Package 'sensitivity'

September 23, 2017

Version 1.15.0

Title Global Sensitivity Analysis of Model Outputs

Author Gilles Pujol, Bertrand Iooss, Alexandre Janon with contributions from Khalid Boumhaout, Sebastien Da Veiga, Thibault Delage, Jana Fruth, Laurent Gilquin, Joseph Guillaume, Loic Le Gratiet, Paul Lemaitre, Barry L. Nelson, Filippo Monari, Roelof Oomen, Bernardo Ramos, Olivier Roustant, Eunhye Song, Jeremy Staum, Taieb Touati, Frank Weber

Maintainer Bertrand Iooss <biooss@yahoo.fr>

Depends R (>= 3.0.0)

Imports boot, methods

Suggests condMVNorm, DiceDesign, DiceKriging, evd, fanovaGraph, ggplot2, ggExtra, gtools, igraph, ks, mc2d, mvtnorm, numbers, parallel, pracma, randtoolbox, reshape2, rgl

Description A collection of functions for factor screening, global sensitivity analysis and reliability sensitivity analysis. Most of the functions have to be applied on model with scalar output, but several functions support multi-dimensional outputs.

License GPL-2

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-09-23 21:58:36 UTC

R topics documented:

`sensitivity-package` *Sensitivity Analysis*

Description

Methods and functions for global sensitivity analysis.

Details

The **sensitivity** package implements some global sensitivity analysis methods:

- Linear regression coefficients: SRC and SRRC (`src`), PCC and PRCC (`pcc`);
- Bettonvil's sequential bifurcations (Bettonvil and Kleijnen, 1996) (`sb`);
- Morris's "OAT" elementary effects screening method (`morris`);
- Derivative-based Global Sensitivity Measures:

  - Poincare constants for Derivative-based Global Sensitivity Measures (DGSM) (Lamboni et al., 2013; Roustant et al., 2017) (PoincareConstant) and (PoincareOptimal),
  - Distributed Evaluation of Local Sensitivity Analysis (DELSA) (Rakovec et al., 2014) (delsa);
- Variance-based sensitivity indices (Sobol' indices):
  - Estimation of the Sobol' first order indices with with B-spline Smoothing (Ratto and Pagano, 2010) (sobolSmthSpl),
  - Monte Carlo estimation of Sobol' indices with independent inputs (also called pick-freeze method):
    * Sobol' scheme (Sobol, 1993) to compute the indices given by the variance decomposition up to a specified order (sobol),
    * Saltelli's scheme (Saltelli, 2002) to compute first order, second order and total indices (sobolSalt),
    * Saltelli's scheme (Saltelli, 2002) to compute first order and total indices (sobol2002),
    * Mauntz-Kucherenko's scheme (Sobol et al., 2007) to compute first order and total indices using improved formulas for small indices (sobol2007),
    * Jansen-Sobol's scheme (Jansen, 1999) to compute first order and total indices using improved formulas (soboljansen),
    * Martinez's scheme using correlation coefficient-based formulas (Martinez, 2011; Touati, 2016) to compute first order and total indices, associated with theoretical confidence intervals (sobolmartinez and soboltouati),
    * Janon-Monod's scheme (Monod et al., 2006; Janon et al., 2013) to compute first order indices with optimal asymptotic variance (sobolEff),
    * Mara's scheme (Mara and Joseph, 2008) to compute first order indices with a cost independent of the dimension, via a unique-matrix permutations (sobolmara),
    * Owen's scheme (Owen, 2013) to compute first order and total indices using improved formulas (via 3 input independent matrices) for small indices (sobolowen),
    * Total Interaction Indices using Liu-Owen's scheme (Liu and Owen, 2006) (sobolTIIlo) and pick-freeze scheme (Fruth et al., 2014) (sobolTIIpf),
  - Estimation of the Sobol' first order and total indices with Saltelli's so-called "extended-FAST" method (Saltelli et al., 1999) (fast99),
  - Estimation of the Sobol' first order and closed second order indices using replicated orthogonal array-based Latin hypecube sample (Tissot and Prieur, 2015) (sobolroalhs),
  - Sobol' indices estimation under inequality constraints (Gilquin et al., 2015) by extension of the replication procedure (Tissot and Prieur, 2015) (sobolroauc),
  - Estimation of the Sobol' first order and total indices with kriging-based global sensitivity analysis (Le Gratiet et al., 2014) (sobolGP);
- Variance-based sensitivity indices (Shapley effects and Sobol' indices, with independent or dependent inputs):
  - Estimation by examining all permutations of inputs (Song et al., 2016) (shapleyPermEx)
  - Estimation by randomly sampling permutations of inputs (Song et al., 2016) (shapleyPermRand)
- Support index functions (support) of Fruth et al. (2016);
- Sensitivity Indices based on Csiszar f-divergence (sensiFdiv) (particular cases: Borgonovo's indices and mutual-information based indices) and Hilbert-Schmidt Independence Criterion (sensiHSIC) of Da Veiga (2015);

- Reliability sensitivity analysis by the Perturbed-Law based Indices (PLI) of Lemaitre et al. (2015) and (PLIquantile) of Sueur et al. (2016, 2017);
- Sobol' indices for multidimensional outputs (sobolMultOut): Aggregated Sobol' indices (Lamboni et al., 2011; Gamboa et al., 2014) and functional (1D) Sobol' indices.

Moreover, some utilities are provided: standard test-cases (testmodels) and template file generation (template.replace).

Model managing

The **sensitivity** package has been designed to work either models written in R than external models such as heavy computational codes. This is achieved with the input argument model present in all functions of this package.

The argument model is expected to be either a funtion or a predictor (i.e. an object with a predict function such as lm).

- If `model = m` where `m` is a function, it will be invoked once by `y <- m(X)`.
- If `model = m` where `m` is a predictor, it will be invoked once by `y <- predict(m, X)`.

X is the design of experiments, i.e. a data.frame with p columns (the input factors) and n lines (each, an experiment), and y is the vector of length n of the model responses.

The model in invoked once for the whole design of experiment.

The argument model can be left to NULL. This is refered to as the decoupled approach and used with external computational codes that rarely run on the statistician's computer. See decoupling.

Author(s)

Gilles Pujol, Bertrand Iooss, Alexandre Janon with contributions from Paul Lemaitre for the PLI function, Laurent Gilquin for the sobolroalhs, sobolroauc and sobolSalt functions, Loic le Gratiet for the sobolGP function, Khalid Boumhaout, Taieb Touati and Bernardo Ramos for the sobolowen and soboltouati functions, Jana Fruth for the PoincareConstant, sobolTIIlo and sobolTIIpf functions, Sebastien Da veiga for the sensiFdiv and sensiHSIC functions, Joseph Guillaume for the delsa and parameterSets functions, Olivier Roustant for the PoincareOptimal and support functions, Eunhye Song, Barry L. Nelson and Jeremy Staum for the shapleyPermEx and shapleyPermRand functions, Filippo Monari for the (sobolSmthSpl) function, Frank Weber, Thibault Delage and Roelof Oomen.

(maintainer: Bertrand Iooss <biooss@yahoo.fr>)

References

R. Faivre, B. Iooss, S. Mahevas, D. Makowski, H. Monod, editors, 2013, *Analyse de sensibilite et exploration de modeles. Applications aux modeles environnementaux*, Editions Quae.

B. Iooss and A. Saltelli, 2017, *Introduction: Sensitivity analysis.* In: *Springer Handbook on Uncertainty Quantification*, R. Ghanem, D. Higdon and H. Owhadi (Eds), Springer. hrefhttp://link.springer.com/referenceworkentry
3-319-11259-6_31-1

B. Iooss and P. Lemaitre, 2015, *A review on global sensitivity analysis methods*. In *Uncertainty management in Simulation-Optimization of Complex Systems: Algorithms and Applications*, C. Meloni and G. Dellino (eds), Springer. https://hal.archives-ouvertes.fr/hal-00975701

A. Saltelli, K. Chan and E. M. Scott eds, 2000, *Sensitivity Analysis*, Wiley.

A. Saltelli et al., 2008, *Global Sensitivity Analysis: The Primer*, Wiley

---

`decoupling` *Decoupling Simulations and Estimations*

Description

`tell` and `ask` are S3 generic methods for decoupling simulations and sensitivity measures estimations. In general, they are not used by the end-user for a simple R model, but rather for an external computational code. Most of the sensitivity analyses objects of this package overload `tell`, whereas `ask` is overloaded for iterative methods only.

