Package 'pedometrics'

February 9, 2020

Version 0.7.0

Date 2020-02-08

Title Miscellaneous Pedometric Tools

Description An R implementation of some useful tools employed in the field of pedometrics.

URL https://github.com/samuel-rosa/pedometrics

BugReports https://github.com/samuel-rosa/pedometrics/issues

Depends R (>= 3.2.0)

Imports lattice, latticeExtra, Rcpp (>= 0.12.0)

LinkingTo Rcpp

Suggests car, fields, geoR, georob, grDevices, grid, gstat, knitr, MASS, methods, moments, pbapply, plyr, randomForest, RColorBrewer, sp, SpatialTools, spsurvey, xtable

License GPL (>= 2)

Encoding UTF-8

Repository CRAN

RoxygenNote 7.0.2

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>), Tony Olsen [ctb], Tom Kincaid [ctb], 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 2020-02-09 13:00:02 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.

Details

Details will be added later.

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_d

Examples

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

---

bbox2sp *Create Spatial object from a bounding box*

Description

This function takes the bounding box of a Spatial* object and creates 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. Available options are "SpatialPoints", "SpatialPointsDataFrame", "SpatialPolygons" and "SpatialPolygonsDataFrame".
keep.crs	Logical for assigning the same coordinate reference system to the resulting Spatial* object.

Value

An object of class SpatialPoints* or SpatialPolygons*.

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

```
require(sp)
data(meuse)
coordinates(meuse) <- ~ x + y
bbox2sp(meuse, keep.crs = FALSE)
```

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

Description

This function allows building a series of linear models (`lm`) using one or more automated variable selection implemented in function `stepVIF` and `stepAIC`.

Usage

```
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 Information Criterion (AIC). Defaults to aic = FALSE.
aic.direction	Character string setting the direction of variable selection when using AIC. Available options are "both", "forward", and "backward". Defaults to aic.direction = "both".
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 the function stepAIC.

Details

This function 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 stepAIC or both. stepVIF is a backward variable selection procedure, while stepAIC supports backward, forward, and bidirectional variable selection. For more information about these functions, please visit their respective help pages.

An important feature of buildMS 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 builtMS, this can be done using statsMS.

Some important details should be clear when using buildMS:

1. this function was originally devised to deal with a list of formulas, but can also be used with a single formula;
2. in the current implementation, stepVIF runs before stepAIC;
3. function arguments imported from 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 stepAIC and stepVIF are run.

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.

See Also

stepAIC, stepVIF, statsMS.

Examples

```
## Not run:
# based on the second example of function stepAIC
require(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 <- buildMS(cpus.form, data, vif = TRUE, aic = TRUE)

## End(Not run)
```

---

cdfPlot *Plot estimated cumulative distribution function with confidence limits*

Description

This function is a modified version of `cdf.plot()` of **spsurvey**-package including new argument options.

Usage

