Package: propagate (via r-universe)

October 20, 2026

Type Package

LazyLoad no

LazyData no

Title Propagation of Uncertainty

Version 1.0-7

Date 2018-05-11

Author Andrej-Nikolai Spiess <a.spiess@uke.uni-hamburg.de>

Maintainer Andrej-Nikolai Spiess <a.spiess@uke.uni-hamburg.de>

Description Propagation of uncertainty using higher-order Taylor expansion and Monte Carlo simulation.

License GPL (>= 2)

Depends R (>= 2.13.0), MASS, tmvtnorm, Rcpp (>= 0.10.1), ff, minpack.lm

LinkingTo Rcpp

NeedsCompilation yes

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

Date/Publication 2018-06-01 20:10:26 UTC

RemoteUrl https://github.com/anspiess/propagate

RemoteRef HEAD

RemoteSha 6d3fb02d711b71aa07d2c29b42dfb58bd07f7627

Contents

bigcor 2
cor2cov 4
datasets 5
fitDistr 7
interval 11
makeDat 14
makeDerivs 15

matrixStats 17
mixCov 18
moments 20
numDerivs 21
plot.propagate 22
predictNLS 23
propagate 27
rDistr 36
statVec 38
stochContr 39
summary.propagate 40
WelchSatter 41

Index **43**

`bigcor` *Creating very large correlation/covariance matrices*

Description

The storage of a value in double format needs 8 bytes. When creating large correlation matrices, the amount of RAM might not suffice, giving rise to the dreaded *"cannot allocate vector of size ..."* error. For example, an input matrix with 50000 columns/100 rows will result in a correlation matrix with a size of 50000 x 50000 x 8 Byte / (1024 x 1024 x 1024) = 18.63 GByte, which is still more than most standard PCs. `bigcor` uses the framework of the 'ff' package to store the correlation/covariance matrix in a file. The complete matrix is created by filling a large preallocated empty matrix with sub-matrices at the corresponding positions. See 'Details'. Calculation time is ~ 20s for an input matrix of 10000 x 100 (cols x rows).

Usage

```
bigcor(x, y = NULL, fun = c("cor", "cov"), size = 2000,
       verbose = TRUE, ...)
```

Arguments

`x`	the input matrix.
`y`	`NULL` (default) or a vector, matrix or data frame with compatible dimensions to `x`.
`fun`	create either a `cor`elation or `cov`ariance matrix.
`size`	the n x n block size of the submatrices. 2000 has shown to be time-effective.
`verbose`	logical. If `TRUE`, information is printed in the console when running.
`...`	other parameters to be passed to `cor` or `cor`.

Details

Calculates a correlation matrix $\mathbf{C}$ or covariance matrix $\mathbf{\Sigma}$ using the following steps:
1) An input matrix `x` with N columns is split into k equal size blocks (+ a possible remainder block) $A_1, A_2, \ldots, A_k$ of size n. The block size can be defined by the user, `size = 2000` is a good value because `cor` can handle this quite quickly (~ 400 ms). For example, if the matrix has 13796 columns, the split will be $A_1 = 1 \ldots 2000$; $A_2 = 2001 \ldots 4000$; $A_3 = 4001 \ldots 6000$; $A_4 = 6000 \ldots 8000$; $A_5 = 8001 \ldots 10000$; $A_6 = 10001 \ldots 12000$; $A_7 = 12001 \ldots 13796$.
2) For all pairwise combinations of blocks $\binom{k}{2}$, the $n \times n$ correlation sub-matrix is calculated. If `y = NULL`, $\mathrm{cor}(A_1, A_1), \mathrm{cor}(A_1, A_2), \ldots, \mathrm{cor}(A_k, A_k)$, otherwise $\mathrm{cor}(A_1, y), \mathrm{cor}(A_2, y), \ldots, \mathrm{cor}(A_k, y)$.
3) The sub-matrices are transferred into a preallocated $N \times N$ empty matrix at the corresponding position (where the correlations would usually reside). To ensure symmetry around the diagonal, this is done twice in the upper and lower triangle. If `y` was supplied, a $N \times M$ matrix is filled, with M = number of columns in `y`.

Since the resulting matrix is in 'ff' format, one has to subset to extract regions into normal `matrix`-like objects. See 'Examples'.

Value

The corresponding correlation/covariance matrix in 'ff' format.

Author(s)

Andrej-Nikolai Spiess

References

http://rmazing.wordpress.com/2013/02/22/bigcor-large-correlation-matrices-in-r/

Examples

```
## Small example to prove similarity
## to standard 'cor'. We create a matrix
## by subsetting the complete 'ff' matrix.
MAT <- matrix(rnorm(70000), ncol = 700)
COR <- bigcor(MAT, size= 500, fun = "cor")
COR <- COR[1:nrow(COR), 1:ncol(COR)]
all.equal(COR, cor(MAT)) # => TRUE

## Example for cor(x, y) with
## y = small matrix.
MAT1 <- matrix(rnorm(50000), nrow = 10)
MAT2 <- MAT1[, 4950:5000]
COR <- cor(MAT1, MAT2)
BCOR <- bigcor(MAT1, MAT2)
BCOR <- BCOR[1:5000, 1:ncol(BCOR)] # => convert 'ff' to 'matrix'
all.equal(COR, BCOR)

## Not run:
## Create large matrix.
```

```
MAT <- matrix(rnorm(57500), ncol = 5750)
COR <- bigcor(MAT, size= 2000, fun = "cor")

## Extract submatrix.
SUB <- COR[1:3000, 1:3000]
all.equal(SUB, cor(MAT[, 1:3000]))

## End(Not run)
```

---

cor2cov *Converting a correlation matrix into a covariance matrix*

Description

Converts a correlation matrix into a covariance matrix using variance information. It is therefore the opposite of cov2cor.

Usage

```
cor2cov(C, var)
```

Arguments

C	a symmetric numeric correlation matrix $\mathbf{C}$.
var	a vector of variances σ_n^2.

Details

Calculates the covariance matrix $\mathbf{\Sigma}$ using a correlation matrix $\mathbf{C}$ and outer products of the standard deviations σ_n:

$$\mathbf{\Sigma} = \mathbf{C} \cdot \sigma_n \otimes \sigma_n$$

Value

The corresponding covariance matrix.

Author(s)

Andrej-Nikolai Spiess

Examples

```
## Example in Annex H.2 from the GUM 2008 manual
## (see 'References'), simultaneous resistance
## and reactance measurement.
data(H.2)
attach(H.2)

## Original covariance matrix.
```

```
COV <- cov(H.2)
## extract variances
VAR <- diag(COV)

## cor2cov covariance matrix.
COV2 <- cor2cov(cor(H.2), VAR)

## Equal to original covariance matrix.
all.equal(COV2, COV)
```

`datasets` *Datasets from the GUM "Guide to the expression of uncertainties in measurement" (2008)*

Description

Several datasets found in "Annex H" of the GUM that are used in illustrating the different approaches to quantifying measurement uncertainty.

Usage

```
H.2
H.3
H.4
```

Details

H.2: Simultaneous resistance and reactance measurement, Table H.2
This example demonstrates the treatment of multiple measurands or output quantities determined simultaneously in the same measurement and the correlation of their estimates. It considers only the random variations of the observations; in actual practice, the uncertainties of corrections for systematic effects would also contribute to the uncertainty of the measurement results. The data are analysed in two different ways with each yielding essentially the same numerical values.
H.2.1 The measurement problem:
The resistance *R* and the reactance *X* of a circuit element are determined by measuring the amplitude *V* of a sinusoidally-alternating potential difference across its terminals, the amplitude *I* of the alternating current passing through it, and the phase-shift angle ϕ of the alternating potential difference relative to the alternating current. Thus the three input quantities are *V*, *I*, and ϕ and the three output quantities -the measurands- are the three impedance components *R*, *X*, and *Z*. Since $Z^2 = R^2 + X^2$, there are only two independent output quantities.
H.2.2 Mathematical model and data:
The measurands are related to the input quantities by Ohm's law:

$$R = \frac{V}{I}\cos\phi; \quad X = \frac{V}{I}\sin\phi; \quad Z = \frac{V}{I} \qquad \text{(H.7)}$$

H.3: Calibration of a thermometer, Table H.6
This example illustrates the use of the method of least squares to obtain a linear calibration curve and

how the parameters of the fit, the intercept and slope, and their estimated variances and covariance, are used to obtain from the curve the value and standard uncertainty of a predicted correction.

H.3.1 The measurement problem:

A thermometer is calibrated by comparing n = 11 temperature readings t_k of the thermometer, each having negligible uncertainty, with corresponding known reference temperatures $t_{R,k}$ in the temperature range 21°C to 27°C to obtain the corrections $b_k = t_{R,k} - t_k$ to the readings. The *measured* corrections b_k and *measured* temperatures t_k are the input quantities of the evaluation. A linear calibration curve

$$b(t) = y_1 + y_2(t - t_0) \qquad \text{(H.12)}$$

is fitted to the measured corrections and temperatures by the method of least squares. The parameters y_1 and y_2, which are respectively the intercept and slope of the calibration curve, are the two measurands or output quantities to be determined. The temperature t_0 is a conveniently chosen exact reference temperature; it is not an independent parameter to be determined by the least-squares fit. Once y_1 and y_2 are found, along with their estimated variances and covariance, Equation (H.12) can be used to predict the value and standard uncertainty of the correction to be applied to the thermometer for any value t of the temperature.

H.4: Measurement of activity, Table H.7

This example is similar to example H.2, the simultaneous measurement of resistance and reactance, in that the data can be analysed in two different ways but each yields essentially the same numerical result. The first approach illustrates once again the need to take the observed correlations between input quantities into account.

H.4.1 The measurement problem:

The unknown radon ($^{222}\mathrm{Rn}$) activity concentration in a water sample is determined by liquid-scintillation counting against a radon-in-water standard sample having a known activity concentration. The unknown activity concentration is obtained by measuring three counting sources consisting of approximately 5g of water and 12g of organic emulsion scintillator in vials of volume 22ml:

Source (a) a *standard* consisting of a mass m_S of the standard solution with a known activity concentration;

Source (b) a matched *blank* water sample containing no radioactive material, used to obtain the background counting rate;

Source (c) the *sample* consisting of an aliquot of mass m_x with unknown activity concentration.

Six cycles of measurement of the three counting sources are made in the order standard - blank - sample; and each dead-time-corrected counting interval T_0 for each source during all six cycles is 60 minutes. Although the background counting rate cannot be assumed to be constant over the entire counting interval (65 hours), it is assumed that the number of counts obtained for each blank may be used as representative of the background counting rate during the measurements of the standard and sample in the same cycle. The data are given in Table H.7, where

t_S, t_B, t_x are the times from the reference time t = 0 to the midpoint of the dead-time-corrected counting intervals T_0 = 60 min for the standard, blank, and sample vials, respectively; although t_B is given for completeness, it is not needed in the analysis;

C_S, C_B, C_x are the number of counts recorded in the dead-time-corrected counting intervals T_0 = 60 min for the standard, blank, and sample vials, respectively.