Usage

```
tell(x, y = NULL, ...)
ask(x, ...)
```

Arguments

`x`	a typed list storing the state of the sensitivity study (parameters, data, estimates), as returned by sensitivity analyses objects constructors, such as `src`, `morris`, etc.
`y`	a vector of model responses.
`...`	additional arguments, depending on the method used.

Details

When a sensitivity analysis method is called with no model (i.e. argument `model = NULL`), it generates an incomplete object `x` that stores the design of experiments (field `X`), allowing the user to launch "by hand" the corresponding simulations. The method `tell` allows to pass these simulation results to the incomplete object `x`, thereafter estimating the sensitivity measures.

When the method is iterative, the data to simulate are not stored in the sensitivity analysis object `x`, but generated at each iteration with the `ask` method; see for example `sb`.

Value

`tell` doesn't return anything. It computes the sensitivity measures, and stores them in the list `x`. **Side effect:** `tell` **modifies its argument** `x`**.**

`ask` returns the set of data to simulate.

Author(s)

Gilles Pujol

Examples

```
# Example of use of fast99 with "model = NULL"
x <- fast99(model = NULL, factors = 3, n = 1000,
            q = "qunif", q.arg = list(min = -pi, max = pi))
y <- ishigami.fun(x$X)
tell(x, y)
print(x)
plot(x)
```

---

delsa *Distributed Evaluation of Local Sensitivity Analysis*

Description

delsa implements Distributed Evaluation of Local Sensitivity Analysis to calculate first order parameter sensitivity at multiple locations in parameter space. The locations in parameter space can either be obtained by a call to parameterSets or by specifying X0 directly, in which case the prior variance of each parameter varprior also needs to be specified. Via plot (which uses functions of the package ggplot2 and reshape2), the indices can be visualized.

Usage

```
delsa(model = NULL, perturb=1.01,
  par.ranges, samples, method,
  X0, varprior,
  ...)

## S3 method for class 'delsa'
tell(x, y = NULL,...)

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

## S3 method for class 'delsa'
plot(x, which=1:3, ask = dev.interactive(), ...)
```

Arguments

model	a function, or a model with a predict method, defining the model to analyze.
perturb	Perturbation used to calculate sensitivity at each evaluation location
par.ranges	A named list of minimum and maximum parameter values
samples	Number of samples to generate. For the "grid" and "innergrid"method, corresponds to the number of samples for each parameter, and may be a vector.
method	Sampling scheme. See parameterSets
X0	Parameter values at which to evaluate sensitivity indices. Can be used instead of specifying sampling method

varprior	Prior variance. If X0 is specified, varprior must also be specified.
...	any other arguments for model which are passed unchanged each time it is called.
x	a list of class "delsa" storing the state of the sensitivity study (parameters, data, estimates).
y	a vector of model responses.
which	if a subset of the plots is required, specify a subset of the numbers 1:3
ask	logical; if TRUE, the user is asked before each plot, see par(ask=.)

Details

print shows summary of the first order indices across parameter space.

plot shows: (1) the cumulative distribution function of first order sensitivity across parameter space, (2) variation of first order sensitivity in relation to model response, and (3) sensitivity in relation to parameter value.

Value

delsa returns a list of class "delsa", containing all the input arguments detailed before, plus the following components:

call	the matched call.
X	a data.frame containing the design of experiments.
y	a vector of model responses.
delsafirst	the first order indices for each location in X0

Author(s)

Conversion for sensitivity package by Joseph Guillaume, based on original R code by Oldrich Rakovec

References

Rakovec, O., M. C. Hill, M. P. Clark, A. H. Weerts, A. J. Teuling, R. Uijlenhoet (2014), Distributed Evaluation of Local Sensitivity Analysis (DELSA), with application to hydrologic models, Water Resour. Res., 50, 1-18

See Also

parameterSets which is used to generate points, sensitivity for other methods in the package

Examples

```
# Test case : the non-monotonic Sobol g-function
# (there are 8 factors, all following the uniform distribution on [0,1])

## Not run:
library(randtoolbox)
x <- delsa(model=sobol.fun,
           par.ranges=replicate(8,c(0,1),simplify=FALSE),
           samples=100,method="sobol")

# Summary of sensitivity indices of each parameter across parameter space
print(x)

library(ggplot2)
library(reshape2)
x11()
plot(x)

## End(Not run)
```

`fast99` *Extended Fourier Amplitude Sensitivity Test*

Description

`fast99` implements the so-called "extended-FAST" method (Saltelli et al. 1999). This method allows the estimation of first order and total Sobol' indices for all the factors (alltogether $2p$ indices, where p is the number of factors) at a total cost of $n \times p$ simulations.

Usage

```
fast99(model = NULL, factors, n, M = 4, omega = NULL,
       q = NULL, q.arg = NULL, ...)
## S3 method for class 'fast99'
tell(x, y = NULL, ...)
## S3 method for class 'fast99'
print(x, ...)
## S3 method for class 'fast99'
plot(x, ylim = c(0, 1), ...)
```

Arguments

`model`	a function, or a model with a `predict` method, defining the model to analyze.
`factors`	an integer giving the number of factors, or a vector of character strings giving their names.
`n`	an integer giving the sample size, i.e. the length of the discretization of the s-space (see Cukier et al.).

M	an integer specifying the interference parameter, i.e. the number of harmonics to sum in the Fourier series decomposition (see Cukier et al.).
omega	a vector giving the set of frequencies, one frequency for each factor (see details below).
q	a vector of quantile functions names corresponding to wanted factors distributions (see details below).
q.arg	a list of quantile functions parameters (see details below).
x	a list of class "fast99" storing the state of the sensitivity study (parameters, data, estimates).
y	a vector of model responses.
ylim	y-coordinate plotting limits.
...	any other arguments for model which are passed unchanged each time it is called.

Details

If not given, the set of frequencies omega is taken from Saltelli et al. The first frequency of the vector omega is assigned to each factor X_i in turn (corresponding to the estimation of Sobol' indices S_i and S_{T_i}), other frequencies being assigned to the remaining factors.

If the arguments q and q.args are not given, the factors are taken uniformly distributed on $[0, 1]$. The argument q must be list of character strings, giving the names of the quantile functions (one for each factor), such as qunif, qnorm...It can also be a single character string, meaning same distribution for all. The argument q.arg must be a list of lists, each one being additional parameters for the corresponding quantile function. For example, the parameters of the quantile function qunif could be list(min=1, max=2), giving an uniform distribution on $[1, 2]$. If q is a single character string, then q.arg must be a single list (rather than a list of one list).

Value

fast99 returns a list of class "fast99", containing all the input arguments detailed before, plus the following components:

call	the matched call.
X	a data.frame containing the factors sample values.
y	a vector of model responses.
V	the estimation of variance.
D1	the estimations of Variances of the Conditional Expectations (VCE) with respect to each factor.
Dt	the estimations of VCE with respect to each factor complementary set of factors ("all but X_i").

Author(s)

Gilles Pujol

References

A. Saltelli, S. Tarantola and K. Chan, 1999, *A quantitative, model independent method for global sensitivity analysis of model output*, Technometrics, 41, 39–56.

R. I. Cukier, H. B. Levine and K. E. Schuler, 1978, *Nonlinear sensitivity analysis of multiparameter model systems*. J. Comput. Phys., 26, 1–42.