```
cdfPlot(
  obj,
  ind,
  units.cdf = "percent",
```

```
  type.plot = "s",
  type.cdf = "continuous",
  logx = "",
  xlbl = NULL,
  ylbl = "Percent",
  ylbl.r = NULL,
  figlab = NULL,
  legloc = "BR",
  confcut = 5,
  show.conflev = TRUE,
  conflev = 95,
  show.param = TRUE,
  round = 0,
  col.param = "black",
  ...
)
```

Arguments

obj	Object with the estimated CDF. The resulting object of `cont.analysis()` of **spsurvey**-package.
ind	Indicator variable. The name of the variable as displayed in the resulting object of `cont.analysis()`.
units.cdf	Indicator for the type of units in which the CDF is plotted, where "percent" means the plot is in terms of percent of the population, and "units" means the plot is in terms of units of the population. Defaults to `units.cdf = "percent"`.
type.plot	Type of plot. Desired type of plot to be produced, with options `type.plot = "l"`, for 'line', and `type.plot = "s"` for 'stair'. See 'Details'. Defaults to `type.plot = "s"`.
type.cdf	Character string consisting of the value "continuous" or "ordinal" that controls the type of CDF plot for each indicator. Defaults to `type.cdf = "continuous"`.
logx	Character string consisting of the value `""` or `"x"` that controls whether the x axis uses the original scale (`""`) or the base 10 logarithmic scale (`"x"`). Defaults to `logx = ""`.
xlbl	Character string providing the x-axis label. If this argument equals `NULL`, then the indicator name is used as the label. Defaults to `xlbl = NULL`.
ylbl	Character string providing the the y-axis label. Defaults to `ylbl = "Percent"`.
ylbl.r	Character string providing the label for the right side y-axis, where `ylbl.r = NULL` means a label is not created, and `ylbl.r = "Same"` means the label is the same as the left side label (i.e., argument `ylbl`). Defaults to `ylbl.r = NULL`.
figlab	Character string providing the plot title. Defaults to `figlab = NULL`.
legloc	Indicator for location of the plot legend, where `legloc = "BR"` means bottom right, `legloc = "BL"` means bottom left, `legloc = "TR"` means top right, and `legloc = "TL"` means top left. Defaults to `legloc = "BR"`.

confcut	Numeric value that controls plotting confidence limits at the CDF extremes. Confidence limits for CDF values (percent scale) less than confcut or greater than 100 minus confcut are not plotted. A value of zero means confidence limits are plotted for the complete range of the CDF. Defaults to confcut = 5.
show.conflev	Logical for showing the confidence limits of the CDF. Defaults to show.conflev = TRUE.
conflev	Numeric value of the confidence level used for confidence limits. Defaults to conflev = 95.
show.param	Logical for showing the parameters of the CDF. Available parameters are the mean, the median, and a percentile defined by the argument conflev. The legend displays de actual values of all three parameters, including the standard deviation of the mean. The percentile value is calculated using spsurvey::interp.cdf().
round	Numeric to set the rounding level of the parameters of the CDF.
col.param	Color of the lines showing the parameters of the CDF. Defaults to col.param = "black".
...	Additional arguments passed to plot(). See 'Details'.

Details

Parameter type.plot is used only when type.cdf = "Continuous".

Care should be taken with possible conflicts between the arguments of the original function cdf.plot and those passed to plot() using .... The existence of conflicts between these two functions was one of the reasons for creating this new implementation.

Value

A plot of the estimated cumulative distribution function with confidence limits.

Note

Most of the source code that constitutes this function was originally published in the **spsurvey**-package, version 2.6 (2013-09-20). The authors were asked to include a few new functionalities, but did not seem to be interested in doing so, since no reply was obtained. This implementation is a way of including such functionalities. When using this function, credit should be given to the authors of the original implementation in the **spsurvey**-package.

Author(s)

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

References

Brus, D. J., Kempen, B. and Heuvelink, G. B. M. (2011). Sampling for validation of digital soil maps. *European Journal of Soil Science*, v. 62, p. 394-407.

Diaz-Ramos, S., D.L. Stevens, Jr., and A.R. Olsen. (1996). *EMAP Statistical Methods Manual*. EPA/620/R-96/XXX. Corvallis, OR: U.S. Environmental Protection Agency, Office of Research and Development, National Health Effects and Environmental Research Laboratory, Western Ecology Division.

Kincaid, T. M. and Olsen, A. R. (2013) *spsurvey: Spatial Survey Design and Analysis*. R package version 2.6. URL: http://www.epa.gov/nheerl/arm/.

See Also

cdf.plot.

Examples

```
## Not run:
if (require(spsurvey)) {
## Estimate the CDF
my.cdf <- spsurvey::cont.analysis(spsurvey.obj = my.spsurvey)

## See indicator levels in the resulting object
levels(my.cdf$Pct$Indicator)

## Plot CDF
cdfPlot(obj = my.cdf, ind = "dz", figlab = "",
   xlbl = "Difference (m)", xlim = c(-30, 10), type.plot = "s")
}

## End(Not run)
```

---

cdfStats — *Descriptive statistics of the cumulative distribution function of a continuous variable*

Description

This function returns summary statistics of the cumulative distribution function of a continuous variable estimated with **spsurvey**-package.

Usage

