = "Model ranking"`.
`at`	Numeric vector indicating the location of tick marks along the x axis (in native coordinates).
`...`	Other arguments for plotting, although most of these have no been tested. Argument `asp`, for example, is not effective since the function automatically identifies the best aspect for plotting based on the dimensions of the design data.

Details

This section gives more details about arguments `obj`, `grid`, `line`, `arrange`, and `ind`.

obj: The argument `obj` usually constitutes a `data.frame` returned by `statsMS()`. However, the user can use any `data.frame` object as far as it contains the two basic units of information needed:

1. design data passed with argument `grid`
2. performance statistic passed with argument `line`

grid: The argument `grid` indicates the *design* data which is used to produce the grid output in the top of the model series plot. By *design* we mean the data that specify the structure of each model and how they differ from each other. Suppose that eight linear models were fit using three types of predictor variables (`a`, `b`, and `c`). Each of these predictor variables is available in two versions that differ by their accuracy, where `0` means a less accurate predictor variable, while `1` means a more accurate predictor variable. This yields 2^3 = 8 total possible combinations. The *design* data would be of the following form:

```
> design
  a b c
1 0 0 0
2 0 0 1
3 0 1 0
4 1 0 0
5 0 1 1
6 1 0 1
7 1 1 0
8 1 1 1
```

line: The argument `line` corresponds to the performance statistic that is used to arrange the models in ascending or descending order, and to produce the line output in the bottom of the model series plot. For example, it can be a series of values of adjusted coefficient of determination, one for each model:

```
adj_r2 <- c(0.87, 0.74, 0.81, 0.85, 0.54, 0.86, 0.90, 0.89)
```

arrange: The argument `arrange` automatically arranges the model series according to the performance statistics selected with argument `line`. If `obj` is a `data.frame` returned by `statsMS()`, then the function uses standard arranging approaches. For most performance statistics, the models are arranged in descending order. The exception is when `"r2"`, `"adj_r2"`, or `"ADJ_r2"` are used, in which case the models are arranged in ascending order. This means that the model with lowest value appears in the leftmost side of the model series plot, while the models with the highest value appears in the rightmost side of the plot.

```
> arrange(obj, adj_r2)
  id a b c adj_r2
1  5 1 0 1   0.54
2  2 0 0 1   0.74
3  3 1 0 0   0.81
4  4 0 1 0   0.85
5  6 0 1 1   0.86
6  1 0 0 0   0.87
7  8 1 1 1   0.89
8  7 1 1 0   0.90
```

This results suggest that the best performing model is that of `id = 7`, while the model of `id = 5` is the poorest one.

ind: The model series plot allows to see how the design influences model performance. This is achieved mainly through the use of different colors in the grid output, where each unique value in the *design* data is represented by a different color. For the example given above, one could try to see if the models built with the more accurate versions of the predictor variables have a better performance by identifying their relative distribution in the model series plot. The models placed at the rightmost side of the plot are those with the best performance.

The argument `ind` provides another tool to help identifying how the design, more specifically how each variable in the *design* data, influences model performance. This is done by simply calculating the mean ranking of the models that were built using the updated version of each predictor variable. This very same mean ranking is also used to rank the predictor variables and thus identify which of them is the most important.

After arranging the `design` data described above using the adjusted coefficient of determination, the following mean rank is obtained for each predictor variable:

```
> rank_center
     a    b    c
1 5.75 6.25 5.25
```

This result suggests that the best model performance is obtained when using the updated version of the predictor variable `b`. In the model series plot, the predictor variable `b` appears in the top row, while the predictor variable `c` appears in the bottom row.

Value

An object of class `"trellis"` consisting of a model series plot.

Dependencies

The **grDevices** package, provider of graphics devices and support for colours and fonts in R, is required for `plotModelSeries()` to work.

The **grid** package, a rewrite of the graphics layout capabilities in R, is required for `plotModelSeries()` to work.

Warning

Use the original functions `lattice::xyplot()` and `lattice::levelplot()` for higher customization.

Note

Some of the solutions used to build this function were found in the source code of the R-package **mvtsplot**. As such, the author of that package, Roger D. Peng `<rpeng@jhsph.edu>`, is entitled 'contributors' to the R-package **pedometrics**.