Examples

```
# Test case : the non-monotonic Ishigami function
x <- fast99(model = ishigami.fun, factors = 3, n = 1000,
            q = "qunif", q.arg = list(min = -pi, max = pi))
print(x)
plot(x)
```

---

morris *Morris's Elementary Effects Screening Method*

Description

`morris` implements the Morris's elementary effects screening method (Morris 1991). This method, based on design of experiments, allows to identify the few important factors at a cost of $r \times (p+1)$ simulations (where p is the number of factors). This implementation includes some improvements of the original method: space-filling optimization of the design (Campolongo et al. 2007) and simplex-based design (Pujol 2009).

Usage

```
morris(model = NULL, factors, r, design, binf = 0, bsup = 1,
       scale = TRUE, ...)
## S3 method for class 'morris'
tell(x, y = NULL, ...)
## S3 method for class 'morris'
print(x, ...)
## S3 method for class 'morris'
plot(x, identify = FALSE, atpen = FALSE, y_col = NULL,
  y_dim3 = NULL, ...)
## S3 method for class 'morris'
plot3d(x, alpha = c(0.2, 0), sphere.size = 1, y_col = NULL,
  y_dim3 = NULL)
```

Arguments

`model`	a function, or a model with a `predict` method, defining the model to analyze.
`factors`	an integer giving the number of factors, or a vector of character strings giving their names.

`r` — either an integer giving the number of repetitions of the design, i.e. the number of elementary effect computed per factor, or a vector of two integers `c(r1, r2)` for the space-filling improvement (Campolongo et al. 2007). In this case, `r1` is the wanted design size, and `r2` ($>$ `r1`) is the size of the (bigger) population in which is extracted the design (this can throw a warning, see below).

`design` — a list specifying the design type and its parameters:

- `type = "oat"` for Morris's OAT design (Morris 1991), with the parameters:
  - `levels` : either an integer specifying the number of levels of the design, or a vector of integers for different values for each factor.
  - `grid.jump` : either an integer specifying the number of levels that are increased/decreased for computing the elementary effects, or a vector of integers for different values for each factor. If not given, it is set to `grid.jump = 1`. Notice that this default value of one does not follow Morris's recommendation of `levels`$/2$.
- `type = "simplex"` for simplex-based design (Pujol 2009), with the parameter:
  - `scale.factor` : a numeric value, the homothety factor of the (isometric) simplexes. Edges equal one with a scale factor of one.

`binf` — either an integer, specifying the minimum value for the factors, or a vector for different values for each factor.

`bsup` — either an integer, specifying the maximum value for the factors, or a vector for different values for each factor.

`scale` — logical. If `TRUE`, the input design of experiments is scaled after building the design and before computing the elementary effects so that all factors vary within the range [0,1]. For each factor, the scaling is done relatively to its corresponding bsup and binf.

`x` — a list of class `"morris"` storing the state of the screening study (parameters, data, estimates).

`y` — a vector of model responses.

`identify` — logical. If `TRUE`, the user selects with the mouse the factors to label on the (μ^*, σ) graph (see `identify`).

`atpen` — logical. If `TRUE` (and `identify = TRUE`), the user-identified labels (more precisely: their lower-left corners) of the factors are plotted at the place where the user had clicked (if near enough to one of the factor points). If `FALSE` (and `identify = TRUE`), the labels are automatically adjusted to the lower, left, upper or right side of the factor point. For further information, see `identify`. Defaults to `FALSE`.

`y_col` — an integer defining the index of the column of `x$y` to be used for plotting the corresponding Morris statistics μ^* and σ (only applies if `x$y` is a matrix or an array). If set to `NULL` (as per default) and `x$y` is a matrix or an array, the first column (respectively the first element in the second dimension) of `x$y` is used (i.e. `y_col = 1`).

`y_dim3` — an integer defining the index in the third dimension of `x$y` to be used for plotting the corresponding Morris statistics μ^* and σ (only applies if `x$y` is an array). If set to `NULL` (as per default) and `x$y` is a three-dimensional array, the first element in the third dimension of `x$y` is used (i.e. `y_dim3 = 1`).

alpha	a vector of three values between 0.0 (fully transparent) and 1.0 (opaque) (see rgl.material). The first value is for the cone, the second for the planes.
sphere.size	a numeric value, the scale factor for displaying the spheres.
...	for morris: any other arguments for model which are passed unchanged each time it is called. For plot.morris: arguments to be passed to plot.default.

Details

plot.morris draws the (μ^*, σ) graph.

plot3d.morris draws the (μ, μ^*, σ) graph (requires the **rgl** package). On this graph, the points are in a domain bounded by a cone and two planes (application of the Cauchy-Schwarz inequality).

When using the space-filling improvement (Campolongo et al. 2007) of the Morris design, we recommend to install before the "pracma" R package: its "distmat"" function makes running the function with a large number of initial estimates (r2) significantly faster (by accelerating the inter-point distances calculations).

This version of morris also supports matrices and three-dimensional arrays as output of model.

Value

morris returns a list of class "morris", containing all the input argument detailed before, plus the following components:

call	the matched call.
X	a data.frame containing the design of experiments.
y	either a vector, a matrix or a three-dimensional array of model responses (depends on the output of model).
ee	• if y is a vector: a $(r \times p)$ - matrix of elementary effects for all the factors. • if y is a matrix: a $(r \times p \times ncol(y))$ - array of elementary effects for all the factors and all columns of y. • if y is a three-dimensional array: a $(r \times p \times dim(y)[2] \times dim(y)[3])$ - array of elementary effects for all the factors and all elements of the second and third dimension of y.

Notice that the statistics of interest (μ, μ^* and σ) are not stored. They can be printed by the print method, but to extract numerical values, one has to compute them with the following instructions:

If x$y is a vector:

```
mu <- apply(x$ee, 2, mean)
mu.star <- apply(x$ee, 2, function(x) mean(abs(x)))
sigma <- apply(x$ee, 2, sd)
```

If x$y is a matrix:

```
mu <- apply(x$ee, 3, function(M){
  apply(M, 2, mean)
})
```

```
mu.star <- apply(abs(x$ee), 3, function(M){
  apply(M, 2, mean)
})
sigma <- apply(x$ee, 3, function(M){
  apply(M, 2, sd)
})
```

If `x$y` is a three-dimensional array:

```
mu <- sapply(1:dim(x$ee)[4], function(i){
  apply(x$ee[, , , i, drop = FALSE], 3, function(M){
    apply(M, 2, mean)
  })
}, simplify = "array")
mu.star <- sapply(1:dim(x$ee)[4], function(i){
  apply(abs(x$ee)[, , , i, drop = FALSE], 3, function(M){
    apply(M, 2, mean)
  })
}, simplify = "array")
sigma <- sapply(1:dim(x$ee)[4], function(i){
  apply(x$ee[, , , i, drop = FALSE], 3, function(M){
    apply(M, 2, sd)
  })
}, simplify = "array")
```

It is highly recommended to use the function with the argument `scale = TRUE` to avoid an uncorrect interpretation of factors that would have different orders of magnitude.

Warning messages

"keeping r' repetitions out of r" when generating the design of experiments, identical repetitions are removed, leading to a lower number than requested.

Author(s)

Gilles Pujol, with contributions from Frank Weber (2016)

References

M. D. Morris, 1991, *Factorial sampling plans for preliminary computational experiments*, Technometrics, 33, 161–174.

F. Campolongo, J. Cariboni and A. Saltelli, 2007, *An effective screening design for sensitivity*, Environmental Modelling \& Software, 22, 1509–1518.

G. Pujol, 2009, *Simplex-based screening designs for estimating metamodels*, Reliability Engineering and System Safety 94, 1156–1160.