```
cdfStats(obj, ind, all = TRUE)
```

Arguments

obj	Object containing the estimated cumulative distribution function of the continuous variable. The resulting object of `cont.analysis()` of **spsurvey**-package.
ind	Indicator variable. The name of the continuous variable as displayed in the resulting object of `cont.analysis()`.

`all` Summary statistics to be returned. The default option (`all = TRUE`) returns all summary statistics available. If `all = FALSE`, then only estimated population mean and standard deviation are returned. See 'Details'.

Details

The function `cont.analysis()` of **spsurvey**-package estimates the population total, mean, variance, and standard deviation of a continuous variable. It also estimates the standard error and confidence bounds of these population estimates. In some cases it may be interesting to see all estimates, for which one uses `all = TRUE`. However, in other circumstances there might be interest only in taking a look at the estimated population mean and standard deviation. Then the argument `all` has to be set to `FALSE`.

Value

A `data.frame` containing summary statistics of the cumulative distribution function of a continuous variable.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

Kincaid, T. M. and Olsen, A. R. (2013). spsurvey: Spatial Survey Design and Analysis. R package version 2.6. URL: http://www.epa.gov/nheerl/arm/.

See Also

`cont.analysis`.

Examples

```
## Not run:
if (require(spsurvey)) {
## Estimate the CDF
my.cdf <- spsurvey::cont.analysis(spsurvey.obj = my.spsurvey)

## See indicator levels in the resulting object
levels(my.cdf$Pct$Indicator)

## Return all summary statistics of indicator variable 'dx'
cdfStats(my.cdf, "dx", all = TRUE)
}

## End(Not run)
```

cdfTable *Table with descriptive statistics of an estimated cumulative distribution function*

Description

This function returns a table containing the descriptive statistics of the cumulative distribution function of a set of continuous variables. TeX code is printed to copy and paste in a document.

Usage

```
cdfTable(x, type = "xy", rounding = 0, tex = FALSE, data.frame = FALSE)
```

Arguments

x	Object with the estimated cumulative distribution function of the set of continuous variables. The resulting object of `cont.analysis()` of **spsurvey**-package.
type	Type of data under analysis. Defaults to `type = "xy"`. See 'Details'.
rounding	Rounding level of the data in the output table. Defaults to `rounding = 0`.
tex	Logical for creating TeX code. Defaults to `tex = FALSE`.
data.frame	Logical for returning a data.frame object. Defaults to `data.frame = FALSE`.

Details

Summary statistics included in the table (estimated population mean and standard deviation) are obtained from the resulting object of `cont.analysis()` by internally using the function `cdfStats()`.

There are two types of data that can be submitted to function `cdfTable()`. The first (`type = "xy"`) is composed by two instances ('x' and 'y') and is produced during horizontal (positional) validation exercises (validation in the geographic space). Thus, 'x' and 'y' represent, respectively, the horizontal displacement (error) in 'x' and 'y' coordinates.

The second type of data (`type = "z"`) is composed by only one instance ('z') and is generated by vertical validation exercises (validation in the attribute space). Thus, 'z' represents the vertical displacement (error) of the attribute 'z' being measured.

Value

Returned value depends on how arguments `type` and `tex` are set.

`list("type")` If `type = "xy"`, then the function returns a table with estimated population mean and standard deviation of error statistics for 'x' and 'y' coordinates. These error statistics include the mean error, mean absolute error, and mean square error. It also returns the estimated mean and mean square error vector (module), and the estimated mean azimuth. The number of ground control points used to make the estimates is printed by default.

If `type = "z"`, then the function returns a table with estimated population mean and standard deviation of error statistics for 'z', the attribute under analysis.

These error statistics include the mean error, mean absolute error, and mean square error. The number of ground control points used to make the estimates is printed by default.

`list("tex")` If `tex = TRUE`, them the function prints the TeX code for the table defined by the argument `type`. Otherwise the TeX code is not generated.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

Kincaid, T. M. and Olsen, A. R. (2013). spsurvey: Spatial Survey Design and Analysis. R package version 2.6. URL: http://www.epa.gov/nheerl/arm/.

See Also

`cdfStats`, `cont.analysis`.

Examples

```
## Not run:
if (require(spsurvey)) {
## Estimate the CDF
my.cdf <- spsurvey::cont.analysis(spsurvey.obj = my.spsurvey)

## Print table and TeX code
cdfTable(my.cdf)
}

## End(Not run)
```

---

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.

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