The observed counts may be expressed as

$$C_S = C_B + \varepsilon A_S T_0 m_S e^{-\lambda t_S} \qquad \text{(H.18a)}$$

$$C_x = C_B + \varepsilon A_x T_0 m_x e^{-\lambda t_x} \qquad \text{(H.18b)}$$

where
ε is the liquid scintillation detection efficiency for $^{222}\mathrm{Rn}$ for a given source composition, assumed to be independent of the activity level;
A_S is the activity concentration of the standard at the reference time t = 0;
A_x is the measurand and is defined as the unknown activity concentration of the sample at the reference time t = 0;
m_S is the mass of the standard solution;
m_x is the mass of the sample aliquot;
λ is the decay constant for $^{222}\mathrm{Rn}$: $\lambda = (ln2)/T_{1/2} = 1.25894 \cdot 10^{-4}\ \mathrm{min}^{-1} (T_{1/2} = 5505.8\ \mathrm{min})$.
(...) This suggests combining Equations (H.18a) and (H.18b) to obtain the following expression for the unknown concentration in terms of the known quantities:

$$... = A_S \frac{m_S}{m_x} \frac{C_x - C_B}{C_S - C_B} e^{\lambda(t_x - t_S)} \qquad \text{(H.19)}$$

where $(C_x - C_B)e^{\lambda t_x}$ and $(C_S - C_B)e^{\lambda t_S}$ are, respectively, the background-corrected counts of the sample and the standard at the reference time t = 0 and for the time interval T_0 = 60 min.

Author(s)

Andrej-Nikolai Spiess, taken mainly from the GUM 2008 manual.

References

Evaluation of measurement data - Guide to the expression of uncertainty in measurement.
JCGM 100:2008 (GUM 1995 with minor corrections).
http://www.bipm.org/utils/common/documents/jcgm/JCGM_100_2008_E.pdf.

Evaluation of measurement data - Supplement 1 to the Guide to the expression of uncertainty in measurement - Propagation of distributions using a Monte Carlo Method.
JCGM 101:2008.
http://www.bipm.org/utils/common/documents/jcgm/JCGM_101_2008_E.pdf.

Examples

```
## See "Examples" in 'propagate'.
```

`fitDistr` *Fitting distributions to observations/Monte Carlo simulations*

Description

This function fits 32 different continuous distributions by (weighted) NLS to the histogram of Monte Carlo simulation results as obtained by `propagate` or any other vector containing large-scale observations. Finally, the fits are sorted by ascending `BIC`.

Usage

```
fitDistr(object, nbin = 100, weights = FALSE, verbose = TRUE,
        brute = c("fast", "slow"), plot = c("hist", "qq"),  distsel = NULL, ...)
```

Arguments

object	an object of class 'propagate' or a vector containing observations.
nbin	the number of bins in the histogram.
weights	numeric or logical. Either a numeric vector of weights, or if TRUE, the distributions are fitted with weights = 1/(counts per bin).
verbose	logical. If TRUE, steps of the analysis are printed to the console.
brute	complexity of the brute force approach. See 'Details'.
plot	if "hist", a plot with the "best" distribution (in terms of lowest BIC) on top of the histogram is displayed. If "qq", a QQ-Plot will display the difference between the observed and fitted quantiles.
distsel	a vector of distribution numbers to select from the complete cohort as listed below, e.g. c(1:10, 15).
...	other parameters to be passed to the plots.

Details

Fits the following 32 distributions using (weighted) residual sum-of-squares as the minimization criterion for nls.lm:
1) Normal distribution (dnorm) => https://en.wikipedia.org/wiki/Normal_distribution
2) Skewed-normal distribution (propagate:::dsn) => https://en.wikipedia.org/wiki/Skew_normal_distribution
3) Generalized normal distribution (propagate:::dgnorm) => https://en.wikipedia.org/wiki/Generalized_normal_distribution
4) Log-normal distribution (dlnorm) => https://en.wikipedia.org/wiki/Log-normal_distribution
5) Scaled and shifted t-distribution (propagate:::dst) => GUM 2008, Chapter 6.4.9.2.
6) Logistic distribution (dlogis) => https://en.wikipedia.org/wiki/Logistic_distribution
7) Uniform distribution (dunif) => https://en.wikipedia.org/wiki/Uniform_distribution_(continuous)
8) Triangular distribution (propagate:::dtriang) => https://en.wikipedia.org/wiki/Triangular_distribution
9) Trapezoidal distribution (propagate:::dtrap) => https://en.wikipedia.org/wiki/Trapezoidal_distribution
10) Curvilinear Trapezoidal distribution (propagate:::dctrap) => GUM 2008, Chapter 6.4.3.1
11) Gamma distribution (dgamma) => https://en.wikipedia.org/wiki/Gamma_distribution
12) Inverse Gamma distribution (propagate:::dinvgamma) => https://en.wikipedia.org/wiki/Inverse-gamma_distribution
13) Cauchy distribution (dcauchy) => https://en.wikipedia.org/wiki/Cauchy_distribution
14) Laplace distribution (propagate:::dlaplace) => https://en.wikipedia.org/wiki/Laplace_distribution
15) Gumbel distribution (propagate:::dgumbel) => https://en.wikipedia.org/wiki/Gumbel_distribution
16) Johnson SU distribution (propagate:::dJSU) => https://en.wikipedia.org/wiki/Johnson_SU_distribution
17) Johnson SB distribution (propagate:::dJSB) => https://www.mathwave.com/articles/johnson_sb_distribution.html
18) Three-parameter Weibull distribution (propagate:::dweibull2) => https://en.wikipedia.

org/wiki/Weibull_distribution
19) Two-parameter beta distribution (dbeta2) => https://en.wikipedia.org/wiki/Beta_distribution#Two_parameters_2
20) Four-parameter beta distribution (propagate:::dbeta2) => https://en.wikipedia.org/wiki/Beta_distribution#Four_parameters_2
21) Arcsine distribution (propagate:::darcsin) => https://en.wikipedia.org/wiki/Arcsine_distribution
22) Von Mises distribution (propagate:::dmises) => https://en.wikipedia.org/wiki/Von_Mises_distribution
23) Inverse Gaussian distribution (propagate:::dinvgauss) => https://en.wikipedia.org/wiki/Inverse_Gaussian_distribution
24) Generalized Extreme Value distribution (propagate:::dgevd) => https://en.wikipedia.org/wiki/Generalized_extreme_value_distribution
25) Rayleigh distribution (propagate:::drayleigh) => https://en.wikipedia.org/wiki/Rayleigh_distribution
26) Chi-square distribution (dchisq) => https://en.wikipedia.org/wiki/Chi-squared_distribution
27) Exponential distribution (dexp) => https://en.wikipedia.org/wiki/Exponential_distribution
28) F-distribution (df) => https://en.wikipedia.org/wiki/F-distribution
29) Burr distribution (dburr) => https://en.wikipedia.org/wiki/Burr_distribution
30) Chi distribution (dchi) => https://en.wikipedia.org/wiki/Chi_distribution
31) Inverse Chi-square distribution (dinvchisq) => https://en.wikipedia.org/wiki/Inverse-chi-squared_distribution
32) Cosine distribution (dcosine) => https://en.wikipedia.org/wiki/Raised_cosine_distribution

All distributions are fitted with a brute force approach, in which the parameter space is extended over three orders of magnitude $(0.1, 1, 10) \times \beta_i$ when `brute = "fast"`, or five orders $(0.01, 0.1, 1, 10, 100) \times \beta_i$ when `brute = "slow"`. Approx. 20-90s are needed to fit for the fast version, depending mainly on the number of bins.

The goodness-of-fit (GOF) is calculated with BIC from the (weighted) log-likelihood of the fit:

$$\ln(\mathrm{L}) = 0.5 \cdot \left(-\mathrm{N} \cdot \left(\ln(2\pi) + 1 + \ln(\mathrm{N}) - \sum_{\mathrm{i}=1}^{\mathrm{n}} \log(\mathrm{w_i}) + \ln\left(\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{w_i} \cdot \mathrm{x_i^2} \right) \right) \right)$$

$$\mathrm{BIC} = -2\ln(\mathrm{L}) + (\mathrm{N} - \mathrm{k})\ln(\mathrm{N})$$

with x_i = the residuals from the NLS fit, N = the length of the residual vector, k = the number of parameters of the fitted model and w_i = the weights.

In contrast to some other distribution fitting softwares (i.e. Easyfit, Mathwave) that use residual sum-of-squares/Anderson-Darling/Kolmogorov-Smirnov statistics as GOF measures, the application of BIC accounts for increasing number of parameters in the distribution fit and therefore compensates for overfitting. Hence, this approach is more similar to ModelRisk (Vose Software) and as employed in `fitdistr` of the 'MASS' package. Another application is to identify a possible distribution for the raw data prior to using Monte Carlo simulations from this distribution. However, a decent number of observations should be at hand in order to obtain a realistic estimate of the proper distribution. See 'Examples'.
The code for the density functions can be found in file "distr-densities.R".

IMPORTANT: It can be feasible to set `weights = TRUE` in order to give more weight to bins with low counts. See 'Examples'. ALSO: Distribution fitting is highly sensitive to the number of de-

fined histogram bins, so it is advisable to change this parameter and inspect if the order of fitted distributions remains stable.

Value

A list with the following items:

`stat`: the by BIC value ascendingly sorted distribution names, including RSS and MSE.
`fit`: a list of the results from `nls.lm` for each distribution model, also sorted ascendingly by BIC values.
`par`: a list of the estimated parameters of the models in `fit`.
`se`: a list of the parameters' standard errors, calculated from the square root of the covariance matrices diagonals.
`dens`: a list with all density function used for fitting, sorted as in `fit`.
`bestfit`: the best model in terms of lowest BIC.
`bestpar`: the parameters of `bestfit`.
`bestse`: the parameters' standard errors of `bestfit`.
`fitted`: the fitted values of `bestfit`.
`residuals`: the residuals of `bestfit`.

Author(s)

Andrej-Nikolai Spiess

References

Continuous univariate distributions, Volume 1.
Johnson NL, Kotz S and Balakrishnan N.
Wiley Series in Probability and Statistics, 2.ed (2004).

A guide on probability distributions.
R-forge distributions core team.
http://dutangc.free.fr/pub/prob/probdistr-main.pdf.

Univariate distribution relationships.
Leemis LM and McQueston JT.
The American Statistician (2008), **62**: 45-53.