Examples

```
# Test case : the non-monotonic function of Morris
x <- morris(model = morris.fun, factors = 20, r = 4,
            design = list(type = "oat", levels = 5, grid.jump = 3))
print(x)
plot(x)
## Not run:
library(rgl)
plot3d.morris(x)  # (requires the package 'rgl')
## End(Not run)

# Only for demonstration purposes: a model function returning a matrix
morris.fun_matrix <- function(X){
  res_vector <- morris.fun(X)
  cbind(res_vector, 2 * res_vector)
}
x <- morris(model = morris.fun_matrix, factors = 20, r = 4,
            design = list(type = "oat", levels = 5, grid.jump = 3))
plot(x, y_col = 2)
title(main = "y_col = 2")

# Also only for demonstration purposes: a model function returning a
# three-dimensional array
morris.fun_array <- function(X){
  res_vector <- morris.fun(X)
  res_matrix <- cbind(res_vector, 2 * res_vector)
  array(data = c(res_matrix, 5 * res_matrix),
        dim = c(length(res_vector), 2, 2))
}
x <- morris(model = morris.fun_array, factors = 20, r = 4,
            design = list(type = "simplex", scale.factor = 1))
plot(x, y_col = 2, y_dim3 = 2)
title(main = "y_col = 2, y_dim3 = 2")
```

`parameterSets` *Generate parameter sets*

Description

Generate parameter sets from given ranges, with chosen sampling scheme

Usage

```
parameterSets(par.ranges, samples, method = c("sobol", "innergrid", "grid"))
```

Arguments

`par.ranges`	A named list of minimum and maximum parameter values

samples	Number of samples to generate. For the "grid" and "innergrid" method, may be a vector of number of samples for each parameter.
method	the sampling scheme; see Details

Details

Method "sobol" generates uniformly distributed Sobol low discrepancy numbers, using the sobol function in the randtoolbox package.

Method "grid" generates a grid within the parameter ranges, including its extremes, with number of points determined by samples

Method "innergrid" generates a grid within the parameter ranges, with edges of the grid offset from the extremes. The offset is calculated as half of the resolution of the grid diff(par.ranges)/samples/2.

Value

the result is a matrix, with named columns for each parameter in par.ranges. Each row represents one parameter set.

Author(s)

Joseph Guillaume, based on similar function by Felix Andrews

See Also

delsa, which uses this function

Examples

```
X.grid <- parameterSets(par.ranges=list(V1=c(1,1000),V2=c(1,4)),
                        samples=c(10,10),method="grid")
plot(X.grid)

X.innergrid<-parameterSets(par.ranges=list(V1=c(1,1000),V2=c(1,4)),
                        samples=c(10,10),method="innergrid")
points(X.innergrid,col="red")

## Not run:
library(randtoolbox)
X.sobol<-parameterSets(par.ranges=list(V1=c(1,1000),V2=c(1,4)),
                         samples=100,method="sobol")
plot(X.sobol)

## End(Not run)
```

pcc *Partial Correlation Coefficients*

Description

pcc computes the Partial Correlation Coefficients (PCC), or Partial Rank Correlation Coefficients (PRCC), which are sensitivity indices based on linear (resp. monotonic) assumptions, in the case of (linearly) correlated factors.

Usage

```
pcc(X, y, rank = FALSE, nboot = 0, conf = 0.95)
## S3 method for class 'pcc'
print(x, ...)
## S3 method for class 'pcc'
plot(x, ylim = c(-1,1), ...)
```

Arguments

X	a data frame (or object coercible by as.data.frame) containing the design of experiments (model input variables).
y	a vector containing the responses corresponding to the design of experiments (model output variables).
rank	logical. If TRUE, the analysis is done on the ranks.
nboot	the number of bootstrap replicates.
conf	the confidence level of the bootstrap confidence intervals.
x	the object returned by pcc.
ylim	the y-coordinate limits of the plot.
...	arguments to be passed to methods, such as graphical parameters (see par).

Value

pcc returns a list of class "pcc", containing the following components:

call	the matched call.
PCC	a data frame containing the estimations of the PCC indices, bias and confidence intervals (if rank = TRUE).
PRCC	a data frame containing the estimations of the PRCC indices, bias and confidence intervals (if rank = TRUE).

Author(s)

Gilles Pujol

References

A. Saltelli, K. Chan and E. M. Scott eds, 2000, *Sensitivity Analysis*, Wiley.

See Also

src

Examples

```
# a 100-sample with X1 ~ U(0.5, 1.5)
#                   X2 ~ U(1.5, 4.5)
#                   X3 ~ U(4.5, 13.5)
library(boot)
n <- 100
X <- data.frame(X1 = runif(n, 0.5, 1.5),
                X2 = runif(n, 1.5, 4.5),
                X3 = runif(n, 4.5, 13.5))

# linear model : Y = X1 + X2 + X3
y <- with(X, X1 + X2 + X3)

# sensitivity analysis
x <- pcc(X, y, nboot = 100)
print(x)
#plot(x) # TODO: find another example...
```

PLI *Perturbed-Law based sensitivity Indices (PLI) for failure probability*

Description

PLI computes the Perturbed-Law based Indices (PLI), also known as the Density Modification Based Reliability Sensitivity Indices (DMBRSI), which are sensitivity indices related to a probability of exceedence of a model output (i.e. a failure probability), estimated by a Monte Carlo method. See Lemaitre et al. (2015).

Usage

```
PLI(failurepoints,failureprobabilityhat,samplesize,deltasvector,
       InputDistributions,type="MOY",samedelta=TRUE)
```

Arguments

failurepoints	a matrix of failure points coordinates, one column per variable.
failureprobabilityhat	the estimation of failure probability P through rough Monte Carlo method.
samplesize	the size of the sample used to estimate P. One must have Pchap=dim(failurepoints)[1]/samplesize
deltasvector	a vector containing the values of delta for which the indices will be computed.

`InputDistributions`
: a list of list. Each list contains, as a list, the name of the distribution to be used and the parameters. Implemented cases so far:

  - For a mean perturbation: Gaussian, Uniform, Triangle, Left Trucated Gaussian, Left Truncated Gumbel. Using Gumber requires the package `evd`.
  - For a variance perturbation: Gaussian, Uniform.

`type`
: a character string in which the user will specify the type of perturbation wanted. The sense of "deltasvector" varies according to the type of perturbation:

  - type can take the value "MOY",in which case deltasvector is a vector of perturbated means.
  - type can take the value "VAR",in which case deltasvector is a vector of perturbated variances, therefore needs to be positive integers.

`samedelta`
: a boolean used with the value "MOY" for type.

  - If it is set at TRUE, the mean perturbation will be the same for all the variables.
  - If not, the mean perturbation will be new_mean = mean+sigma*delta where mean, sigma are parameters defined in InputDistributions and delta is a value of deltasvector.

Value

`PLI` returns a list of size 2, including:

- A matrix where the PLI are stored. Each column corresponds to an input, each line corresponds to a twist of amplitude delta.
- A matrix where their standard deviation are stored.

Author(s)

Paul Lemaitre

References

P. Lemaitre, E. Sergienko, A. Arnaud, N. Bousquet, F. Gamboa and B. Iooss, *Density modification based reliability sensitivity analysis*, Journal of Statistical Computation and Simulation, 85:1200-1223.

E. Borgonovo and B. Iooss, 2017, *Moment independent importance measures and a common rationale*, In: *Springer Handbook on UQ*, R. Ghanem, D. Higdon and H. Owhadi (Eds).