Author(s)

Alessandro Samuel-Rosa `<alessandrosamuelrosa@gmail.com>`

References

Deepayan Sarkar (2008). *Lattice: Multivariate Data Visualization with R.* Springer, New York. ISBN 978-0-387-75968-5.

Roger D. Peng (2008). *A method for visualizing multivariate time series data.* Journal of Statistical Software. v. 25 (Code Snippet), p. 1-17.

Roger D. Peng (2012). *mvtsplot: Multivariate Time Series Plot.* R package version 1.0-1. https://CRAN.R-project.org/package=mvtsplot.

A. Samuel-Rosa, G. B. M. Heuvelink, G. de Mattos Vasques, and L. H. C. dos Anjos, Do more detailed environmental covariates deliver more accurate soil maps?, *Geoderma*, vol. 243–244, pp. 214–227, May 2015, doi: 10.1016/j.geoderma.2014.12.017.

See Also

`lattice::xyplot()` `lattice::levelplot()`

Examples

```
if (all(require(grDevices), require(grid))) {
  # This example follows the discussion in section "Details"
  # Note that the data.frame is created manually
  id <- c(1:8)
  design <- data.frame(a = c(0, 0, 1, 0, 1, 0, 1, 1),
                       b = c(0, 0, 0, 1, 0, 1, 1, 1),
                       c = c(0, 1, 0, 0, 1, 1, 0, 1))
  adj_r2 <- c(0.87, 0.74, 0.81, 0.85, 0.54, 0.86, 0.90, 0.89)
  obj <- cbind(id, design, adj_r2)
  p <- plotModelSeries(obj, grid = c(2:4), line = "adj_r2", ind = 1,
              color = c("lightyellow", "palegreen"),
              main = "Model Series Plot")
}
```

---

rowMinCpp *Return the minimum value in each row of a numeric matrix*

Description

This function returns the minimum value in each row of a numeric matrix.

Usage

```
rowMinCpp(x)
```

Arguments

x	Numeric matrix with two or more rows and/or columns.

Details

This function is implemented in C++ to speed-up the computation time for large matrices.

Value

A numeric vector with the minimum value of each row if the matrix.

Author(s)

Alessandro Samuel-Rosa <`alessandrosamuelrosa@gmail.com`>

See Also

`rowMins()` in `https://cran.r-project.org/package=matrixStats`.

Examples

```
x <- matrix(rnorm(20), nrow = 5)
rowMinCpp(x)
```

`skewness` *Moment coefficient of skewness*

Description

Compute the moment coefficient of skewness of a continuous, possibly non-normal variable.

Usage

```
skewness(x)
```

Arguments

`x`	Numeric vector, the values of the variable of interest.

Value

A numerical value: the moment coefficient of skewness of `x`.

Author(s)

Alessandro Samuel-Rosa <`alessandrosamuelrosa@gmail.com`>

References

B. S. Everitt, *The Cambridge Dictionary of Statistics*, 3rd ed. Cambridge: Cambridge University Press, 2006, p. 432.

D. N. Joanes and C. A. Gill, Comparing measures of sample skewness and kurtosis, *J Royal Statistical Soc D*, vol. 47, no. 1, pp. 183–189, Mar. 1998, doi: 10.1111/1467-9884.00122.

H. Cramér, *Mathematical Methods of Statistics*. Princeton: Princeton University Press, 1946, p. 575.

Examples

```
x <- rlnorm(10)
skw <- skewness(x)
```

---

statsModelSeries *Obtain performance statistics of a series of linear models*

Description

Compute several statistics measuring the performance of a series of linear models built using `buildModelSeries()`, with an option to rank the models based on one of the returned performance statistics.

Usage

```
statsModelSeries(model, design.info, arrange.by, digits)

statsMS(model, design.info, arrange.by, digits)
```

Arguments

`model`	A list of linear models returned by `buildModelSeries()`.
`design.info`	Extra information about the linear models in the series.
`arrange.by`	Character string defining if the table with the performance statistics of the linear models should be arranged, and which column should be used. Available options are `"candidates"`, `"df"`, `"aic"`, `"rmse"`, `"nrmse"`, `"r2"`, `"adj_r2"`, and `"ADJ_r2"`. Descending order is used by default and cannot be changed in the current implementation. See 'Value' for more information.
`digits`	Integer or vector with six integers indicating the number of decimal places to be used to round the performance statistics. If a vector is passed to the function, the number of decimal places should be in the following order: `c("aic", "rmse", "nrmse", "r2", "adj_r2", "ADJ_r2")`.