Examples

```
## Not run:
## Linear example, small error
## => Normal distribution.
EXPR1 <- expression(x + 2 * y)
x <- c(5, 0.01)
y <- c(1, 0.01)
DF1 <- cbind(x, y)
RES1 <- propagate(expr = EXPR1, data = DF1, type = "stat",
                  do.sim = TRUE, verbose = TRUE)
fitDistr(RES1)
```

```
## Ratio example, larger error
## => Gamma distribution.
EXPR2 <- expression(x/2 * y)
x <- c(5, 0.1)
y <- c(1, 0.02)
DF2 <- cbind(x, y)
RES2 <- propagate(expr = EXPR2, data = DF2, type = "stat",
                  do.sim = TRUE, verbose = TRUE)
fitDistr(RES2)

## Exponential example, large error
## => Log-Normal distribution.
EXPR3 <- expression(x^(2 * y))
x <- c(5, 0.1)
y <- c(1, 0.1)
DF3 <- cbind(x, y)
RES3 <- propagate(expr = EXPR3, data = DF3, type = "stat",
                  do.sim = TRUE, verbose = TRUE)
fitDistr(RES3)

## Rectangular input distributions result
## in Curvilinear Trapezoidal output distribution.
A <- runif(100000, 20, 25)
B <- runif(100000, 3, 3.5)
DF4 <- cbind(A, B)
EXPR4 <- expression(A + B)
RES4 <- propagate(EXPR4, data = DF4, type = "sim",
                  use.cov = FALSE, do.sim = TRUE)
fitDistr(RES4)

## Fitting with 1/counts as weights.
EXPR5 <- expression(x + 2 * y)
x <- c(5, 0.05)
y <- c(1, 0.05)
DF5 <- cbind(x, y)
RES5 <- propagate(expr = EXPR5, data = DF5, type = "stat",
                  do.sim = TRUE, verbose = TRUE, weights = TRUE)
fitDistr(RES5)

## Using only selected distributions.
EXPR6 <- expression(x + sin(y))
x <- c(5, 0.1)
y <- c(1, 0.2)
DF6 <- cbind(x, y)
RES6 <- propagate(expr = EXPR6, data = DF6, type = "stat",
                  do.sim = TRUE)
fitDistr(RES6, distsel = c(1:10, 15, 28))

## End(Not run)
```

`interval` *Uncertainty propagation based on interval arithmetics*

Description

Calculates the uncertainty of a model by using interval arithmetics based on a "combinatorial sequence grid evaluation" approach, thereby avoiding the classical dependency problem that inflates the result interval.

Usage

```
interval(df, expr, seq = 10, plot = TRUE)
```

Arguments

`df`	a 2-row dataframe/matrix with lower border values A_i in the first row and upper border values B_i in the second row. Column names must correspond to the variable names in `expr`.
`expr`	an expression, such as `expression(x/y)`.
`seq`	the sequence length from A_i to B_i in $[A_i, B_i]$.
`plot`	logical. If `TRUE`, plots the evaluations and min/max values as blue border lines.

Details

For two variables x, y with intervals $[x_1, x_2]$ and $[y_1, y_2]$, the four basic arithmetic operations $\langle \text{op} \rangle \in \{+, -, \cdot, /\}$ are

$$[x_1, x_2] \langle \text{op} \rangle [y_1, y_2] =$$

$$[\min(x_1 \langle \text{op} \rangle y_1, x_1 \langle \text{op} \rangle y_2, x_2 \langle \text{op} \rangle y_1, x_2 \langle \text{op} \rangle y_2), \max(x_1 \langle \text{op} \rangle y_1, x_1 \langle \text{op} \rangle y_2, x_2 \langle \text{op} \rangle y_1, x_2 \langle \text{op} \rangle y_2)]$$

So for a function $f([x_1, x_2], [y_1, y_2], [z_1, z_2], ...)$ with k variables, we have to create all combinations $C_i = \binom{\{\{x_1,x_2\},\{y_1,y2\},\{z_1,z_2\},...\}}{k}$, evaluate their function values $R_i = f(C_i)$ and select $R = [\min R_i, \max R_i]$.
The so-called *dependency problem* is a major obstacle to the application of interval arithmetic and arises when the same variable exists in several terms of a complicated and often nonlinear function. In these cases, over-estimation can cover a range that is significantly larger, i.e. $\min R_i \ll \min f(x, y, z, ...), \max R_i \gg \max f(x, y, z, ...)$. For an example, see http://en.wikipedia.org/w/index.php?title=Interval_arithmetic under "Dependency problem". A partial solution to this problem is to refine R_i by dividing $[x_1, x_2]$ into i smaller subranges to obtain sequence $(x_1, x_{1.1}, x_{1.2}, x_{1.i}, x_2)$. Again, all combinations are evaluated as described above, resulting in a larger number of R_i in which $\min R_i$ and $\max R_i$ may be closer to $\min f(x, y, z, ...)$ and $\max f(x, y, z, ...)$, respectively. This is the "combinatorial sequence grid evaluation" approach which works quite well in scenarios where monotonicity changes direction (see 'Examples'), obviating the need to create multivariate derivatives (Hessians) or use some multivariate minimization algorithm.
If intervals are of type $[x_1 < 0, x_2 > 0]$, a zero is included into the middle of the sequence to avoid wrong results in case of even powers, i.e. $[-1, 1]^2 = [-1, 1][-1, 1] = [-1, 1]$ when actually the right interval is $[0, 1]$, see `curve(x^2, -1, 1)`.

Value

A 2-element vector with the resulting interval and an (optional) plot of all evaluations.

Author(s)

Andrej-Nikolai Spiess

References

Wikipedia entry is quite good, especially the section on the 'dependency problem':
http://en.wikipedia.org/wiki/Interval_arithmetic

Comparison to Monte Carlo and error propagation:
Interval Arithmetic in Power Flow Analysis.
Wang Z & Alvarado FL.
Power Industry Computer Application Conference (1991): 156-162.

Computer implementation
Interval arithmetic: From principles to implementation.
Hickey T, Ju Q, Van Emden MH.
JACM (2001), **48**: 1038-1068.

Complete Interval Arithmetic and its Implementation on the Computer.
Kulisch UW.
In: Numerical Validation in Current Hardware Architectures. Lecture Notes in Computer Science **5492** (2009): 7-26.

Examples

```
## Example 1: even squaring of negative interval.
EXPR1 <- expression(x^2)
DAT1 <- data.frame(x = c(-1, 1))
interval(DAT1, EXPR1)

## Example 2: A complicated nonlinear model.
## Reduce sequence length to 2 => original interval
## for quicker evaluation.
EXPR2 <- expression(C * sqrt((520 * H * P)/(M *(t + 460))))
H <- c(64, 65)
M <- c(16, 16.2)
P <- c(361, 365)
t <- c(165, 170)
C <- c(38.4, 38.5)
DAT2 <- makeDat(EXPR2)
interval(DAT2, EXPR2, seq = 2)

## Example 3: Body Mass Index taken from
## http://en.wikipedia.org/w/index.php?title=Interval_arithmetic
EXPR3 <- expression(m/h^2)
m <- c(79.5, 80.5)
h <- c(1.795, 1.805)
DAT3 <- makeDat(EXPR3)
interval(DAT3, EXPR3)

## Example 4: Linear model.
EXPR4 <- expression(a * x + b)
a <- c(1, 2)
```

```
b <- c(5, 7)
x <- c(2, 3)
DAT4 <- makeDat(EXPR4)
interval(DAT4, EXPR4)

## Example 5: Overestimation from dependency problem.
# Original interval with seq = 2 => [1, 7]
EXPR5 <- expression(x^2 - x + 1)
x <- c(-2, 1)
DAT5 <- makeDat(EXPR5)
interval(DAT5, EXPR5, seq = 2)

# Refine with large sequence => [0.75, 7]
interval(DAT5, EXPR5, seq = 100)
# Tallies with curve function.
curve(x^2 - x + 1, -2, 1)

## Example 6: Underestimation from dependency problem.
# Original interval with seq = 2 => [0, 0]
EXPR6 <- expression(x - x^2)
x <- c(0, 1)
DAT6 <- makeDat(EXPR6)
interval(DAT6, EXPR6, seq = 2)

# Refine with large sequence => [0, 0.25]
interval(DAT6, EXPR6, seq = 100)
# Tallies with curve function.
curve(x - x^2, 0, 1)
```

`makeDat` *Create a dataframe from the variables defined in an expression*

Description

Creates a dataframe from the variables defined in an expression by `cbind`ing the corresponding data found in the workspace. This is a convenience function for creating a dataframe to be passed to `propagate`, when starting with data which was simulated from distributions, i.e. when `type = "sim"`. Will throw an error if a variable is defined in the expression but is not available from the workspace.

Usage

```
makeDat(expr)
```

Arguments

`expr`	an expression to be use for `propagate`.

Value

A dataframe containing the data defined in `expr` in columns.

Author(s)

Andrej-Nikolai Spiess

Examples

```
## Simulating from uniform
## and normal distribution,
## run 'propagate'.
EXPR1 <- expression(a + b^c)
a <- rnorm(100000, 12, 1)
b <- rnorm(100000, 5, 0.1)
c <- runif(100000, 6, 7)

DAT1 <- makeDat(EXPR1)
propagate(EXPR1, DAT1, type = "sim", cov = FALSE)
```

`makeDerivs` *Utility functions for creating Gradient- and Hessian-like matrices with symbolic derivatives and evaluating them in an environment*

Description

These are three different utility functions that create matrices containing the symbolic partial derivatives of first (`makeGrad`) and second (`makeHess`) order and a function for evaluating these matrices in an environment. The evaluations of the symbolic derivatives are used within the `propagate` function to calculate the propagated uncertainty, but these functions may also be useful for other applications.

Usage

```
makeGrad(expr, order = NULL)
makeHess(expr, order = NULL)
evalDerivs(deriv, envir)
```

Arguments

`expr`	an expression, such as `expression(x/y)`.
`order`	order of creating partial derivatives, i.e. `c(2, 1)`. See 'Examples'.
`deriv`	a matrix returned from `makeGrad` or `makeHess`.
`envir`	an environment to evaluate in. By default the workspace.

Details

Given a function $f(x_1, x_2, \ldots, x_n)$, the following matrices containing symbolic derivatives of f are returned:

makeGrad:

$$\nabla(f) = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right]$$

makeHess:

$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Value

The symbolic or evaluated Gradient/Hessian matrices.

Author(s)

Andrej-Nikolai Spiess

References

The Matrix Cookbook (Version November 2012).
Petersen KB & Pedersen MS.
http://ais.informatik.uni-freiburg.de/teaching/ws12/mapping/pdf/imm3274.pdf

Examples

```
EXPR <- expression(a^b + sin(c))
ENVIR <- list(a = 2, b = 3, c = 4)

## First-order partial derivatives: Gradient.
GRAD <- makeGrad(EXPR)

## This will evaluate the Gradient.
evalDerivs(GRAD, ENVIR)

## Second-order partial derivatives: Hessian.
HESS <- makeHess(EXPR)

## This will evaluate the Hessian.
evalDerivs(HESS, ENVIR)

## Change derivatives order.
```

```
GRAD <- makeGrad(EXPR, order = c(2,1,3))
evalDerivs(GRAD, ENVIR)
```

---

matrixStats *Fast column- and row-wise versions of variance coded in C++*

Description

These two functions are fast C++ versions for column- and row-wise variance calculation on matrices/data.frames and are meant to substitute the classical `apply(mat, 1, var)` approach.

Usage

```
colVarsC(x)
rowVarsC(x)
```

Arguments

x	a matrix or data.frame

Details

They are coded in a way that they automatically remove NA values, so they behave like `na.rm = TRUE`.

Value

A vector with the variance values.