See Also

`PLIquantile`

Examples

```
## Not run:

# Model: Ishigami function with a treshold at -7
# Failure points are those < -7

  distributionIshigami = list()
for (i in 1:3){
distributionIshigami[[i]]=list("unif",c(-pi,pi))
distributionIshigami[[i]]$r=("runif")
}

# Monte Carlo sampling to obtain failure points

  N = 10^5
X = matrix(0,ncol=3,nrow=N)
for( i in 1:3){
    X[,i] = runif(N,-pi,pi)
    }

T = ishigami.fun(X)
s = sum(as.numeric(T < -7)) # Number of failure
pdefchap = s/N      # Failure probability
ptsdef = X[T < -7,] # Failure points

# sensitivity indices with perturbation of the mean

v_delta = seq(-3,3,1/20)
Toto = PLI(failurepoints=ptsdef,failureprobabilityhat=pdefchap,samplesize=N,
deltasvector=v_delta,InputDistributions=distributionIshigami,type="MOY",
samedelta=TRUE)
BIshm = Toto[[1]]
SIshm = Toto[[2]]

par(mar=c(4,5,1,1))
plot(v_delta,BIshm[,2],ylim=c(-4,4),xlab=expression(delta),
ylab=expression(hat(S[i*delta])),pch=19,cex=1.5)
points(v_delta,BIshm[,1],col="darkgreen",pch=15,cex=1.5)
points(v_delta,BIshm[,3],col="red",pch=17,cex=1.5)
lines(v_delta,BIshm[,2]+1.96*SIshm[,2],col="black");
lines(v_delta,BIshm[,2]-1.96*SIshm[,2],col="black")
lines(v_delta,BIshm[,1]+1.96*SIshm[,1],col="darkgreen");
lines(v_delta,BIshm[,1]-1.96*SIshm[,1],col="darkgreen")
lines(v_delta,BIshm[,3]+1.96*SIshm[,3],col="red");
lines(v_delta,BIshm[,3]-1.96*SIshm[,3],col="red");
abline(h=0,lty=2)
legend(0,3,legend=c("X1","X2","X3"),
col=c("darkgreen","black","red"),pch=c(15,19,17),cex=1.5)

# sensitivity indices with perturbation of the variance

v_delta = seq(1,5,1/4) # user parameter. (the true variance is 3.29)
```

```
Toto = PLI(failurepoints=ptsdef,failureprobabilityhat=pdefchap,samplesize=N,
deltasvector=v_delta,InputDistributions=distributionIshigami,type="VAR",
samedelta=TRUE)
BIshv=Toto[[1]]
SIshv=Toto[[2]]

par(mfrow=c(2,1),mar=c(1,5,1,1)+0.1)
plot(v_delta,BIshv[,2],ylim=c(-.5,.5),xlab=expression(V_f),
ylab=expression(hat(S[i*delta])),pch=19,cex=1.5)
points(v_delta,BIshv[,1],col="darkgreen",pch=15,cex=1.5)
points(v_delta,BIshv[,3],col="red",pch=17,cex=1.5)
lines(v_delta,BIshv[,2]+1.96*SIshv[,2],col="black");
lines(v_delta,BIshv[,2]-1.96*SIshv[,2],col="black")
lines(v_delta,BIshv[,1]+1.96*SIshv[,1],col="darkgreen");
lines(v_delta,BIshv[,1]-1.96*SIshv[,1],col="darkgreen")
lines(v_delta,BIshv[,3]+1.96*SIshv[,3],col="red");
lines(v_delta,BIshv[,3]-1.96*SIshv[,3],col="red");

par(mar=c(4,5.1,1.1,1.1))
plot(v_delta,BIshv[,2],ylim=c(-30,.7),xlab=expression(V[f]),
ylab=expression(hat(S[i*delta])),pch=19,cex=1.5)
points(v_delta,BIshv[,1],col="darkgreen",pch=15,cex=1.5)
points(v_delta,BIshv[,3],col="red",pch=17,cex=1.5)
lines(v_delta,BIshv[,2]+1.96*SIshv[,2],col="black");
lines(v_delta,BIshv[,2]-1.96*SIshv[,2],col="black")
lines(v_delta,BIshv[,1]+1.96*SIshv[,1],col="darkgreen");
lines(v_delta,BIshv[,1]-1.96*SIshv[,1],col="darkgreen")
lines(v_delta,BIshv[,3]+1.96*SIshv[,3],col="red");
lines(v_delta,BIshv[,3]-1.96*SIshv[,3],col="red");
legend(2.5,-10,legend=c("X1","X2","X3"),col=c("darkgreen","black","red"),
pch=c(15,19,17),cex=1.5)


## End(Not run)
```

`PLIquantile` *Perturbed-Law based sensitivity Indices (PLI) for quantile*

Description

PLIquantile computes the Perturbed-Law based Indices (PLI) for quantile, which are sensitivity indices related to a quantile of a model output, estimated by a Monte Carlo method, See Sueur et al. (2016, 2017).

Usage

```
PLIquantile(order,x,y,quantilehat,deltasvector,
       InputDistributions,type="MOY",samedelta=TRUE)
```

Arguments

order	the order of the quantile to estimate.
x	the matrix of simulation points coordinates, one column per variable.
y	the vector of model outputs.
quantilehat	the estimation of quantile q.
deltasvector	a vector containing the values of delta for which the indices will be computed.
InputDistributions	a list of list. Each list contains, as a list, the name of the distribution to be used and the parameters. Implemented cases so far: • For a mean perturbation: Gaussian, Uniform, Triangle, Left Trucated Gaussian, Left Truncated Gumbel. Using Gumber requires the package evd. • For a variance perturbation: Gaussian, Uniform.
type	a character string in which the user will specify the type of perturbation wanted. The sense of "deltasvector" varies according to the type of perturbation: • type can take the value "MOY",in which case deltasvector is a vector of perturbated means. • type can take the value "VAR",in which case deltasvector is a vector of perturbated variances, therefore needs to be positive integers.
samedelta	a boolean used with the value "MOY" for type. • If it is set at TRUE, the mean perturbation will be the same for all the variables. • If not, the mean perturbation will be new_mean = mean+sigma*delta where mean, sigma are parameters defined in InputDistributions and delta is a value of deltasvector.

Value

PLIquantile returns a matrix where the PLI are stored. Each column corresponds to an input, each line corresponds to a twist of amplitude delta.

Author(s)

Paul Lemaitre, Bertrand Iooss and Thibault Delage

References

P. Lemaitre, E. Sergienko, A. Arnaud, N. Bousquet, F. Gamboa and B. Iooss, 2015, *Density modification based reliability sensitivity analysis*, Journal of Statistical Computation and Simulation, 85:1200-1223.

R. Sueur, N. Bousquet, B. Iooss and J. Bect, 2016, *Perturbed-Law based sensitivity Indices for sensitivity analysis in structural reliability*, Proceedings of the SAMO 2016 Conference, Reunion Island, France, December 2016.

R. Sueur, B. Iooss and T. Delage, 2017, *Sensitivity analysis using perturbed-law based indices for quantiles and application to an industrial case*, 10th International Conference on Mathematical Methods in Reliability (MMR 2017), Grenoble, France, July 2017.

See Also

PLI

Examples

```
## Not run:

# Model: 3D function

  distribution = list()
for (i in 1:3) distribution[[i]]=list("norm",c(0,1))

# Monte Carlo sampling to obtain failure points

  N = 10000
X = matrix(0,ncol=3,nrow=N)
for(i in 1:3) X[,i] = rnorm(N,0,1)

Y = 2 * X[,1] + X[,2] + X[,3]/2
q95 = quantile(Y,0.95)

# sensitivity indices with perturbation of the mean

v_delta = seq(-1,1,1/10)
toto = PLIquantile(0.95,X,Y,q95,deltasvector=v_delta,
  InputDistributions=distribution,type="MOY",samedelta=TRUE)

par(mar=c(4,5,1,1))
plot(v_delta,toto[,2],ylim=c(-4.5,4.5),xlab=expression(delta),
ylab=expression(hat(S[i*delta])),pch=19,cex=1.5)
points(v_delta,toto[,1],col="darkgreen",pch=15,cex=1.5)
points(v_delta,toto[,3],col="red",pch=17,cex=1.5)
abline(h=0,lty=2)
legend(0.8,4.4,legend=c("X1","X2","X3"),
col=c("darkgreen","black","red"),pch=c(15,19,17),cex=1.5)


## End(Not run)
```

`plot.support` *Support index functions: Measuring the effect of input variables over their support*

Description

Methods to plot the normalized support index functions (Fruth et al., 2016).

Usage

```
## S3 method for class 'support'
plot(x, i = 1:ncol(x$X),
        xprob = FALSE, p = NULL, p.arg = NULL,
        ylim = NULL, col = 1:3, lty = 1:3, lwd = c(2,2,1), cex = 1, ...)
## S3 method for class 'support'
scatterplot(x, i = 1:ncol(x$X),
               xprob = FALSE, p = NULL, p.arg = NULL,
               cex = 1, cex.lab = 1, ...)
```

Arguments

x	an object of class support.
i	an optional vector of integers indicating the subset of input variables X_i for plotting. Default is the entire set of input variables.
xprob	an optional boolean indicating whether the inputs should be plotted in probability scale.
p	,
p.arg	list of probability names and parameters for the input distribution.
ylim	,
col	,
lty	,
lwd	,
cex	,
cex.lab	usual graphical parameters.
...	additional graphical parameters to be passed to scatterplot method (ggMarginal function).