```
## Not run:
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

## End(Not run)
```

---

cont2cat *Categorize/stratify continuous variable(s)*

Description

Create break points, compute strata proportions, and stratify continuous 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 continuous variable(s) to be categorized/stratified.
breaks	Vector or list containing the lower and upper limits that should be used to break the continuous variable(s) into categories. See 'Details' for more information.
integer	Logical value indicating if the categorical variable(s) 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 strata that should be created.
type	Character value indicating the type of strata that should be used, with options "area", for equal-area, and "range", for equal-range strata. Defaults to type = "area".
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.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

See Also

cut2

Examples

```
## 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)
x

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

## 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` *Prepare object for argument* `design` *of* `spsurvey.analysis()`

Description

This function returns an object to feed the argument `design` when creating an object of class `spsurvey.analysis`.

Usage

```
coordenadas(x)
```

Arguments

`x`	Object of class `SpatialPointsDataFrame` from which site ID and XY coordinates are to be returned.

Details

The argument `design` used to create object of class `spsurvey.analysis` requires a series of inputs. However, it can be fed with data about site ID and coordinates. `coordenadas()` returns a data frame that provides this information, assuming that all other design variables are provided manually in the arguments list.

Value

An object of class `data.frame` containing three columns with names `siteID`, `xcoord`, and `ycoord`.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

Kincaid, T. M. and Olsen, A. R. (2013). *spsurvey: Spatial Survey Design and Analysis*. R package version 2.6. URL: https://www.epa.gov/nheerl/arm/.

See Also

gcpDiff, cont.analysis.

Examples

```
## Not run:
if (require(spsurvey)) {
  ## Create an spsurvey.analysis object
  my.spsurvey <-
    spsurvey::spsurvey.analysis(
      design = coordenadas(my.data),
      data.cont = delta(ref.data, my.data),
      popcorrect = TRUE, pcfsize = length(my.data$id),
      support = rep(1, length(my.data$id)),
      wgt = rep(1, length(my.data$id)), vartype = "SRS")
}

## End(Not run)
```

---

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.

Note

The original code is available at http://sas-and-r.blogspot.nl/, 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**.

The function `bigtabulate` used to compute the chi-squared test is the main bottleneck in the current version of `cramer`. Ideally it will be implemented in C++.

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.

See Also

assocstats

Examples

```
## Not run:
data <- read.csv("http://www.math.smith.edu/r/data/help.csv")
data <- data[, c("female", "homeless", "racegrp")]
str(data)
test <- cramer(data)
test

## End(Not run)
```

`gcpDiff` *Difference on xyz coordinates between ground control points*

Description

This function estimates the difference, absolute difference, and squared difference on x, y and z coordinates of two sets of ground control points (GCP). It also estimates the module (difference vector), its square and azimuth. The result is a data frame ready to be used to define a object of class `spsurvey.object`.

Usage

```
gcpDiff(measured, predicted, type = "xy", aggregate = FALSE, rounding = 0)
```

Arguments

measured	Object of class SpatialPointsDataFrame with the reference GCP. A column named 'siteID' giving case names is mandatory. See 'Details', item 'Type of data'.
predicted	An object of class SpatialPointsDataFrame with the point data being validated. A column named 'siteID' giving case names is mandatory. See 'Details', item 'Type of data'.
type	Type of data under analysis. Defaults to type = "xy". 'Details', item 'Type of data'.
aggregate	Logical for aggregating the data when it comes from cluster sampling. Used only when type = "z". Defaults to aggregate = FALSE. See 'Details', item 'Data aggregation'.
rounding	Rounding level of the data in the output data frame.

Details

Type of data: Two types of validation data that can be submitted to function gcpDiff(): those coming from horizontal (positional) validation exercises (type = "xy"), and those coming from vertical validation exercises (type = "z").

Horizontal (positional) validation exercises compare the position of measured point data with the position of predicted point data. Horizontal displacement (error) is measured in both 'x' and 'y' coordinates, and is used to calculate the error vector (module) and its azimuth. Both objects measured and predicted used with function gcpDiff() must be of class SpatialPointsDataFrame. They must have at least one column named 'siteID' giving the identification of every case. Matching of case IDs is mandatory. Other columns are discarded.