Author(s)

Andrej-Nikolai Spiess

Examples

```
## Speed comparison on large matrix.
## ~ 110x speed increase!
## Not run:
MAT <- matrix(rnorm(10 * 500000), ncol = 10)
system.time(RES1 <- apply(MAT, 1, var))
system.time(RES2 <- rowVarsC(MAT))
all.equal(RES1, RES2)

## End(Not run)
```

`mixCov` *Aggregating covariances matrices and/or error vectors into a single covariance matrix*

Description

This function aggregates covariances matrices, single variance values or a vector of multiple variance values into one final covariance matrix suitable for `propagate`.

Usage

```
mixCov(...)
```

Arguments

...	either covariance matrices, or a vector of single/multiple variance values.

Details

'Mixes' (aggregates) data of the following types into a final covariance matrix:
1) covariance matrices $\mathbf{\Sigma}$ that are already available.
2) single variance values σ^2.
3) a vector of variance values $\sigma_1^2, \sigma_2^2, ..., \sigma_n^2$.

This is accomplished by filling a $m_1 + m_2 + \ldots + m_n$ sized square matrix $\mathbf{C}$ succesively with elements $1 \ldots m_1, m_1+1 \ldots m1+m2, \ldots, m_n+1 \ldots m_n+m_{n+1}$ with either covariance matrices at $C_{m_n+1 \ldots m_n+m_{n+1}, m_n+1 \ldots m_n+m_{n+1}}$ or single variance values on the diagonals at C_{m_n,m_n}.

Value

The aggregated covariance matrix.

Author(s)

Andrej-Nikolai Spiess

References

Evaluation of measurement data - Guide to the expression of uncertainty in measurement.
JCGM 100:2008 (GUM 1995 with minor corrections).
http://www.bipm.org/utils/common/documents/jcgm/JCGM_100_2008_E.pdf.

Evaluation of measurement data - Supplement 1 to the Guide to the expression of uncertainty in measurement - Propagation of distributions using a Monte Carlo Method.
JCGM 101:2008.
http://www.bipm.org/utils/common/documents/jcgm/JCGM_101_2008_E.pdf.

Examples

```
#######################################################
## Example in Annex H.4.1 from the GUM 2008 manual
## (see 'References'), measurement of activity.
## This will give exactly the same values as Table H.8.
data(H.4)
attach(H.4)
T0 <- 60
lambda <- 1.25894E-4
Rx <- ((Cx - Cb)/60) * exp(lambda * tx)
Rs <- ((Cs - Cb)/60) * exp(lambda * ts)

mRx <- mean(Rx)
sRx <- sd(Rx)/sqrt(6)
mRx
sRx

mRs <- mean(Rs)
sRs <- sd(Rs)/sqrt(6)
mRs
sRs

R <- Rx/Rs
mR <- mean(R)
sR <- sd(R)/sqrt(6)
mR
sR

cor(Rx, Rs)

## Definition as in H.4.3.
As <- c(0.1368, 0.0018)
ms <- c(5.0192, 0.005)
mx <- c(5.0571, 0.001)

## We have to scale Rs/Rx by sqrt(6) to get the
## corresponding covariances.
Rs <- Rs/sqrt(6)
Rx <- Rx/sqrt(6)

## Here we create an aggregated covariance matrix
## from the raw and summary data.
COV1 <- cov(cbind(Rs, Rx))
COV <- mixCov(COV1, As[2]^2, ms[2]^2, mx[2]^2)
COV

## Prepare the data for 'propagate'.
MEANS <- c(mRs, mRx, As[1], ms[1], mx[1])
SDS <- c(sRs, sRx, As[2], ms[2], mx[2])
DAT <- rbind(MEANS, SDS)
colnames(DAT) <- c("Rs", "Rx", "As", "ms", "mx")
```

```
## This will give exactly the same values as
## in H.4.3/H.4.3.1.
EXPR <- expression(As * (ms/mx) * (Rx/Rs))
RES <- propagate(EXPR, data = DAT, cov = COV, nsim = 100000)
RES
```

`moments` *Skewness and (excess) Kurtosis of a vector of values*

Description

These functions calculate skewness and excess kurtosis of a vector of values. They were taken from the package 'moments'.

Usage

```
skewness(x, na.rm = FALSE)
kurtosis(x, na.rm = FALSE)
```

Arguments

`x`	a numeric vector, matrix or data frame.
`na.rm`	logical. Should missing values be removed?

Details

Skewness:

$$\frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^3}{\left(\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2\right)^{3/2}}$$

(excess) Kurtosis:

$$\frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^4}{\left(\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2\right)^{2}} - 3$$

Value

The skewness/kurtosis values.

Author(s)

Andrej-Nikolai Spiess

Examples

```
X <- rnorm(100, 20, 2)
skewness(X)
kurtosis(X)
```

numDerivs *Functions for creating Gradient and Hessian matrices by numerical differentiation (Richardson's method) of the partial derivatives*

Description

These two functions create Gradient and Hessian matrices by Richardson's central finite difference method of the partial derivatives for any expression.

Usage

```
numGrad(expr, envir = .GlobalEnv)
numHess(expr, envir = .GlobalEnv)
```

Arguments

expr	an expression, such as `expression(x/y)`.
envir	the `environment` to evaluate in.

Details

Calculates first- and second-order numerical approximation using Richardson's **central difference formula**:

$$f_i'(x) \approx \frac{f(x_1, \ldots, x_i + d, \ldots, x_n) - f(x_1, \ldots, x_i - d, \ldots, x_n)}{2d}$$

$$f_i''(x) \approx \frac{f(x_1, \ldots, x_i + d, \ldots, x_n) - 2f(x_1, \ldots, x_n) + f(x_1, \ldots, x_i - d, \ldots, x_n)}{d^2}$$

Value

The numeric Gradient/Hessian matrices.

Note

The two functions are modified versions of the `genD` function in the 'numDeriv' package, but a bit more easy to handle because they use expressions and the function's `x` value must not be defined as splitted scalar values `x[1], x[2], ... x[n]` in the body of the function.

Author(s)

Andrej-Nikolai Spiess

Examples

```
## Check for equality of symbolic
## and numerical derivatives.
EXPR <- expression(2^x + sin(2 * y) - cos(z))
x <- 5
y <- 10
z <- 20

symGRAD <- evalDerivs(makeGrad(EXPR))
numGRAD <- numGrad(EXPR)
all.equal(symGRAD, numGRAD)

symHESS <- evalDerivs(makeHess(EXPR))
numHESS <- numHess(EXPR)
all.equal(symHESS, numHESS)
```

`plot.propagate` *Plotting function for 'propagate' objects*

Description

Creates a histogram of the evaluated results from the multivariate simulated data, along with a density curve, alpha-based confidence intervals, median and mean.

Usage

```
## S3 method for class 'propagate'
plot(x, logx = FALSE, ...)
```

Arguments

`x`	an object returned from `propagate`.
`logx`	logical. Should the data be displayed on a logarithmic abscissa?
`...`	other parameters to `hist`.

Value

A plot as described above.

Author(s)

Andrej-Nikolai Spiess

Examples

```
EXPR1 <- expression(x^2 * sin(y))
x <- c(5, 0.01)
y <- c(1, 0.01)
DF1 <- cbind(x, y)
RES1 <- propagate(expr = EXPR1, data = DF1, type = "stat",
                  nsim = 100000, alpha = 0.01)
plot(RES1)
```

`predictNLS` *Confidence/prediction intervals for (weighted) nonlinear models based on uncertainty propagation*

Description

A function for calculating confidence/prediction intervals of (weighted) nonlinear models for the supplied or new predictor values, by using first-/second-order Taylor expansion and Monte Carlo simulation. This approach can be used to construct more realistic error estimates and confidence/prediction intervals for nonlinear models than what is possible with only a simple linearization (first-order Taylor expansion) approach. Another application is when there is an "error in x" setup with uncertainties in the predictor variable (See 'Examples'). This function will also work in the presence of multiple predictors with/without errors.

Usage

```
predictNLS(model, newdata, newerror, interval = c("confidence", "prediction", "none"),
           alpha = 0.05, ...)
```

Arguments

`model`	a model obtained from `nls` or `nlsLM` (package 'minpack.lm').
`newdata`	a data frame with new predictor values, having the same column names as in `model`. See `predict.nls` and 'Examples'. If omitted, the model's predictor values are employed.
`newerror`	a data frame with optional error values, having the same column names as in `model` and in the same order as in `newdata`. See 'Examples'.
`interval`	A character string indicating if confidence/prediction intervals are to be calculated or not.
`alpha`	the α level.
`...`	other parameters to be supplied to `propagate`.

Details

Calculation of the propagated uncertainty σ_y using $\nabla\Sigma\nabla^T$ is called the "Delta Method" and is widely applied in NLS fitting. However, this method is based on first-order Taylor expansion and thus assummes linearity around $\hat{y} = f(x, \hat{\beta}_i)$. The second-order approach as implemented in the `propagate` function can partially correct for this restriction by using a second-order polynomial around $\hat{y}$.
Confidence and prediction intervals are calculated in a usual way using $t(1 - \frac{\alpha}{2}, \nu) \cdot \sigma_y$ (1) or $t(1 - \frac{\alpha}{2}, \nu) \cdot \sqrt{\sigma_y^2 + \sigma_r^2}$ (2), respectively, where the residual variance $\sigma_r^2 = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\nu}$ (3). The inclusion of σ_r^2 in the prediction interval is implemented as an extended gradient and "augmented" covariance matrix. So instead of using $\hat{y} = f(x, \hat{\beta})$ (4), we take $\hat{y} = f(x, \hat{\beta}) + \sigma_r^2$ (5) as the expression and augment the $i \times i$ covariance matrix Σ to an $(i + 1) \times (i + 1)$ covariance matrix, where $\Sigma_{i+1,i+1} = \sigma_r^2$. Partial differentiation and matrix multiplication will then yield, for example with two coefficients β_1 and β_2 and their corresponding covariance matrix Σ:

$$\begin{bmatrix} \frac{\partial f}{\partial \beta_1} & \frac{\partial f}{\partial \beta_2} & 1 \end{bmatrix} \begin{bmatrix} \sigma_1^2 & \sigma_1\sigma_2 & 0 \\ \sigma_2\sigma_1 & \sigma_2^2 & 0 \\ 0 & 0 & \sigma_r^2 \end{bmatrix} \begin{bmatrix} \frac{\partial f}{\partial \beta_1} \\ \frac{\partial f}{\partial \beta_2} \\ 1 \end{bmatrix} \quad (6)$$

$$= \left(\frac{\partial f}{\partial \beta_1}\right)^2 \sigma_1^2 + 2\frac{\partial f}{\partial \beta_1}\frac{\partial f}{\partial \beta_2}\sigma_1\sigma_2 + \left(\frac{\partial f}{\partial \beta_2}\right)^2 \sigma_2^2 + \sigma_r^2 \quad (7)$$

$\equiv \sigma_y^2 + \sigma_r^2$, which then goes into (2).
The advantage of the augmented covariance matrix is that it can be exploited for creating Monte Carlo simulation-based prediction intervals. This is new from version 1.0-6 and is based on the paradigm that we simply add another dimension by employing the augmented covariance matrix of (6) in the multivariate t-distribution random number generator (in our case `rtmvt`), with $\mu = 0$. All n simulations are then evaluated with (5) and the usual $[1 - \frac{\alpha}{2}, \frac{\alpha}{2}]$ quantiles calculated.