Details

This function was devised to deal with a list of linear models generated by the function `buildModelSeries()`. The main objective is to compare several linear models using several performance statistics. Such statistics can then be used to rank the linear models and identify, for example, the best performing model, given the selected performance statistics.

An important feature of `buildModelSeries()` is that it uses the information about the initial number of candidate predictor variables offered to the build the model to calculate penalized or adjusted measures of model performance. Such information is recorded as an attribute of the final model selected by `buildModelSeries()`. This feature was included in `statsModelSeries()` because data-driven variable selection results biased linear models (too optimistic), and the effective number of degrees of freedom is close to the number of candidate predictor variables initially offered to the model (Harrell, 2001).

Value

A data frame with several performance statistics:

id Identification of the model.

candidates Number of candidate predictor variables initially offered to the model.

df Number of degrees of freedom of the final selected model.

aic Akaike's Information Criterion (AIC). Obtained using `extractAIC`.

rmse Root-mean squared error, calculated based on the number of candidate predictor variables initially offered to the model.

nrmse Normalized Root-mean squared error, calculated as the ratio between the RMSE and the standard deviation of the observed values of the dependent variable.

r2 Multiple coefficient of determination.

adj_r2 Adjusted multiple coefficient of determination.

ADJ_r2 Adjusted multiple coefficient of determination. Calculations are done based on the number of candidate predictor variables initially offered to the model.

TODO

- Include other performance statistics such as: PRESS, BIC, Mallow's Cp, max(VIF);
- Add option to select which performance statistics should be returned.

Author(s)

Alessandro Samuel-Rosa <`alessandrosamuelrosa@gmail.com`>

References

Harrell, F. E. (2001) *Regression modelling strategies: with applications to linear models, logistic regression, and survival analysis.* First edition. New York: Springer.

Venables, W. N. and Ripley, B. D. (2002) *Modern applied statistics with S.* Fourth edition. New York: Springer.

A. Samuel-Rosa, G. B. M. Heuvelink, G. de Mattos Vasques, and L. H. C. dos Anjos, Do more detailed environmental covariates deliver more accurate soil maps?, *Geoderma*, vol. 243–244, pp. 214–227, May 2015, doi: 10.1016/j.geoderma.2014.12.017.

See Also

`buildModelSeries()`, `plotModelSeries()`

Examples

```
if (interactive()) {
  # based on the second example of function MASS:stepAIC()
  require(MASS)
  cpus1 <- cpus
  for(v in names(cpus)[2:7])
```

```
    cpus1[[v]] <- cut(cpus[[v]], unique(quantile(cpus[[v]])),
                      include.lowest = TRUE)
  cpus0 <- cpus1[, 2:8]  # excludes names, authors' predictions
  cpus.samp <- sample(1:209, 100)
  cpus.form <- list(formula(log10(perf) ~ syct + mmin + mmax + cach + chmin +
                    chmax + perf),
                    formula(log10(perf) ~ syct + mmin + cach + chmin + chmax),
                    formula(log10(perf) ~ mmax + cach + chmin + chmax + perf))
  data <- cpus1[cpus.samp,2:8]
  cpus.ms <- buildModelSeries(cpus.form, data, vif = TRUE, aic = TRUE)
  cpus.des <- data.frame(a = c(0, 1, 0), b = c(1, 0, 1), c = c(1, 1, 0))
  stats <- statsModelSeries(cpus.ms, design.info = cpus.des, arrange.by = "aic")
}
```

stepVIF — *Variable selection using the (generalized) variance-inflation factor (VIF)*

Description

This function takes a linear model and selects the subset of predictor variables that meet a user-specific collinearity threshold measured by the (generalized) variance-inflation factor (VIF).