Vertical validation exercises are interested in comparing the measured value of a variable at a given location with that predicted by some model. In this case, error statistics are calculated only for the the vertical displacement (error) in the 'z' coordinate. Both objects measured and predicted used with function gcpDiff() must be of class SpatialPointsDataFrame. They also must have a column named 'siteID' giving the identification of every case. Again, matching of case IDs is mandatory. However, both objects must have a column named 'z' which contains the values of the 'z' coordinate. Other columns are discarded.

Data aggregation: Validation is sometimes performed using cluster or transect sampling. Before estimation of error statistics, the data needs to be aggregated by cluster or transect. The function gcpDiff() aggregates validation data of type = "z" calculating the mean value per cluster. Thus, aggregation can only be properly done if the 'siteID' column of both objects measured and predicted provides the identification of clusters. Setting aggregate = TRUE will return aggregated estimates of error statistics. If the data has been aggregated beforehand, the parameter aggregate can be set to FALSE.

Case matching: There are circumstances in which the number of cases in the object measured is larger than that in the object predicted. The function gcpDiff() compares the number of

cases in both objects and automatically drops those cases of object `measured` that do not match the cases of object `predicted`. However, case matching can only be done if case IDs are exactly the same for both objects. Otherwise, estimated error statistics will have no meaning at all.

Value

An object of class `data.frame` ready to be used to feed the argument `data.cont` when creating a `spsurvey.analysis` object.

Note

Data of `type = "xy"` cannot be submitted to cluster aggregation in the present version.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

Kincaid, T. M. and Olsen, A. R. (2013). spsurvey: Spatial Survey Design and Analysis. R package version 2.6. URL: http://www.epa.gov/nheerl/arm/.

See Also

coordenadas, gcpVector, spsurvey.analysis.

Examples

```
## Not run:
if (require(spsurvey)) {
## Create an spsurvey.analysis object
my.spsurvey <-
  spsurvey.analysis(design = coordenadas(my.data),
                    data.cont = delta(ref.data, my.data),
                    popcorrect = TRUE, pcfsize = length(my.data$id),
                    support = rep(1, length(my.data$id)),
                    wgt = rep(1, length(my.data$id)), vartype = "SRS")
}

## End(Not run)
```

gcpVector *Calculate module and azimuth*

Description

This function calculates the module and azimuth of the difference on x and y coordinates between two sets of ground control points (GCP).

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.

Details

This function is suited to perform calculations for topographical coordinates only. The origin is set in the y coordinate, and rotation performed clockwise.

Value

An object of the class `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 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

See Also

LoadData, gcpDiff

Examples

```
## Not run:
gcpVector(dx = rnorm(3, 5, 10), dy = rnorm(3, 5, 10))

## End(Not run)
```

---

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

is.numeric, is.integer, 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
```

---

optimRandomForest *Optimum number of iterations to de-bias a random forest regression*

Description

Compute the optimum number of iterations needed to de-bias a random forest regression.

Usage