If errors are supplied to the predictor values in `newerror`, they need to have the same column names and order than the new predictor values.

Value

A list with the following items:
`summary`: The mean/error estimates obtained from first-/second-order Taylor expansion and Monte Carlo simulation, together with calculated confidence/prediction intervals based on asymptotic normality.
`prop`: the complete output from `propagate` for each value in `newdata`.

Author(s)

Andrej-Nikolai Spiess

References

Nonlinear Regression.
Seber GAF & Wild CJ.
John Wiley & Sons; 1ed, 2003.

Nonlinear Regression Analysis and its Applications.
Bates DM & Watts DG.
Wiley-Interscience; 1ed, 2007.

Statistical Error Propagation.
Tellinghuisen J.
J. Phys. Chem. A (2001), **47**: 3917-3921.

Least-squares analysis of data with uncertainty in x and y: A Monte Carlo methods comparison.
Tellinghuisen J.
Chemometr Intell Lab (2010), **47**: 160-169.

From the author's blog:
http://rmazing.wordpress.com/2013/08/14/predictnls-part-1-monte-carlo-simulation-confidence-intervals-for-nls-models/
http://rmazing.wordpress.com/2013/08/26/predictnls-part-2-taylor-approximation-confidence-intervals-for-nls-models/
https://rmazing.wordpress.com/2018/05/11/monte-carlo-based-prediction-intervals-for-nonlinear-regression/

Examples

```
## In these examples, 'nsim = 100000' to save
## Rcmd check time (CRAN). It is advocated
## to use at least 'nsim = 1000000' though...

## Example from ?nls.
DNase1 <- subset(DNase, Run == 1)
fm3DNase1 <- nls(density ~ Asym/(1 + exp((xmid - log(conc))/scal)),
                 data = DNase1, start = list(Asym = 3, xmid = 0, scal = 1))

## Using a single predictor value without error.
PROP1 <- predictNLS(fm3DNase1, newdata = data.frame(conc = 2), nsim = 100000)
PRED1 <- predict(fm3DNase1, newdata = data.frame(conc = 2))
PROP1$summary
PRED1
## => Prop.Mean.1 equal to PRED1

## Using a single predictor value with error.
PROP2 <- predictNLS(fm3DNase1, newdata = data.frame(conc = 2),
                    newerror = data.frame(conc = 0.5), nsim = 100000)
PROP2$summary

## Not run:
## Using a sequence of predictor values without error.
CONC <- seq(1, 12, by = 1)
PROP3 <- predictNLS(fm3DNase1, newdata = data.frame(conc = CONC))
PRED3 <- predict(fm3DNase1, newdata = data.frame(conc = CONC))
PROP3$summary
PRED3
## => Prop.Mean.1 equal to PRED3

## Plot mean and confidence values from first-/second-order
## Taylor expansion and Monte Carlo simulation.
plot(DNase1$conc, DNase1$density)
```

```
lines(DNase1$conc, fitted(fm3DNase1), lwd = 2, col = 1)
points(CONC, PROP3$summary[, 1], col = 2, pch = 16)
lines(CONC, PROP3$summary[, 5], col = 2)
lines(CONC, PROP3$summary[, 6], col = 2)
lines(CONC, PROP3$summary[, 11], col = 4)
lines(CONC, PROP3$summary[, 12], col = 4)

## Using a sequence of predictor values with error.
PROP4 <- predictNLS(fm3DNase1, newdata = data.frame(conc = 1:5),
                    newerror = data.frame(conc = (1:5)/10))
PROP4$summary

## Using multiple predictor values.
## 1: Setup of response values with gaussian error of 10%.
x <- seq(1, 10, by = 0.01)
y <- seq(10, 1, by = -0.01)
a <- 2
b <- 5
c <- 10
z <- a * exp(b * x)^sin(y/c)
z <- z + sapply(z, function(x) rnorm(1, x, 0.10 * x))

## 2: Fit 'nls' model.
MOD <- nls(z ~ a * exp(b * x)^sin(y/c),
           start = list(a = 2, b = 5, c = 10))

## 3: Single newdata without errors.
DAT1 <- data.frame(x = 4, y = 3)
PROP5 <- predictNLS(MOD, newdata = DAT1)
PROP5$summary

## 4: Single newdata with errors.
DAT2 <- data.frame(x = 4, y = 3)
ERR2 <- data.frame(x = 0.2, y = 0.1)
PROP6 <- predictNLS(MOD, newdata = DAT2, newerror = ERR2)
PROP6$summary

## 5: Multiple newdata with errors.
DAT3 <- data.frame(x = 1:4, y = 3)
ERR3 <- data.frame(x = rep(0.2, 4), y = seq(1:4)/10)
PROP7 <- predictNLS(MOD, newdata = DAT3, newerror = ERR3)
PROP7$summary

## 6: Linear model to compare conf/pred intervals.
set.seed(123)
X <- 1:20
Y <- 3 + 2 * X + rnorm(20, 0, 2)
plot(X, Y)
LM <- lm(Y ~ X)
NLS <- nlsLM(Y ~ a + b * X, start = list(a = 3, b = 2))
predict(LM, newdata = data.frame(X = 14.5), interval = "conf")
predictNLS(NLS, newdata = data.frame(X = 14.5), interval = "conf")$summary
predict(LM, newdata = data.frame(X = 14.5), interval = "pred")
```

```
predictNLS(NLS, newdata = data.frame(X = 14.5), interval = "pred")$summary

## 7: compare to 'predFit' function of 'investr' package.
## Same results when using only first-order Taylor expansion.
require(investr)
data(Puromycin, package = "datasets")
Puromycin2 <- Puromycin[Puromycin$state == "treated", ][, 1:2]
Puro.nls <- nls(rate ~ Vm * conc/(K + conc), data = Puromycin2,
                start = c(Vm = 200, K = 0.05))
PRED1 <- predFit(Puro.nls, interval = "prediction")
PRED2 <- predictNLS(Puro.nls, interval = "prediction", second.order = FALSE, do.sim = FALSE)
all.equal(PRED1[, "lwr"], PRED2$summary[, 5]) # => TRUE

## End(Not run)
```

`propagate` *Propagation of uncertainty using higher-order Taylor expansion and Monte Carlo simulation*

Description

A general function for the calculation of uncertainty propagation by first-/second-order Taylor expansion and Monte Carlo simulation including covariances. Input data can be any symbolic/numeric differentiable expression and data based on summaries (mean & s.d.) or sampled from distributions. Uncertainty propagation is based completely on matrix calculus accounting for full covariance structure. Monte Carlo simulation is conducted using a multivariate t-distribution with covariance structure. Propagation confidence intervals are calculated from the expanded uncertainties by means of the degrees of freedom obtained from `WelchSatter`, or from the $[\frac{\alpha}{2}, 1 - \frac{\alpha}{2}]$ quantiles of the MC evaluations.

Usage

```
propagate(expr, data, second.order = TRUE, do.sim = TRUE, cov = TRUE,
          df = NULL, nsim = 1000000, alpha = 0.05, ...)
```

Arguments

`expr`	an expression, such as `expression(x/y)`.
`data`	a dataframe or matrix containing either a) the means μ_i, standard deviations σ_i and degrees of freedom ν_i (optionally) in the first, second and third (optionally) row, or b) sampled data generated from any of R's `distributions` or those implemented in this package (`rDistr`). If `nrow(data)` > 3, sampled data is assumed. The column names must match the variable names.
`second.order`	logical. If `TRUE`, error propagation will be calculated with first- and second-order Taylor expansion. See 'Details'.
`do.sim`	logical. Should Monte Carlo simulation be applied?
`cov`	logical or variance-covariance matrix with the same column names as `data`. See 'Details'.

`df`	an optional scalar with the total degrees of freedom ν_{tot} of the system.
`nsim`	the number of Monte Carlo simulations to be performed, minimum is 10000.
`alpha`	the 1 - confidence level.
`...`	other parameters to be supplied to future methods.

Details

The implemented methods are:

1) **Monte Carlo simulation:**
For each variable m in `data`, simulated data $X = [x_1, x_2, \ldots, x_n]$ with n = `nsim` samples is generated from a multivariate t-distribution $X_{m,n} \sim t(\mu, \Sigma, \nu)$ using means μ_i and covariance matrix $\mathbf{\Sigma}$ constructed from the standard deviations σ_i of each variable. All data is coerced into a new dataframe that has the same covariance structure as the initial `data`: $\mathbf{\Sigma}(\texttt{data}) = \mathbf{\Sigma}(X_{m,n})$. Each row $i = 1, \ldots, n$ of the simulated dataset $X_{m,n}$ is evaluated with `expr`, $y_i = f(x_{m,i})$, and summary statistics (mean, sd, median, mad, quantile-based confidence interval based on $[\frac{\alpha}{2}, 1 - \frac{\alpha}{2}]$) are calculated on y.

2) **Error propagation:**
The propagated error is calculated by first-/second-order Taylor expansion accounting for full covariance structure using matrix algebra.
The following transformations based on two variables x_1, x_2 illustrate the equivalence of the matrix-based approach with well-known classical notations:
First-order mean: $\mathrm{E[y]} = \mathrm{f(\bar{x}_i)}$
First-order variance: $\sigma_y^2 = \nabla\mathbf{\Sigma}\nabla^T$:

$$[\mathrm{j_1}\ \mathrm{j_2}] \begin{bmatrix} \sigma_1^2 & \sigma_1\sigma_2 \\ \sigma_2\sigma_1 & \sigma_2^2 \end{bmatrix} \begin{bmatrix} \mathrm{j_1} \\ \mathrm{j_2} \end{bmatrix} = \mathrm{j_1}^2\sigma_1^2 + 2\mathrm{j_1}\mathrm{j_2}\sigma_1\sigma_2 + \mathrm{j_2}^2\sigma_2^2$$

$$= \underbrace{\sum_{i=1}^{2} \mathrm{j_i}^2\sigma_\mathrm{i}^2 + 2\sum_{\substack{\mathrm{i}=1 \\ \mathrm{i}\neq\mathrm{k}}}^{2}\sum_{\substack{\mathrm{k}=1 \\ \mathrm{k}\neq\mathrm{i}}}^{2} \mathrm{j_i}\mathrm{j_k}\sigma_{\mathrm{ik}}}_{\text{classical notation}} = \frac{1}{1!}\left(\sum_{i=1}^{2} \frac{\partial f}{\partial x_i}\sigma_i\right)^2$$