Details

If xprob = TRUE, the input variable X_i is plotted in probability scale according to the informations provided in the arguments p, p.arg: The x-axis is thus F(x), where F is the cdf of X_i. If these ones are not provided, the empirical distribution is used for rescaling: The x-axis is thus Fn(x), where Fn is the empirical cdf of X_i.

Legend details:

zeta*T : normalized total support index function

zeta* : normalized 1st-order support index function

nu* : normalized DGSM

Notice that the sum of (normalized) DGSM (nu*) over all input variables is equal to 1. Furthermore, the expectation of the total support index function (zeta*T) is equal to the (normalized) DGSM (nu*).

Author(s)

O. Roustant

See Also

Estimation of support index functions: `support`

`PoincareConstant` *Poincare constants for Derivative-based Global Sensitivity Measures (DGSM)*

Description

A DGSM is a sensitivity index relying on the integral (over the space domain of the input variables) of the squared derivatives of a model output with respect to one model input variable. The product between a DGSM and a Poincare Constant (Roustant et al., 2014: Roustant et al., 2017) gives an upper bound of the total Sobol' index corresponding to the same input (Lamboni et al., 2013; Kucherenko and Iooss, 2016).

This Poincare constant depends on the type of probability distribution of the input variable. In the particular case of log-concave distribution, analytical formulas are available for double-exponential transport by the way of the median value (Lamboni et al., 2013). For truncated log-concave distributions, different formulas are available (Roustant et al., 2014). For general distributions (truncated or not), some Poincare constants can be computed via a relatively simple optimization process using different formula coming from transport inequalities (Roustant et al., 2017).

Notice that the analytical formula based on the log-concave law cases is a subcase of the double-exponential transport. In all cases, with this function, the smallest constant is obtained using the logistic transport formula. `PoincareOptimal` allows to obtained the best (optimal) constant using another (spectral) method.

IMPORTANT: This program is useless for the two following input variable distributions:

- uniform on $[min, max]$ interval: The optimal Poincare constant is $\frac{(max-min)^2}{pi^2}$.
- normal with a standard deviation sd: The optimal Poincare constant is sd^2.

Usage

```
PoincareConstant(dfct=dnorm, qfct=qnorm, pfct=pnorm,
             logconcave=FALSE, transport="logistic", optimize.interval=c(-100, 100),
                truncated=FALSE, min=0, max=1,  ...)
```

Arguments

`dfct`	the probability density function of the input variable
`qfct`	the quantile function of the input variable
`pfct`	the distribution function of the input variable

logconcave	logical value: TRUE for a log-concave distribution (analyical formula will be used). Requires argument 'dfct' and 'qfct'. FALSE (default value) means that the calculations will be performed using transport-based formulas (applicable for log-concave and non-log concave cases)
transport	If logconcave=FALSE, choice of the transport inequalities to be used: "double_exp" (default value) for double exponential transport and "logistic" for logistic transport". Requires argument 'dfct' and 'pfct'
optimize.interval	In the transport-based case (logconcave=FALSE), a vector containing the endpoints of the interval to be searched for the maximum of the function to be optimized
truncated	logical value: TRUE for a truncated distribution. Default value is FALSE
min	the minimal bound in the case of a truncated distribution
max	the maximal bound in the case of a truncated distribution
...	additional arguments

Details

In the case of truncated distributions (truncated=TRUE), in addition to the min and max arguments: - the truncated distribution name has to be passed in the 'dfct' and 'pfct' arguments if logconcave=FALSE, - the non-truncated distribution name has to be passed in the 'dfct' and 'qfct' arguments if logconcave=TRUE. Moreover, if min and max are finite, optimize.interval is required to be defined as c(min,max).

Value

`PoincareConstant` returns the value of the Poincare constant.

Author(s)

Jana Fruth, Bertrand Iooss and Olivier Roustant

References

S. Kucherenko and B. Iooss, Derivative-based global sensitivity measures, In: R. Ghanem, D. Higdon and H. Owhadi (eds.), Handbook of Uncertainty Quantification, 2016.

M. Lamboni, B. Iooss, A-L. Popelin and F. Gamboa, Derivative-based global sensitivity measures: General links with Sobol' indices and numerical tests, Mathematics and Computers in Simulation, 87:45-54, 2013.

O. Roustant, F. Barthe and B. Iooss, Poincare inequalities on intervals - application to sensitivity analysis, Electronic Journal of Statistics, Vol. 11, No. 2, 3081-3119, 2017, https://hal.archives-ouvertes.fr/hal-01388758.

O. Roustant, J. Fruth, B. Iooss and S. Kuhnt, Crossed-derivative-based sensitivity measures for interaction screening, Mathematics and Computers in Simulation, 105:105-118, 2014.

See Also

PoincareOptimal

Examples

```
# Exponential law (log-concave)
PoincareConstant(dfct=dexp,qfct=qexp,pfct=NULL,rate=1,logconcave=TRUE) # log-concave assumption
PoincareConstant(dfct=dexp,qfct=NULL,pfct=pexp,rate=1,optimize.interval=c(0, 15))
          # logistic transport approach

# Weibull law (log-concave)
PoincareConstant(dfct=dweibull,qfct=NULL,pfct=pweibull,optimize.interval=c(0, 15),shape=1,scale=1)
          # logistic transport approach

## Not run:
# Triangular law (log-concave)
library(triangle)
PoincareConstant(dfct=dtriangle, qfct=qtriangle, pfct=NULL, a=-1, b=1, c=0, logconcave=TRUE)
          # log-concave assumption
PoincareConstant(dfct=dtriangle, qfct=NULL, pfct=ptriangle, a=-1, b=1, c=0,
  transport="double_exp", optimize.interval=c(-1,1)) # Double-exponential transport approach
PoincareConstant(dfct=dtriangle, qfct=NULL, pfct=ptriangle, a=-1, b=1, c=0,
  optimize.interval=c(-1,1)) # Logistic transport calculation

# Normal N(0,1) law truncated on [-1.87,+infty]
PoincareConstant(dfct=dnorm, qfct=qnorm, pfct=pnorm, mean=0, sd=1, logconcave=TRUE,
  transport="double_exp", truncated=TRUE, min=-1.87, max=999) # log-concave assumption
PoincareConstant(dfct=dtnorm, qfct=qtnorm, pfct=ptnorm, mean=0, sd=1, truncated=TRUE,
  min=-1.87, max=999,   transport="double_exp", optimize.interval=c(-1.87,20))
           # Double-exponential transport approach
PoincareConstant(dfct=dtnorm, qfct=qtnorm, pfct=ptnorm, mean=0, sd=1, truncated=TRUE,
  min=-1.87, max=999, optimize.interval=c(-1.87,20)) # Logistic transport approach


# Gumbel law (log-concave)
library(evd)
PoincareConstant(dfct=dgumbel, qfct=qgumbel, pfct=NULL, loc=0, scale=1, logconcave=TRUE,
  transport="double_exp") # log-concave assumption
PoincareConstant(dfct=dgumbel, qfct=NULL, pfct=pgumbel, loc=0, scale=1,
  transport="double_exp", optimize.interval=c(-3,20)) # Double-exponential transport approach
PoincareConstant(dfct=dgumbel, qfct=qgumbel, pfct=pgumbel, loc=0, scale=1,
  optimize.interval=c(-3,20)) # Logistic transport approach

# Truncated Gumbel law (log-concave)
PoincareConstant(dfct=dgumbel, qfct=qgumbel, pfct=pgumbel, loc=0, scale=1, logconcave=TRUE,
  transport="double_exp", truncated=TRUE, min=-0.92, max=3.56) # log-concave assumption
PoincareConstant(dfct=dtgumbel, qfct=NULL, pfct=ptgumbel, loc=0, scale=1, truncated=TRUE,
  min=-0.92, max=3.56, transport="double_exp", optimize.interval=c(-0.92,3.56))
           # Double-exponential transport approach
PoincareConstant(dfct=dtgumbel, qfct=qtgumbel, pfct=ptgumbel, loc=0, scale=1, truncated=TRUE,
  min=-0.92, max=3.56, optimize.interval=c(-0.92,3.56)) # Logistic transport approach
```

```
## End(Not run)
```

`PoincareOptimal` *Optimal Poincare constants for Derivative-based Global Sensitivity Measures (DGSM)*

Description

A DGSM is a sensitivity index relying on the integral (over the space domain of the input variables) of the squared derivatives of a model output with respect to one model input variable. The product between a DGSM and a Poincare Constant (Roustant et al., 2014: Roustant et al., 2017), on the type of probability distribution of the input variable, gives an upper bound of the total Sobol' index corresponding to the same input (Lamboni et al., 2013; Kucherenko and Iooss, 2016).