```
optimRandomForest(
  x,
  y,
  niter = 10,
  nruns = 100,
  ntree = 500,
  ntrain = 2/3,
  nodesize = 5,
  mtry = max(floor(ncol(x)/3), 1),
  profile = TRUE,
  progress = TRUE
)
```

Arguments

x	Data frame or matrix of covariates (predictor variables).
y	Numeric vector with the response variable.
niter	Number of iterations. Defaults to niter = 10.
nruns	Number of simulations to be used in each iteration. Defaults to nruns = 100.
ntree	Number of trees to grow. Defaults to ntree = 500.
ntrain	Number (or proportion) of observation to be used as training cases. Defaults to 2/3 of the total number of observations.
nodesize	Minimum size of terminal nodes. Defaults to nodesize = 5.
mtry	Number of variables randomly sampled as candidates at each split. Defaults to 1/3 of the total number of covariates.
profile	Should the profile of the standardized mean squared prediction error be plotted at the end of the optimization? Defaults to profile = TRUE.
progress	Should a progress bar be displayed. Defaults to progress = TRUE.

Details

A fixed proportion of the total number of observations is used to calibrate (train) the random forest regression. The set of calibration observations is randomly selected from the full set of observations in each simulation. The remaining observations are used as test cases (validation). In general, the smaller the calibration dataset, the more simulation runs are needed to obtain stable estimates of the mean squared prediction error (MSPE).

The optimum number of iterations needed to de-bias the random forest regression is obtained observing the evolution of the MSPE as the number of iterations increases. The MSPE is defined as the mean of the squared differences between predicted and observed values.

Note

The original function was published as part of the dissertation of Ruo Xu, which was developed under the supervision of Daniel S Nettleton <dnett@iastate.edu> and Daniel J Nordman <dnordman@iastate.edu>.

Author(s)

Ruo Xu <xuruo.isu@gmail.com>, with improvements by Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

Breiman, L. Random forests. *Machine Learning*. v. 45, p. 5-32, 2001.

Breiman, L. *Using adaptive bagging to debias regressions*. Berkeley: University of California, p. 16, 1999.

Liaw, A. & Wiener, M. Classification and regression by randomForest. *R News*. v. 2/3, p. 18-22, 2002.

Xu, R. *Improvements to random forest methodology*. Ames, Iowa: Iowa State University, p. 87, 2013.

See Also

randomForest

---

plotCor *Correlation plot*

Description

Plotting correlation matrices.

Usage

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

Arguments

r	Square matrix with correlation values.
r2	(optional) A second square matrix with correlation values.
col	(optional) Color table to use for image – see image for details. The default is a colorblind-friendly palette ("RdBu") created using brewer.pal.
breaks	(optional) Break points in sorted order to indicate the intervals for assigning the colors. See 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 and interesting way of showing the strength of correlations between variables. This is done by using a diverging color palette, where the darker the color, the stronger the absolute correlation.

`plotCor` also enables comparing correlations between the same variables at different points in time or space or for different observations. This can be 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.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

Examples

```
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

This function creates 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 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 variogram, plotHD, bubble and 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.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

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

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

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

See Also

variogram, plotHD, bubble, spplot.

Examples

```
# library(sp)
# data(meuse)
# plotESDA(z = meuse$zinc, lat = meuse$y, lon = meuse$x)
```

---

plotHD *Histogram and density plot*

Description

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

Usage

```
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 nd 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 histogram, densityplot, panel.mathdensity, powerTransform and bcPower to obtain more details about the main functions used to built plotHD.

Value

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

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

histogram, densityplot, panel.mathdensity, powerTransform, bcPower.

Examples

```
x <- rnorm(100, 10, 2)
plotHD(x, HD = "stack")
plotHD(x, HD = "over")
```

plotMS *Model series plot*

Description

This function produces a graphical output that allows the examination of the effect of using different model specifications (design) on the predictive performance of these models (a model series). It generally is used to access the results of functions buildMS and statsMS, but can be easily adapted to work with any model structure and performance measure.

Usage

```
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 a 1) 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 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 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 xyplot. Defaults to type = c("b","g"). See 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 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.points for more information.
arrange	Character string indicating how the model series should be arranged, which can be in ascending (asc) or descending (desc) order. Defaults to arrange = "desc". See arrange for more information.
color	Vector defining the colors to be used in the grid produced by function levelplot. If NULL, defaults to color = cm.colors(n), where n is the number of unique values in the columns defined by argument grid. See cm.colors to see how to use other color palettes.
xlim	Numeric vector of length 2, giving the x coordinates range. If 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 length 1, giving the x-axis labels. Defaults to 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.

Warning

Use the original functions xyplot and 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.

See Also

levelplot, xyplot, mvtsplot.

Examples

```
# 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 <- plotMS(obj, grid = c(2:4), line = "adj_r2", ind = 1,
            color = c("lightyellow", "palegreen"),
            main = "Model Series Plot")
print(p)
```

---

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

Examples

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

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

Description

This function returns several statistics measuring the performance of a series of linear models built using the function buildMS, with an option to rank the models based on one of the returned performance statistics.

Usage

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

Arguments

model	A list of linear models returned by buildMS.
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 buildMS. 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 statsMS 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 buildMS. This feature was included in statsMS 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

1. Include other performance statistics such as: PRESS, BIC, Mallow's Cp, max(VIF);
2. 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.

See Also

`buildMS`, `plotMS`.

Examples

```
## Not run:
# based on the second example of function 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 <- buildMS(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 <- statsMS(cpus.ms, design.info = cpus.des, arrange.by = "aic")

## End(Not run)
```

`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 `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.

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.

See Also

`vif`, `stepAIC`.