Second-order mean: $\mathrm{E[y]} = \mathrm{f(\bar{x}_i)} + \frac{1}{2}\mathrm{tr}(\mathbf{H\Sigma})$:

$$\frac{1}{2}\mathrm{tr}\begin{bmatrix} \mathrm{h_1} & \mathrm{h_2} \\ \mathrm{h_3} & \mathrm{h_4} \end{bmatrix}\begin{bmatrix} \sigma_1^2 & \sigma_1\sigma_2 \\ \sigma_2\sigma_1 & \sigma_2^2 \end{bmatrix} = \frac{1}{2}\mathrm{tr}\begin{bmatrix} \mathrm{h_1}\sigma_1^2 + \mathrm{h_2}\sigma_1\sigma_2 & \mathrm{h_1}\sigma_1\sigma_2 + \mathrm{h_2}\sigma_2^2 \\ \mathrm{h_3}\sigma_1^2 + \mathrm{h_4}\sigma_1\sigma_2 & \mathrm{h_3}\sigma_1\sigma_2 + \mathrm{h_4}\sigma_2^2 \end{bmatrix}$$

$$= \frac{1}{2}(\mathrm{h_1}\sigma_1^2 + \mathrm{h_2}\sigma_1\sigma_2 + \mathrm{h_3}\sigma_1\sigma_2 + \mathrm{h_4}\sigma_2^2) = \frac{1}{2!}\left(\sum_{\mathrm{i}=1}^{2} \frac{\partial}{\partial \mathrm{x_i}}\sigma_\mathrm{i}\right)^2 f$$

Second-order variance: $\sigma_y^2 = \nabla\mathbf{\Sigma}\nabla^T + \frac{1}{2}\mathrm{tr}(\mathbf{H\Sigma H\Sigma})$:

$$\frac{1}{2}\mathrm{tr}\begin{bmatrix} \mathrm{h_1} & \mathrm{h_2} \\ \mathrm{h_3} & \mathrm{h_4} \end{bmatrix}\begin{bmatrix} \sigma_1^2 & \sigma_1\sigma_2 \\ \sigma_2\sigma_1 & \sigma_2^2 \end{bmatrix}\begin{bmatrix} \mathrm{h_1} & \mathrm{h_2} \\ \mathrm{h_3} & \mathrm{h_4} \end{bmatrix}\begin{bmatrix} \sigma_1^2 & \sigma_1\sigma_2 \\ \sigma_2\sigma_1 & \sigma_2^2 \end{bmatrix} = \ldots$$

$$= \frac{1}{2}(\mathrm{h_1}^2\sigma_1^4 + 2\mathrm{h_1}\mathrm{h_2}\sigma_1^3\sigma_2 + 2\mathrm{h_1}\mathrm{h_3}\sigma_1^3\sigma_2 + \mathrm{h_2}^2\sigma_1^2\sigma_2^2 + 2\mathrm{h_2}\mathrm{h_3}\sigma_1^2\sigma_2^2 + \mathrm{h_3}^2\sigma_1^2\sigma_2^2 + 2\mathrm{h_1}\mathrm{h_4}\sigma_1^2\sigma_2^2$$

$$+2h_2h_4\sigma_1\sigma_2^3 + 2h_3h_4\sigma_1\sigma_2^3 + {h_4}^2\sigma_2^4 = \frac{1}{2}(h_1\sigma_1^2 + h_2\sigma_1\sigma_2 + h_3\sigma_1\sigma_2 + h_4\sigma_2^2)^2$$

$$= \frac{1}{2!}\left(\left(\sum_{i=1}^{2}\frac{\partial}{\partial x_i}\sigma_i\right)^2 f\right)^2$$

with $\mathrm{E}(y)$ = expectation of y, $\sigma_{\mathbf{y}}^{\mathbf{2}}$ = variance of y, ∇ = the p x n gradient matrix with all partial first derivatives j_i, $\mathbf{\Sigma}$ = the p x p covariance matrix, $\mathbf{H}$ the Hessian matrix with all partial second derivatives h_i, σ_i = the uncertainties and $\mathrm{tr}(\cdot)$ = the trace (sum of diagonal) of a matrix. Note that because the Hessian matrices are symmetric, $h_2 = h_3$. For a detailed derivation, see 'References'.
The second-order Taylor expansion corrects for bias in nonlinear expressions as the first-order Taylor expansion assumes linearity around $\bar{x}_i$. There is also a Python library available for second-order error propagation ('soerp', https://pypi.python.org/pypi/soerp). The 'propagate' package gives **exactly** the same results, see last example under "Examples".
Depending on the input expression, the uncertainty propagation may result in an error that is not normally distributed. The Monte Carlo simulation, starting with a symmetric t-distributions of the variables, can clarify this. For instance, a high tendency from deviation of normality is encountered in formulas in which the error of the denominator is relatively large or in exponential models with a large error in the exponent.

For setups in which there is no symbolic derivation possible (i.e. `e <- expression(abs(x))` => "Function 'abs' is not in the derivatives table") the function automatically switches from symbolic (using `makeGrad` or `makeHess`) to numeric (`numGrad` or `numHess`) differentiation.

The function will try to evaluate the expression in an environment using `eval` which results in a significant speed enhancement (~ 10-fold). If that fails, evaluation is done over the rows of the simulated data using `apply`.

`cov` is used in the following ways:
1) If μ_i, σ_i are supplied, a covariance matrix is built with diagonals σ_i^2, independent of `cov = TRUE, FALSE`.
2) When simulated data is supplied, a covariance matrix is constructed that either has (`cov = TRUE`) or has not (`cov = FALSE`) off-diagonal covariances.
3) The user can supply an own covariance matrix Σ, with the same column/row names as in `data`.

The expanded uncertainty used for constructing the confidence interval is calculated from the Welch-Satterthwaite degrees of freedom ν_{WS} of the `WelchSatter` function.

Value

A list with the following components:

`gradient`	the symbolic gradient vector ∇ of partial first-order derivatives.
`evalGrad`	the evaluated gradient vector ∇ of partial first-order derivatives, also known as the "sensitivity". See `summary.propagate`.
`hessian`	the symbolic Hessian matrix $\mathbf{H}$ of partial second-order derivatives.
`evalHess`	the evaluated Hessian matrix $\mathbf{H}$ of partial second-order derivatives.
`rel.contr`	the relative contribution matrix, see `summary.propagate`.

covMat	the covariance matrix $\mathbf{\Sigma}$ used for Monte Carlo simulation and uncertainty propagation.
ws.df	the Welch-Satterthwaite degrees of freedom ν_{ws}, as obtained from WelchSatter.
k	the coverage factor k, as calculated by $t(1-(\alpha/2),\nu_{\mathrm{ws}})$.
u.exp	the expanded uncertainty, $k\sigma(y)$, where $\sigma(y)$ is derived either from the second-order uncertainty, if successfully calculated, or first-order otherwise.
resSIM	a vector containing the nsim values obtained from the row-wise expression evaluations $f(x_{m,i})$ of the simulated data in datSIM.
datSIM	a vector containing the nsim simulated multivariate values for each variable in column format.
prop	a summary vector containing first-/second-order expectations and uncertainties as well as the confidence interval based on alpha.
sim	a summary vector containing the mean, standard deviation, median, MAD as well as the confidence interval based on alpha.
expr	the original expression expr.
data	the original data data.
alpha	the otiginal alpha.

Author(s)

Andrej-Nikolai Spiess

References

Error propagation (in general):
An Introduction to error analysis.
Taylor JR.
University Science Books (1996), New York.

Evaluation of measurement data - Guide to the expression of uncertainty in measurement.
JCGM 100:2008 (GUM 1995 with minor corrections).
http://www.bipm.org/utils/common/documents/jcgm/JCGM_100_2008_E.pdf.

Evaluation of measurement data - Supplement 1 to the Guide to the expression of uncertainty in measurement - Propagation of distributions using a Monte Carlo Method.
JCGM 101:2008.
http://www.bipm.org/utils/common/documents/jcgm/JCGM_101_2008_E.pdf.

Higher-order Taylor expansion:
On higher-order corrections for propagating uncertainties.
Wang CM & Iyer HK.
Metrologia (2005), **42**: 406-410.

Propagation of uncertainty: Expressions of second and third order uncertainty with third and fourth moments.
Mekid S & Vaja D.
Measurement (2008), **41**: 600-609.

Matrix algebra for error propagation:
An Introduction to Error Propagation: Derivation, Meaning and Examples of Equation Cy = Fx-CxFx^t.
www.nada.kth.se/~kai-a/papers/arrasTR-9801-R3.pdf.

Second order nonlinear uncertainty modeling in strapdown integration using MEMS IMUs.
Zhang M, Hol JD, Slot L, Luinge H.
2011 Proceedings of the 14th International Conference on Information Fusion (FUSION) (2011).

Uncertainty propagation in non-linear measurement equations.
Mana G & Pennecchi F.
Metrologia (2007), **44**: 246-251.

A compact tensor algebra expression of the law of propagation of uncertainty.
Bouchot C, Quilantan JLC, Ochoa JCS.
Metrologia (2011), **48**: L22-L28.

Nonlinear error propagation law.
Kubacek L.
Appl Math (1996), **41**: 329-345.

Monte Carlo simulation (normal- and t-distribution):
MUSE: computational aspects of a GUM supplement 1 implementation.
Mueller M, Wolf M, Roesslein M.
Metrologia (2008), **45**: 586-594.

Copulas for uncertainty analysis.
Possolo A.
Metrologia (2010), **47**: 262-271.

Multivariate normal distribution:
Stochastic Simulation.
Ripley BD.
Stochastic Simulation (1987). Wiley. Page 98.

Testing for normal distribution:
Testing for Normality.
Thode Jr. HC.
Marcel Dekker (2002), New York.

Approximating the Shapiro-Wilk W-test for non-normality.
Royston P.
Stat Comp (1992), **2**: 117-119.