This function provides the optimal Poincare constant as explained in Roustant et al. (2017). It solves numerically the spectral problem corresponding to the Poincare inequality, with Neumann conditions. The differential equation is f" - V'f'= - lambda f with f'(a) = f'(b) = 0. In addition, all the spectral decomposition can be returned by the function. The information corresponding to the optimal constant is given in the second to last column.

IMPORTANT: This program is useless for the two following input variable distributions:

- uniform on $[min, max]$ interval: The optimal Poincare constant is $\frac{(max-min)^2}{pi^2}$.
- normal with a standard deviation sd: The optimal Poincare constant is sd^2.

Usage

```
PoincareOptimal(distr=list("unif",c(0,1)), min=NULL, max=NULL, n = 500,
    method = c("quadrature", "integral"), only.values = TRUE, plot = FALSE, ...)
```

Arguments

`distr`	a list or a function corresponding to the probability distribution. • If it is a list, it contains the name of the R distribution of the variable and its parameters. Possible choices are: "unif" (uniform), "norm" (normal), "exp" (exponential), "triangle" (triangular from package triangle), "gumbel" (from package evd), "beta", "gamma", "weibull" and "lognorm" (log-normal). The values of the distribution parameters have to be passed in arguments in the same order than the corresponding R function. • If it is a function, it corresponds to the pdf. Notice that the normalizing constant has no impact on the computation of the optimal Poincare constant and can be ommitted.
`min`	see below

max	[min,max]: interval on which the distribution is truncated. Choose low and high quantiles in case of unbounded distribution. Choose NULL for uniform and triangular distributions
n	number of discretization steps
method	method of integration: "quadrature" (default value) uses the trapez quadrature (close and quicker), "integral" is longer but does not make any approximation
only.values	if TRUE, only eigen values are computed and returned, otherwise both eigenvalues and eigenvectors are returned (default value is TRUE)
plot	logical:if TRUE and only.values=FALSE, plots a minimizer of the Rayleigh ratio (default value is FALSE)
...	additional arguments

Details

For the uniform, normal, triangular and Gumbel distributions, the optimal constants are computed on the standardized correponding distributions (for a better numerical efficiency). In these cases, the return optimal constant and eigen values correspond to original distributions, while the eigen vectors are not rescaled.

Value

PoincareOptimal returns a list containing:

opt	the optimal Poincare constant
values	the eigen values
vectors	the eigen vectors

Author(s)

Olivier Roustant and Bertrand Iooss

References

O. Roustant, F. Barthe and B. Iooss, Poincare inequalities on intervals - application to sensitivity analysis, Electronic Journal of Statistics, Vol. 11, No. 2, 3081-3119, 2017, https://hal.archives-ouvertes.fr/hal-01388758.

See Also

PoincareConstant

Examples

```
# uniform [0,1]
out <- PoincareOptimal(distr=list("unif",0,1))
print(out$opt)
```

```
# truncated standard normal on [-1, 1]
out <- PoincareOptimal(distr=dnorm, min=-1, max=1, plot=TRUE, only.values=FALSE)
print(out$opt)

## Not run:
# truncated standard normal on [-1.87, +infty]
out <- PoincareOptimal(distr=list("norm",0,1), min=-1.87, max=5, method="integral", n=500)
print(out$opt)

# truncated Gumbel(0,1) on [-0.92, 3.56]
out <- PoincareOptimal(distr=list("gumbel",0,1), min=-0.92, max=3.56, method="integral", n=500)
print(out$opt)

# symetric triangular [-1,1]
out <- PoincareOptimal(distr=list("triangle",-1,1,0), min=NULL, max=NULL)
print(out$opt)

# Lognormal distribution
out <- PoincareOptimal(distr=list("lognorm",1,2), min=3, max=10, only.values=FALSE,plot=TRUE,
  method="integral")
print(out$opt)


## End(Not run)
```

---

sb *Sequential Bifurcations*

Description

sb implements the Sequential Bifurcations screening method (Bettonvil and Kleijnen 1996).

Usage

```
sb(p, sign = rep("+", p), interaction = FALSE)
## S3 method for class 'sb'
ask(x, i = NULL, ...)
## S3 method for class 'sb'
tell(x, y, ...)
## S3 method for class 'sb'
print(x, ...)
## S3 method for class 'sb'
plot(x, ...)
```

Arguments

p	number of factors.

sign	a vector fo length p filled with "+" and "-", giving the (assumed) signs of the factors effects.
interaction	a boolean, TRUE if the model is supposed to be with interactions, FALSE otherwise.
x	a list of class "sb" storing the state of the screening study at the current iteration.
y	a vector of model responses.
i	an integer, used to force a wanted bifurcation instead of that proposed by the algorithm.
...	not used.

Details

The model without interaction is

$$Y = \beta_0 + \sum_{i=1}^{p} \beta_i X_i$$

while the model with interactions is

$$Y = \beta_0 + \sum_{i=1}^{p} \beta_i X_i + \sum_{1 \leq i < j \leq p} \gamma_{ij} X_i X_j$$

In both cases, the factors are assumed to be uniformly distributed on $[-1, 1]$. This is a difference with Bettonvil et al. where the factors vary across $[0, 1]$ in the former case, while $[-1, 1]$ in the latter.

Another difference with Bettonvil et al. is that in the current implementation, the groups are splitted right in the middle.

Value

sb returns a list of class "sb", containing all the input arguments detailed before, plus the following components:

i	the vector of bifurcations.
y	the vector of observations.
ym	the vector of mirror observations (model with interactions only).

The groups effects can be displayed with the print method.

Author(s)

Gilles Pujol

References

B. Bettonvil and J. P. C. Kleijnen, 1996, *Searching for important factors in simulation models with many factors: sequential bifurcations*, European Journal of Operational Research, 96, 180–194.

Examples

```
# a model with interactions
p <- 50
beta <- numeric(length = p)
beta[1:5] <- runif(n = 5, min = 10, max = 50)
beta[6:p] <- runif(n = p - 5, min = 0, max = 0.3)
beta <- sample(beta)
gamma <- matrix(data = runif(n = p^2, min = 0, max = 0.1), nrow = p, ncol = p)
gamma[lower.tri(gamma, diag = TRUE)] <- 0
gamma[1,2] <- 5
gamma[5,9] <- 12
f <- function(x) { return(sum(x * beta) + (x %*% gamma %*% x))}

# 10 iterations of SB
sa <- sb(p, interaction = TRUE)
for (i in 1 : 10) {
  x <- ask(sa)
  y <- list()
  for (i in names(x)) {
    y[[i]] <- f(x[[i]])
  }
  tell(sa, y)
}
print(sa)
plot(sa)
```

---

sensiFdiv *Sensitivity Indices based on Csiszar f-divergence*

Description

sensiFdiv conducts a density-based sensitivity analysis where the impact of an input variable is defined in terms of dissimilarity between the original output density function and the output density function when the input variable is fixed. The dissimilarity between density functions is measured with Csiszar f-divergences. Estimation is performed through kernel density estimation and the function kde of the package ks.