Examples

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

trend.terms *Extract spatial trend data*

Description

Extract spatial trend data from an object of class `likfit`.

Usage

```
trend.terms(x)

trend.matrix(x)
```

Arguments

`x`	Object of class `likfit`.

Details

`trend.terms` is similar to `terms`.

`trend.matrix` is similar to `model.frame`.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

See Also

`likfit`

vgmICP *Initial covariance parameters (ICP)*

Description

Guess the initial values for the covariance parameters required to fit a variogram model.

Usage

```
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 response variable for which the initial values for the 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 lag-distance classes, or a numeric vector with the lower and upper bounds of the lag-distance classes. If missing, the lag-distance classes are computed using vgmLags. 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 lag-distance classes 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 initial covariance parameters. Defaults to method = "a". See 'Details' for more information.
min.npairs	Positive integer defining the minimum number of point-pairs required so that a lag-distance class is used for guessing the initial covariance parameters. Defaults to min.npairs = 30.
model	Character keyword defining the variogram model that will be fitted to the data. Currently, most basic variogram models are accepted. See cov.spatial for more information. Defaults to model = "matern".
nu	numerical value for the additional smoothness parameter ν of the correlation function. See RMmodel and argument kappa of cov.spatial for more information.
estimator	Character keyword defining the estimator for computing the sample variogram, with options "qn", "mad", "matheron", and "ch". Defaults to estimator = "qn". See sample.variogram for more details.
plotit	Should the guessed initial covariance parameters be plotted along with the sample variogram? Defaults to plotit = FALSE.

Details

There are five methods two guess the initial covariance parameters (ICP). Two of them, `"a"` and `"c"`, rely a sample variogram with exponentially spaced lag-distance classes, while the other three, `"b"`, `"d"`, and `"e"`, use equidistant lag-distance classes (see `vgmLags`). 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 Jian et al. (1996) and is implemented in SAS/STAT(R) 9.22.

Method `"c"` is implemented in the **automap**-package and was developed by Hiemstra et al. (2009).

Method `"d"` was developed by Desassis & Renard (2012).

Method `"e"` was proposed by Larrondo et al. (2003) and is implemented in the VARFIT module of GSLIB.

Value

A vector of numeric values: the guesses for the covariance parameters nugget, partial sill, and range.

Note

Package **geoR** is used to guess the range (scale) parameter of the following covariance models: "matern" (except when `nu = 0.5`), "powered.exponential", "stable", "cauchy", "gencauchy", "gneiting", and "gneiting.matern". However, **geoR** is an orphan package since 2020-01-12. Thus, if **geoR** is not installed, a guess of the practical range of these covariance models is returned. The practical range is the distance at which the semivariance reaches its maximum, i.e. the sill.

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

`vgmLags`, `sample.variogram`, `autofitVariogram`

Examples

```
data(meuse, package = "sp")
icp <- vgmICP(z = log(meuse$copper), coords = meuse[, 1:2])
```

`vgmLags` *Lag-distance classes for variogram estimation*

Description

Computation of lag-distance classes for variogram estimation.

Usage

```
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 lag-distance classes that should be computed. Defaults to `n = 7`.
`type`	Character value defining the type of lag-distance classes that should be computed, with options `"equi"` (equidistant) and `"exp"` (exponential). 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 lag-distance classes 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 spaced lag-distance classes. Used only when `type = "exp"`. Defaults to `base = 2`, i.e. the width of the rightmost lag-distance classes 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 lag-distance classes. Defaults to `zero = 0.0001`.
`count`	Should the number of points (`"points"`) or point-pairs (`"pairs"`) per lag-distance class be computed? Defaults to `count = "pairs"`.

Value

Vector of numeric values with the lower and upper boundaries of the lag-distance classes. The number of points or point-pairs per lag-distance class is returned as an attribute.

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.

See Also

optimPPL

Examples

```
data(meuse, package = "sp")
lags_points <- vgmLags(coords = meuse[, 1:2], count = "points")
lags_pairs <- vgmLags(coords = meuse[, 1:2], count = "pairs")
```

---

vgmSCV *Spatially correlated variance (SCV)*

Description

Compute the proportion of the variance that is spatially correlated.

Usage

```
## 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

vgmLags

Index