Usage

```
stepVIF(model, threshold = 10, verbose = FALSE)
```

Arguments

`model`	Linear model (object of class 'lm') containing collinear predictor variables.
`threshold`	Positive number defining the maximum allowed VIF. Defaults to `threshold = 10`.
`verbose`	Logical indicating if iteration results should be printed. Defaults to `verbose = FALSE`.

Details

`stepVIF` starts computing the VIF of all predictor variables in the linear model. If the linear model contains categorical predictor variables, generalized variance-inflation factors (GVIF) (Fox and Monette, 1992), are calculated instead using `car::vif()`. GVIF is interpretable as the inflation in size of the confidence ellipse or ellipsoid for the coefficients of the predictor variable in comparison with what would be obtained for orthogonal, uncorrelated data. Since categorical predictors have more than one degree of freedom, *df*, the confidence ellipsoid will have *df* dimensions, and GVIF will need to be adjusted so that it can be comparable across predictor variables. The adjustment is made using the following equation:

$GVIF^{1/(2\times df)}$

The next step consists of evaluating if any of the predictor variables has a (G)VIF larger than the specified threshold, the function default being `threshold = 10`. For, GVIF^(1/(2*df)), the threshold will be `sqrt(threshold)`.

If there is only one predictor variable that does not meet the VIF threshold, it is automatically removed from the model and no further processing occurs. When there are two or more predictor variables that do not meet the (G)VIF threshold, `stepVIF()` fits a linear model between each of them and the dependent variable. The predictor variable with the lowest adjusted coefficient of determination is dropped from the model and new coefficients are calculated, resulting in a new linear model.

This process lasts until all predictor variables included in the new model meet the (G)VIF threshold.

Nothing is done if all predictor variables have a (G)VIF value lower that the threshold, and `stepVIF()` returns the original linear model.

Value

A linear model (object of class 'lm') with low collinearity.

Dependencies

The **car** package, provider of functions to accompany Fox and Weisberg's An R Companion to Applied Regression, is required for `plotHist()` to work. The development version of the **car** package is available on `https://r-forge.r-project.org/projects/car/` while its old versions are available on the CRAN archive at `https://cran.r-project.org/src/contrib/Archive/car/`.

Note

More on the use of GVIF to measure the collinearity in linear models containing categorical predictor variables can be found on StackExchange.

Author(s)

Alessandro Samuel-Rosa <`alessandrosamuelrosa@gmail.com`>

References

Fox, J. and Monette, G. (1992) Generalized collinearity diagnostics. *JASA*, 87, 178–183.

Fox, J. (2008) *Applied Regression Analysis and Generalized Linear Models*, Second Edition. Sage.

Fox, J. and Weisberg, S. (2011) *An R Companion to Applied Regression*, Second Edition. Thousand Oaks: Sage.

Hair, J. F., Black, B., Babin, B. and Anderson, R. E. (2010) *Multivariate data analysis*. New Jersey: Pearson Prentice Hall.

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S*. Fourth edition. Springer.

A. Samuel-Rosa, G. B. M. Heuvelink, G. de Mattos Vasques, and L. H. C. dos Anjos, Do more detailed environmental covariates deliver more accurate soil maps?, *Geoderma*, vol. 243–244, pp. 214–227, May 2015, doi: 10.1016/j.geoderma.2014.12.017.

See Also

MASS::stepAIC()

Examples

```
if (require(car)) {
  fit <- lm(prestige ~ income + education + type, data = Duncan)
  fit <- stepVIF(fit, threshold = 10, verbose = TRUE)
}
```

variogramBins *Variogram binning*

Description

Computation of bins for sample (experimental) variograms.

Usage

```
variogramBins(
  coords,
  n.lags = 7,
  type = "exp",
  cutoff = 0.5,
  base = 2,
  zero = 0.001,
  count = "pairs"
)

vgmLags(
  coords,
  n.lags = 7,
  type = "exp",
  cutoff = 0.5,
  base = 2,
  zero = 0.001,
  count = "pairs"
)
```

Arguments

coords	Data frame or matrix with the projected x- and y-coordinates.
n.lags	Integer value defining the number of variogram bins (distance classes) that should be computed. Defaults to n = 7.
type	Character value defining the type of variogram bins that should be computed, with options "equi" (equidistant, equal-width) and "exp" (exponential, exponentially growing). Defaults to type = "exp".