Examples

```
## In these examples, 'nsim = 100000' to save
## Rcmd check time (CRAN). It is advocated
## to use at least 'nsim = 1000000' though...

## Example without given degrees-of-freedom.
EXPR1 <- expression(x/y)
x <- c(5, 0.01)
y <- c(1, 0.01)
DF1 <- cbind(x, y)
RES1 <- propagate(expr = EXPR1, data = DF1, type = "stat",
```

```
                  do.sim = TRUE, verbose = TRUE,
                  nsim = 100000)
RES1

## Same example with given degrees-of-freedom
## => third row in input 'data'.
EXPR2 <- expression(x/y)
x <- c(5, 0.01, 12)
y <- c(1, 0.01, 5)
DF2 <- cbind(x, y)
RES2 <- propagate(expr = EXPR2, data = DF2, type = "stat",
                  do.sim = TRUE, verbose = TRUE,
                  nsim = 100000)
RES2

## With the 'summary' function, we can get the
## Welch-Satterthwaite DF's, coverage, expanded uncertainty,
## Gradient and Hessian matrix etc.
summary(RES2)

## Example using a recursive function:
## no Taylor expansion possible, only Monte-Carlo.
a <- c(5, 0.1)
b <- c(100, 2)
DAT <- cbind(a, b)

f <- function(a, b) {
  N <- 0
  for (i in 1:100) {
    N <- N + i * log(a) + b^(1/i)
  }
  return(N)
}

propagate(f, DAT, nsim = 100000)

## Not run:
################# GUM 2008 (1) ########################
## Example in Annex H.1 from the GUM 2008 manual
## (see 'References'), an end gauge calibration
## study. We use only first-order error propagation,
## with total df = 16 and alpha = 0.01,
## as detailed in GUM H.1.6.
EXPR3 <- expression(ls + d - ls * (da * the + as * dt))
ls <- c(50000623, 25)
d <- c(215, 9.7)
da <- c(0, 0.58E-6)
the <- c(-0.1, 0.41)
as <- c(11.5E-6, 1.2E-6)
dt <- c(0, 0.029)
DF3 <- cbind(ls, d, da, the, as, dt)
RES3 <- propagate(expr = EXPR3, data = DF3, second.order = FALSE,
                  df = 16, alpha = 0.01)
```

```
RES3
## propagate: sd.1 = 31.71
## GUM H.1.4/H.6c: u = 32

## Expanded uncertainty, from summary function.
summary(RES3)
## propagate: 92.62
## GUM H.1.6: 93

## Proof that covariance of Monte-Carlo
## simulated dataset is "fairly"" the same
## as from initial data.
RES3$covMat
cov(RES3$datSIM)
all.equal(RES3$covMat, cov(RES3$datSIM))

## Now using second-order Taylor expansion.
RES4 <- propagate(expr = EXPR3, data = DF3)
RES4
## propagate: sd.2 = 33.91115
## GUM H.1.7: u = 34.
## Also similar to the non-matrix-based approach
## in Wang et al. (2005, page 408): u1 = 33.91115.
## NOTE: After second-order correction ("sd.2"),
## uncertainty is more similar to the uncertainty
## obtained from Monte Carlo simulation!

#################### GUM 2008 (2) #################
## Example in Annex H.2 from the GUM 2008 manual
## (see 'References'), simultaneous resistance
## and reactance measurement.
data(H.2)

## This gives exactly the means, uncertainties and
## correlations as given in Table H.2:
colMeans(H.2)
sqrt(colVarsC(H.2))/sqrt(5)
cor(H.2)

## H.2.3 Approach 1 using mean values and
## standard uncertainties:
EXPR6a <- expression((V/I) *  cos(phi)) ## R
EXPR6b <- expression((V/I) *  sin(phi)) ## X
EXPR6c <- expression(V/I) ## Z
MEAN6 <- colMeans(H.2)
SD6 <- sqrt(colVarsC(H.2))
DF6 <- rbind(MEAN6, SD6)
COV6ab <- cov(H.2) ## covariance matrix of V, I, phi
COV6c <- cov(H.2[, 1:2])  ## covariance matrix of V, I

RES6a <- propagate(expr = EXPR6a, data = DF6, cov = COV6ab)
RES6b <- propagate(expr = EXPR6b, data = DF6, cov = COV6ab)
RES6c <- propagate(expr = EXPR6c, data = DF6[, 1:2],
```

```
                  cov = COV6c)

## This gives exactly the same values of mean and sd/sqrt(5)
## as given in Table H.4.
RES6a$prop # 0.15892/sqrt(5) = 0.071
RES6b$prop # 0.66094/sqrt(5) = 0.296
RES6c$prop # 0.52846/sqrt(5) = 0.236

######### GUM 2008 Supplement 1 (1) ########################
## Example from 9.2.2 of the GUM 2008 Supplement 1
## (see 'References'), normally distributed input
## quantities. Assign values as in 9.2.2.1.
EXPR7 <- expression(X1 + X2 + X3 + X4)
X1 <- c(0, 1)
X2 <- c(0, 1)
X3 <- c(0, 1)
X4 <- c(0, 1)
DF7 <- cbind(X1, X2, X3, X4)
RES7 <- propagate(expr = EXPR7, data = DF7, nsim = 1E5)
## This will give exactly the same results as in
## 9.2.2.6, Table 2.
RES7

######### GUM 2008 Supplement 1 (2) ########################
## Example from 9.3 of the GUM 2008 Supplement 1
## (see 'References'), mass calibration.
## Formula 24 in 9.3.1.3 and values as in 9.3.1.4, Table 5.
EXPR8 <- expression((Mrc + dMrc) * (1 + (Pa - Pa0) * ((1/Pw) - (1/Pr))) - Mnom)
Mrc <- rnorm(1E5, 100000, 0.050)
dMrc <- rnorm(1E5, 1.234, 0.020)
Pa <- runif(1E5, 1.10, 1.30)  ## E(Pa) = 1.2, (b-a)/2 = 0.1
Pw <- runif(1E5, 7000, 9000)  ## E(Pw) = 8000, (b-a)/2 = 1000
Pr <- runif(1E5, 7950, 8050) ## E(Pr) = 8000, (b-a)/2 = 50
Pa0 <- 1.2
Mnom <- 100000
DF8 <- cbind(Mrc, dMrc, Pa, Pw, Pr, Pa0, Mnom)
RES8 <- propagate(expr = EXPR8, data = DF8, nsim = 1E5)
## This will give exactly the same results as in
## 9.3.2.3, Table 6
RES8
RES8

######### GUM 2008 Supplement 1 (3) ########################
## Example from 9.4 of the GUM 2008 Supplement 1
## (see 'References'), comparioson loss in microwave
## power meter calibration, zero covariance.
## Formula 28 in 9.4.1.5 and values as in 9.4.1.7.
EXPR9 <- expression(X1^2 - X2^2)
X1 <- c(0.050, 0.005)
X2 <- c(0, 0.005)
DF9 <- cbind(X1, X2)
RES9a <- propagate(expr = EXPR9, data = DF9, nsim = 1E5)
## This will give exactly the same results as in
```

```
## 9.4.2.2.7, Table 8, x1 = 0.050.
RES9a

## Using covariance matrix with r(x1, x2) = 0.9
## We convert to covariances using cor2cov.
COR9 <- matrix(c(1, 0.9, 0.9, 1), nrow = 2)
COV9 <- cor2cov(COR9, c(0.005^2, 0.005^2))
colnames(COV9) <- c("X1", "X2")
rownames(COV9) <- c("X1", "X2")
RES9b <- propagate(expr = EXPR9, data = DF9, cov = COV9)
## This will give exactly the same results as in
## 9.4.3.2.1, Table 9, x1 = 0.050.
RES9b

######### GUM 2008 Supplement 1 (4) ######################
## Example from 9.5 of the GUM 2008 Supplement 1
## (see 'References'), gauge block calibration.
## Assignment of PDF's as in Table 10 of 9.5.2.1.
EXPR10 <- expression(Ls + D + d1 + d2 - Ls *(da *(t0 + Delta) + as * dt) - Lnom)
Lnom <- 50000000
Ls <- propagate:::rst(1000000, mean = 50000623, sd  = 25, df = 18)
D <- propagate:::rst(1000000, mean = 215, sd = 6, df = 25)
d1 <- propagate:::rst(1000000, mean = 0, sd = 4, df = 5)
d2 <- propagate:::rst(1000000, mean = 0, sd = 7, df = 8)
as <- runif(1000000, 9.5E-6, 13.5E-6)
t0 <- rnorm(1000000, -0.1, 0.2)
Delta <- propagate:::rarcsin(1000000, -0.5, 0.5)
da <- propagate:::rctrap(1000000, -1E-6, 1E-6, 0.1E-6)
dt <- propagate:::rctrap(1000000, -0.050, 0.050, 0.025)
DF10 <- cbind(Ls, D, d1, d2, as, t0, Delta, da, dt, Lnom)
RES10 <- propagate(expr = EXPR10, data = DF10, cov = FALSE, alpha = 0.01)
RES10
## This gives the same results as in 9.5.4.2, Table 11.
## However: results are exacter than in the GUM 2008
## manual, especially when comparing sd(Monte Carlo) with sd.2!
## GUM 2008 gives 32 and 36, respectively.
RES10

########## Comparison to Pythons 'soerp' ##################
## Exactly the same results as under
## https://pypi.python.org/pypi/soerp !
EXPR11 <- expression(C * sqrt((520 * H * P)/(M *(t + 460))))
H <- c(64, 0.5)
M <- c(16, 0.1)
P <- c(361, 2)
t <- c(165, 0.5)
C <- c(38.4, 0)
DAT11 <- makeDat(EXPR11)
RES11 <- propagate(expr = EXPR11, data = DAT11)
RES11

## End(Not run)
```

rDistr *Creating random samples from a variety of useful distributions*

Description

These are random sample generators for 22 different continuous distributions which are not readily available as Distributions in R. Some of them are implemented in other specialized packages (i.e. rsn in package 'sn' or rtrapezoid in package 'trapezoid'), but here they are collated in a way that makes them easily accessible for Monte Carlo-based uncertainty propagation.

Details

Random samples can be drawn from the following distributions:
1) Skewed-normal distribution: `propagate:::rsn(n, location = 0, scale = 1, shape = 0)`
2) Generalized normal distribution: `propagate:::rgnorm(n, alpha = 1, xi = 1, kappa = -0.1)`
3) Scaled and shifted t-distribution: `propagate:::rst(n, mean = 0, sd = 1, df = 2)`
4) Gumbel distribution: `propagate:::rgumbel(n, location = 0, scale = 1)`
5) Johnson SU distribution: `propagate:::rJSU(n, xi = 0, lambda = 1, gamma = 1, delta = 1)`
6) Johnson SB distribution: `propagate:::rJSB(n, xi = 0, lambda = 1, gamma = 1, delta = 1)`
7) 3P Weibull distribution: `propagate:::rweibull2(n, location = 0, shape = 1, scale = 1)`
8) 4P Beta distribution: `propagate:::rbeta2(n, alpha1 = 1, alpha2 = 1, a = 0, b = 0)`
9) Triangular distribution: `propagate:::rtriang(n, a = 0, b = 1, c = 0.5)`
10) Trapezoidal distribution: `propagate:::rtrap(n, a = 0, b = 1, c = 2, d = 3)`
11) Laplacian distribution: `propagate:::rlaplace(n, mean = 0, sigma = 1)`
12) Arcsine distribution: `propagate:::rarcsin(n, a = 2, b = 1)`
13) von Mises distribution: `propagate:::rmises(n, mu = 1, kappa = 3)`
14) Curvilinear Trapezoidal distribution: `propagate:::rctrap(n, a = 0, b = 1, d = 0.1)`
15) Generalized trapezoidal distribution:
`propagate:::rgtrap(n, min = 0, mode1 = 1/3, mode2 = 2/3, max = 1, n1 = 2, n3 = 2, alpha = 1)`
16) Inverse Gaussian distribution: `propagate:::rinvgauss(n, mean = 1, dispersion = 1)` 17) Generalized Extreme Value distribution: `propagate:::rgevd(n, loc = 0, scale = 1, shape = 0)`
with n = number of samples.
18) Inverse Gamma distribution: `propagate:::rinvgamma(n, shape = 1, scale = 5)`
19) Rayleigh distribution: `propagate:::rrayleigh(n, mu = 1, sigma = 1)`
20) Burr distribution: `propagate:::rburr(n, k = 1)`
21) Chi distribution: `propagate:::rchi(n, nu = 5)`
22) Inverse Chi-Square distribution: `propagate:::rinvchisq(n, nu = 5)`
23) Cosine distribution: `propagate:::rcosine(n, mu = 5, sigma = 1)`

1) - 12), 17) - 22) use the inverse cumulative distribution function as mapping functions for runif (**Inverse Transform Method**):
(1) $U \sim \mathcal{U}(0,1)$
(2) $Y = F^{-1}(U, \beta)$

16) uses binomial selection from a χ^2-distribution.