cutoff	Numeric value defining the fraction of the diagonal of the rectangle that spans the data (bounding box) that should be used to set the maximum distance up to which variogram bins should be computed. Defaults to cutoff = 0.5, i.e. half the diagonal of the bounding box.
base	Numeric value defining the base of the exponential expression used to create exponentially growing variogram bins. Used only when type = "exp". Defaults to base = 2, i.e. the width of the rightmost bin is equal to half the diagonal of cutoff, and so on.
zero	Numeric value setting the minimum pair-wise separation distance that should be used to compute the variogram bins. Defaults to zero = 0.0001.
count	Should the number of points ("points") or point-pairs ("pairs") per variogram bin be computed? Defaults to count = "pairs".

Value

Vector of numeric values with the lower and upper boundaries of the variogram bins. The number of points or point-pairs per variogram bin is returned as an attribute.

Dependencies

The **SpatialTools** package, provider of tools for spatial data analysis in R, is required for variogramBins() to work. The development version of the **SpatialTools** package is available on https://github.com/jfrench/SpatialTools while its old versions are available on the CRAN archive at https://cran.r-project.org/src/contrib/Archive/SpatialTools/.

Reverse dependencies

The **spsann** package, provider of methods for the optimization of sample configurations using spatial simulated annealing in R, requires variogramBins() for some of its functions to work. The development version of the **spsann** package is available on https://github.com/Laboratorio-de-Pedometria/spsann-package.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

Truong, P. N.; Heuvelink, G. B. M.; Gosling, J. P. Web-based tool for expert elicitation of the variogram. *Computers and Geosciences*. v. 51, p. 390-399, 2013.

Examples

```
if (all(c(require(SpatialTools), require(sp)))) {
  data(meuse, package = "sp")
  lags_points <- variogramBins(coords = meuse[, 1:2], count = "points")
  lags_pairs <- variogramBins(coords = meuse[, 1:2], count = "pairs")
}
```

`variogramGuess` *Guess the parameters of a spatial covariance function*

Description

Guess the parameters of the spatial covariance function of a random, regionalized variable. A guess of such parameters is required to start the fitting functions of many geostatistical packages such as **gstat**, **geoR**, and **georob**.

Usage

```
variogramGuess(
  z,
  coords,
  lags,
  cutoff = 0.5,
  method = "a",
  min.npairs = 30,
  model = "matern",
  nu = 0.5,
  estimator = "qn",
  plotit = FALSE
)

vgmICP(
  z,
  coords,
  lags,
  cutoff = 0.5,
  method = "a",
  min.npairs = 30,
  model = "matern",
  nu = 0.5,
  estimator = "qn",
  plotit = FALSE
)
```

Arguments

`z`	Numeric vector with the values of the regionalized variable for which the values for the spatial covariance parameters should be guessed.
`coords`	Data frame or matrix with the projected x- and y-coordinates.
`lags`	Numeric scalar defining the width of the variogram bins or a numeric vector with the lower and upper bounds of the variogram bins. If missing, the variogram bins are computed using `variogramBins()`. See 'Details' for more information.

`cutoff`	Numeric value defining the fraction of the diagonal of the rectangle that spans the data (bounding box) that should be used to set the maximum distance up to which variogram bins should be computed. Defaults to `cutoff = 0.5`, i.e. half the diagonal of the bounding box.
`method`	Character keyword defining the method used for guessing the spatial covariance parameters of the regionalized variable. Defaults to `method = "a"`. See 'Details' for more information.
`min.npairs`	Positive integer defining the minimum number of point-pairs required so that a variogram bin is used to guessing the spatial covariance parameters of the of the regionalized variable. Defaults to `min.npairs = 30`.
`model`	Character keyword defining the spatial covariance function that will be fitted to the data of the regionalized variable. Currently, most of the basic spatial covariance function are accepted. See `geoR::cov.spatial()` for more information. Defaults to `model = "matern"`.
`nu`	Numerical value for the additional smoothness parameter ν of the spatial covariance function of the regionalized variable. See `RandomFields::RMmodel()` and argument `kappa` of `geoR::cov.spatial()` for more information.
`estimator`	Character keyword defining the estimator for computing the sample (experimental) variogram of the regionalized variable, with options `"qn"` (default), `"mad"`, `"matheron"`, and `"ch"`. See `georob::sample.variogram()` for more details.
`plotit`	Should the guessed spatial covariance parameters be plotted along with the sample (experimental) variogram of the regionalized variable? Defaults to `plotit = FALSE`.

Details

There are five methods two guess the covariance parameters. Two of them, `"a"` and `"c"`, rely on a sample variogram with exponentially growing variogram bins, while the other three, `"b"`, `"d"`, and `"e"`, use equal-width variogram bins (see `variogramBins()`). All of them are heuristic.

Method `"a"` was developed in-house and is the most elaborated of them, specially for guessing the nugget variance.

Method `"b"` was proposed by doi:10.1016/00983004(95)00095XJian et al. (1996) and is implemented in SAS/STAT(R) 9.22.

Method `"c"` is implemented in the **automap**-package and was developed by doi:10.1016/j.cageo.2008.10.011Hiemstra et al. (2009).

Method `"d"` was developed by doi:10.1007/s1100401294341Desassis & Renard (2012).

Method `"e"` was proposed by Larrondo et al. (2003) `http://www.ccgalberta.com/ccgresources/report05/2003-122-varfit.pdf` and is implemented in the VARFIT module of GSLIB `http://www.gslib.com/`.

Value

A vector of numerical values, the guesses for the spatial covariance parameters of the regionalized variable:

- nugget

- partial sill
- range

Dependencies

The **georob** package, provider of functions for the robust geostatistical analysis of spatial data in R, is required for `variogramGuess()` to work. The old versions of the **georob** package are available on the CRAN archive at https://cran.r-project.org/src/contrib/Archive/georob/.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

Desassis, N. & Renard, D. Automatic variogram modelling by iterative least squares: univariate and multivariate cases. *Mathematical Geosciences*. Springer Science + Business Media, v. 45, p. 453-470, 2012.

Hiemstra, P. H.; Pebesma, E. J.; Twenhöfel, C. J. & Heuvelink, G. B. Real-time automatic interpolation of ambient gamma dose rates from the Dutch radioactivity monitoring network. *Computers & Geosciences*. Elsevier BV, v. 35, p. 1711-1721, 2009.

Jian, X.; Olea, R. A. & Yu, Y.-S. Semivariogram modelling by weighted least squares. *Computers & Geosciences*. Elsevier BV, v. 22, p. 387-397, 1996.

Larrondo, P. F.; Neufeld, C. T. & Deutsch, C. V. *VARFIT: a program for semi-automatic variogram modelling*. Edmonton: Department of Civil and Environmental Engineering, University of Alberta, p. 17, 2003.

See Also

`variogramBins()`

Examples

```
if (all(c(require(sp), require(georob)))) {
  data(meuse, package = "sp")
  icp <- variogramGuess(z = log(meuse$copper), coords = meuse[, 1:2])
}
```

---

vgmSCV *Spatially correlated variance (SCV)*

Description

Compute the proportion of the variance that is spatially correlated.

Usage

```
vgmSCV(obj, digits = 4)

## S3 method for class 'variomodel'
vgmSCV(obj, digits = 4)

## S3 method for class 'variogramModel'
vgmSCV(obj, digits = 4)

## S3 method for class 'georob'
vgmSCV(obj, digits = 4)
```

Arguments

`obj`	Variogram model fitted with available function in geostatistical packages such as **gstat**, **geoR**, and **georob**.
`digits`	Integer indicating the number of decimal places to be used.

Value

Numeric value indicating the proportion of the variance that is spatially correlated.

Author(s)

Alessandro Samuel-Rosa <`alessandrosamuelrosa@gmail.com`>

See Also

`variogramBins()`

Examples

```
if (all(c(require(gstat), require(sp)))) {
  library(gstat)
  library(sp)
  data(meuse)
  sp::coordinates(meuse) <- ~ x + y
  vgm1 <- gstat::variogram(log(zinc) ~ 1, meuse)
  fit <- gstat::fit.variogram(vgm1, gstat::vgm(1, "Sph", 300, 1))
  res <- vgmSCV(fit)
}
```

Index