13) - 15), 23) employ "Rejection Sampling" using a uniform envelope distribution (**Acceptance Rejection Method**):

(1) Find $F_{max} = \max(F([x_{min}, x_{max}], \beta)$
(2) $U_{max} = 1/(x_{max} - x_{min})$
(3) $A = F_{max}/U_{max}$
(4) $U \sim \mathcal{U}(0, 1)$
(5) $X \sim \mathcal{U}(x_{min}, x_{max})$
(6) $Y \iff U \leq A \cdot \mathcal{U}(X, x_{min}, x_{max})/F(X, \beta)$

These four distributions are coded in a vectorized approach and are hence not much slower than implementations in C/C++ (0.2 - 0.5 sec for 100000 samples; 3 GHz Quadcore processor, 4 GByte RAM). The code for the random generators is in file "distr-samplers.R".

Value

A vector with n samples from the corresponding distribution.

Author(s)

Andrej-Nikolai Spiess

References

Inverse CDFs were taken from:
"The Ultimate Univariate Probability Distribution Explorer"
http://blog.wolfram.com/data/uploads/2013/02/ProbabilityDistributionExplorer.zip.

Rejection Sampling in R:
Rejection Sampling.
https://www.r-bloggers.com/rejection-sampling/.

An example of rejection sampling.
http://www.mas.ncl.ac.uk/~ndjw1/teaching/sim/reject/circ.html.

Rejection Sampling in general:
Non-uniform random variate generation.
Devroye L.
Springer-Verlag, New York (1986).

Distributions:
Continuous univariate distributions, Volume 1.
Johnson NL, Kotz S and Balakrishnan N.
Wiley Series in Probability and Statistics, 2.ed (2004).

See Also

See also propagate, in which GUM 2008 Supplement 1 examples use these distributions.

Examples

```
## Not run:
## First we create random samples from the
## von Mises distribution.
X <- propagate:::rmises(1000000, mu = 1, kappa = 2)
```

```
## then we fit all available distributions
## with 'fitDistr'.
fitDistr(X, nbin = 200)
## => von Mises wins! (lowest BIC)

## End(Not run)
```

`statVec` *Transform an input vector into one with defined mean and standard deviation*

Description

Transforms an input vector into one with defined μ and σ by using a scaled-and-shifted Z-transformation.

Usage

```
statVec(x, mean, sd)
```

Arguments

`x`	the input vector to be transformed.
`mean`	the desired mean of the created vector.
`sd`	the desired standard deviation of the created vector.

Details

Calculates vector V using a Z-transformation of the input vector X and subsequent scaling by `sd` and shifting by `mean`:

$$V = \frac{X - \mu_X}{\sigma_X} \cdot \mathrm{sd} + \mathrm{mean}$$

Value

A vector with defined μ and σ.

Author(s)

Andrej-Nikolai Spiess

Examples

```
## Create a 10-sized vector with mean = 10 and s.d. = 1.
x <- rnorm(10, 5, 2)
mean(x) ## => mean is not 5!
sd(x) ## => s.d. is not 2!

z <- statVec(x, 5, 2)
mean(z) ## => mean is 5!
sd(z) ## => s.d. is 2!
```

`stochContr` *Stochastic contribution analysis of Monte Carlo simulation-derived propagated uncertainty*

Description

Conducts a "stochastic contribution analysis" by calculating the change in propagated uncertainty when each of the simulated variables is kept constant at its mean, i.e. the uncertainty is removed.

Usage

```
stochContr(prop, plot = TRUE)
```

Arguments

`prop`	a `propagate` object.
`plot`	logical. If `TRUE`, a boxplot with the original and mean-value propagated distribution.

Details

This function takes the Monte Carlo simulated data X_n from a `propagate` object (`...$datSIM`), sequentially substitutes each variable β_i by its mean $\bar{\beta}_i$ and then re-evaluates the output distribution $Y_n = f(\beta, X_n)$. Optional boxplots are displayed that compare the original $Y_n(\text{orig})$ to those obtained from removing σ from each β_i. Finally, the relative contribution C_i for all β_i is calculated by $C_i = \sigma(Y_n(\text{orig})) - \sigma(Y_n)$, and divided by its sum so that $\sum_{i=1}^{n} C_i = 1$.

Value

The relative contribution C_i for all variables.

Author(s)

Andrej-Nikolai Spiess

Examples

```
a <- c(15, 1)
b <- c(100, 5)
c <- c(0.5, 0.02)
DAT <- cbind(a, b, c)
EXPR <- expression(a * b^sin(c))
RES <- propagate(EXPR, DAT, nsim = 100000)
stochContr(RES)
```

`summary.propagate` *Summary function for 'propagate' objects*

Description

Provides a printed summary of the results obtained by `propagate`, such as statistics of the first/second-order uncertainty propagation, Monte Carlo simulation, the covariance matrix, symbolic as well as evaluated versions of the Gradient ("sensitivity") and Hessian matrices, relative contributions, the coverage factor and the Welch-Satterthwaite degrees of freedom. If `do.sim = TRUE` was set in `propagate`, skewness/kurtosis and Shapiro-Wilks/Kolmogorov-Smirnov tests for normality are calculated on the Monte-Carlo evaluations.

Usage

```
## S3 method for class 'propagate'
summary(object, ...)
```

Arguments

`object`	an object returned from `propagate`.
`...`	other parameters for future methods.

Details

Calculates the "sensitivity"" S_i of each variable x_i to the propagated uncertainty, as defined in the *Expression of the Uncertainty of Measurement in Calibration, Eqn 4.2, page 9* (see 'References'):

$$S_i = \text{eval}\left(\frac{\partial f}{\partial x_i}\right)$$

The "contribution" matrix is then $\mathbf{C} = \mathbf{S}\mathbf{S}^T\mathbf{\Sigma}$, where $\mathbf{\Sigma}$ is the covariance matrix. In the implementation here, the "relative contribution" matrix $\mathbf{C}_{\text{rel}}$ is rescaled to sum up to 1.

Value

A printed output with the items listed in 'Description'.

Author(s)

Andrej-Nikolai Spiess

References

Expression of the Uncertainty of Measurement in Calibration.
European Cooperation for Accreditation (EA-4/02), 1999.

Examples

```
EXPR1 <- expression(x^2 * sin(y))
x <- c(5, 0.01)
y <- c(1, 0.01)
DF1 <- cbind(x, y)
RES1 <- propagate(expr = EXPR1, data = DF1, type = "stat",
                  do.sim = TRUE, verbose = TRUE, nsim = 100000)
summary(RES1)
```

WelchSatter — *Welch-Satterthwaite approximation to the 'effective degrees of freedom'*

Description

Calculates the Welch-Satterthwaite approximation to the 'effective degrees of freedom' by using the samples' uncertainties and degrees of freedoms, as described in Welch (1947) and Satterthwaite (1946). External sensitivity coefficients can be supplied optionally.

Usage

```
WelchSatter(ui, ci = NULL, df = NULL, dftot = NULL, uc = NULL, alpha = 0.05)
```

Arguments

ui	the uncertainties u_i for each variable x_i.
ci	the sensitivity coefficients $c_i = \partial y / \partial x_i$.
df	the degrees of freedom for the samples, ν_i.
dftot	an optional known total degrees of freedom for the system, ν_{tot}. Overrides the internal calculation of ν_{ws}.
uc	the combined uncertainty, u(y).
alpha	the significance level for the t-statistic. See 'Details'.

Details

$$\nu_{\mathrm{eff}} \approx \frac{u(y)^4}{\sum_{i=1}^{n} \frac{(c_i u_i)^4}{\nu_i}}, \quad k = t(1-(\alpha/2), \nu_{\mathrm{eff}}), \quad u_{\mathrm{exp}} = k u(y)$$

Value

A list with the following items:

`ws.df`	the 'effective degrees of freedom'.
`k`	the coverage factor for calculating the expanded uncertainty.
`u.exp`	the expanded uncertainty u_{exp}.

Author(s)

Andrej-Nikolai Spiess

References

An Approximate Distribution of Estimates of Variance Components.
Satterthwaite FE.
Biometrics Bulletin (1946), **2**: 110-114.

The generalization of "Student's" problem when several different population variances are involved.
Welch BL.
Biometrika (1947), **34**: 28-35.

Examples

```
## Taken from GUM H.1.6, 4).
WelchSatter(ui = c(25, 9.7, 2.9, 16.6), df = c(18, 25.6, 50, 2), uc = 32, alpha = 0.01)
```

Index

∗ **algebra**
bigcor, 2
cor2cov, 4
fitDistr, 7
interval, 12
makeDat, 14
makeDerivs, 15
mixCov, 18
moments, 20
numDerivs, 21
predictNLS, 23
propagate, 27
rDistr, 36
statVec, 38
stochContr, 39
WelchSatter, 41
∗ **array**
makeDerivs, 15
mixCov, 18
moments, 20
numDerivs, 21
predictNLS, 23
propagate, 27
∗ **datasets**
datasets, 5
∗ **matrix**
bigcor, 2
cor2cov, 4
interval, 12
statVec, 38
stochContr, 39
WelchSatter, 41
∗ **models**
plot.propagate, 22
summary.propagate, 40
∗ **multivariate**
bigcor, 2
cor2cov, 4
interval, 12
makeDerivs, 15
mixCov, 18
moments, 20
numDerivs, 21
predictNLS, 23
propagate, 27
statVec, 38
stochContr, 39
WelchSatter, 41
∗ **nonlinear**
plot.propagate, 22
summary.propagate, 40
∗ **univariate**
fitDistr, 7
makeDat, 14
rDistr, 36
∗ **univar**
matrixStats, 17

apply, *29*

BIC, *7*, *9*
bigcor, 2

cbind, *14*
colVarsC *(*matrixStats*)*, 17
cor, *2, 3*
cor2cov, 4
cov, *2*
cov2cor, *4*

datasets, 5
Distributions, *36*
distributions, *27*

environment, *21*
eval, *29*
evalDerivs *(*makeDerivs*)*, 15

fitDistr, 7

H.2 (datasets), 5
H.3 (datasets), 5
H.4 (datasets), 5
hist, *22*

interval, 11

kurtosis (moments), 20

makeDat, 14
makeDerivs, 15
makeGrad, *29*
makeGrad (makeDerivs), 15
makeHess, *29*
makeHess (makeDerivs), 15
matrix, *3*
matrixStats, 17
mixCov, 18
moments, 20

nls, *23*
nls.lm, *8*, *10*
numDerivs, 21
numGrad, *29*
numGrad (numDerivs), 21
numHess, *29*
numHess (numDerivs), 21

plot.propagate, 22
predict.nls, *23*
predictNLS, 23
propagate, *7*, *14, 15*, *18*, *22–24*, 27, *37*, *39, 40*

rDistr, *27*, 36
rowVarsC (matrixStats), 17
rtmvt, *24*
runif, *36*

skewness (moments), 20
statVec, 38
stochContr, 39
summary.propagate, *29*, 40

var, *17*

WelchSatter, *27*, *29, 30*, 41