Usage

```
sensiFdiv(model = NULL, X, fdiv = "TV", nboot = 0, conf = 0.95, ...)
## S3 method for class 'sensiFdiv'
tell(x, y = NULL, ...)
## S3 method for class 'sensiFdiv'
print(x, ...)
## S3 method for class 'sensiFdiv'
plot(x, ylim = c(0, 1), ...)
```

Arguments

model	a function, or a model with a predict method, defining the model to analyze.
X	a matrix or data.frame representing the input random sample.
fdiv	a string or a list of strings specifying the Csiszar f-divergence to be used. Available choices are "TV" (Total-Variation), "KL" (Kullback-Leibler), "Hellinger" and "Chi2" (Neyman chi-squared).
nboot	the number of bootstrap replicates
conf	the confidence level for confidence intervals.
x	a list of class "sensiFdiv" storing the state of the sensitivity study (parameters, data, estimates).
y	a vector of model responses.
ylim	y-coordinate plotting limits.
...	any other arguments for model which are passed unchanged each time it is called.

Details

Some of the Csiszar f-divergences produce sensitivity indices that have already been studied in the context of sensitivity analysis. In particular, "TV" leads to the importance measure proposed by Borgonovo (2007) (up to a constant), "KL" corresponds to the mutual information (Krzykacz-Hausmann 2001) and "Chi2" produces the squared-loss mutual information. See Da Veiga (2015) for details.

Value

sensiFdiv returns a list of class "sensiFdiv", containing all the input arguments detailed before, plus the following components:

call	the matched call.
X	a data.frame containing the design of experiments.
y	a vector of model responses.
S	the estimations of the Csiszar f-divergence sensitivity indices. If several divergences have been selected, Sis a list where each element encompasses the estimations of the sensitivity indices for one of the divergence.

Author(s)

Sebastien Da Veiga, Snecma

References

Borgonovo E. (2007), *A new uncertainty importance measure*, Reliability Engineering and System Safety 92(6), 771–784.

Da Veiga S. (2015), *Global sensitivity analysis with dependence measures*, Journal of Statistical Computation and Simulation, 85(7), 1283–1305.

Krzykacz-Hausmann B. (2001), *Epistemic sensitivity analysis based on the concept of entropy*, Proceedings of SAMO2001, 53–57.

See Also

kde, sensiHSIC

Examples

```
## Not run:
library(ks)

# Test case : the non-monotonic Sobol g-function
n <- 100
X <- data.frame(matrix(runif(8 * n), nrow = n))

# Density-based sensitivity analysis
x <- sensiFdiv(model = sobol.fun, X = X, fdiv = c("TV","KL"), nboot=30)
print(x)

## End(Not run)
```

---

sensiHSIC *Sensitivity Indices based on Hilbert-Schmidt Independence Criterion (HSIC)*

Description

sensiHSIC conducts a sensitivity analysis where the impact of an input variable is defined in terms of the distance between the input/output joint probability distribution and the product of their marginals when they are embedded in a Reproducing Kernel Hilbert Space (RKHS). This distance corresponds to the Hilbert-Schmidt Independence Criterion (HSIC) proposed by Gretton et al. (2005) and serves as a dependence measure between random variables, see Da Veiga (2015) for an illustration in the context of sensitivity analysis.

Usage

```
  sensiHSIC(model = NULL, X, kernelX = "rbf", paramX = NA,
            kernelY = "rbf", paramY = NA, nboot = 0, conf = 0.95, ...)
  ## S3 method for class 'sensiHSIC'
tell(x, y = NULL, ...)
  ## S3 method for class 'sensiHSIC'
print(x, ...)
  ## S3 method for class 'sensiHSIC'
plot(x, ylim = c(0, 1), ...)
```

Arguments

model	a function, or a model with a predict method, defining the model to analyze.
X	a matrix or data.frame representing the input random sample.

`kernelX` a string or a list of strings specifying the reproducing kernel to be used for the input variables. If only one kernel is provided, it is used for all input variables. Available choices are "rbf" (Gaussian), "laplace" (exponential), "dcov" (distance covariance, see details), "raquad" (rationale quadratic), "invmultiquad" (inverse multiquadratic), "linear" (Euclidean scalar product), "matern3" (Matern 3/2), "matern5" (Matern 5/2), "ssanova1" (kernel of Sobolev space of order 1) and "ssanova2" (kernel of Sobolev space of order 2).

`paramX` a scalar or a vector of hyperparameters to be used in the input variable kernels. If only one scalar is provided, it is replicated for all input variables. By default `paramX` is equal to the standard deviation of the input variable for "rbf", "laplace", "raquad", "invmultiquad", "matern3" and "matern5" and to 1 for "dcov". Kernels "linear", "ssanova1" and "ssanova2" do not involve hyperparameters. If `kernelX` is a combination of kernels with and without hyperparameters, `paramX` must have a (dummy) value for the hyperparameter-free kernels, see examples below.

`kernelY` a string specifying the reproducing kernel to be used for the output variable. Available choices are "rbf" (Gaussian), "laplace" (exponential), "dcov" (distance covariance, see details), "raquad" (rationale quadratic), "invmultiquad" (inverse multiquadratic), "linear" (Euclidean scalar product), "matern3" (Matern 3/2), "matern5" (Matern 5/2), "ssanova1" (kernel of Sobolev space of order 1) and "ssanova2" (kernel of Sobolev space of order 2).

`paramY` a scalar to be used in the output variable kernel. By default `paramY` is equal to the standard deviation of the output variable for "rbf", "laplace", "raquad", "invmultiquad", "matern3" and "matern5" and to 1 for "dcov". Kernels "linear", "ssanova1" and "ssanova2" do not involve hyperparameters.

`nboot` the number of bootstrap replicates

`conf` the confidence level for confidence intervals.

`x` a list of class `"sensiHSIC"` storing the state of the sensitivity study (parameters, data, estimates).

`y` a vector of model responses.

`ylim` y-coordinate plotting limits.

`...` any other arguments for `model` which are passed unchanged each time it is called.

Details

The HSIC sensitivity indices are obtained as a normalized version of the Hilbert-Schmidt independence criterion:

$$S_i^{HSIC} = \frac{HSIC(X_i, Y)}{\sqrt{HSIC(X_i, X_i)}\sqrt{HSIC(Y, Y)}},$$

see Da Veiga (2014) for details. When `kernelX="dcov"` and `kernelY="dcov"`, the kernel is given by $k(u, u') = 1/2(||u|| + ||u'|| - ||u - u'||)$ and the sensitivity index is equal to the distance correlation introduced by Szekely et al. (2007) as was recently proven by Sejdinovic et al. (2013).

Value

`sensiHSIC` returns a list of class `"sensiHSIC"`, containing all the input arguments detailed before, plus the following components:

`call`	the matched call.
`X`	a `data.frame` containing the design of experiments.
`y`	a vector of model responses.
`S`	the estimations of HSIC sensitivity indices.

Author(s)

Sebastien Da Veiga, Snecma

References

Da Veiga S. (2015), *Global sensitivity analysis with dependence measures*, Journal of Statistical Computation and Simulation, 85(7), 1283–1305.

Gretton A., Bousquet O., Smola A., Scholkopf B. (2005), *Measuring statistical dependence with hilbert-schmidt norms*, Jain S, Simon H, Tomita E, editors: Algorithmic learning theory, Lecture Notes in Computer Science, Vol. 3734, Berlin: Springer, 63–77.

Sejdinovic D., Sriperumbudur B., Gretton A., Fukumizu K., (2013), *Equivalence of distance-based and RKHS-based statistics in hypothesis testing*, Annals of Statistics 41(5), 2263–2291.

Szekely G.J., Rizzo M.L., Bakirov N.K. (2007), *Measuring and testing dependence by correlation of distances*, Annals of Statistics 35(6), 2769–2794.

See Also

`kde`, `sensiFdiv`

Examples

```
## Not run:
 # Test case : the non-monotonic Sobol g-function
 # Only one kernel is provided with default hyperparameter value
 n <- 100
 X <- data.frame(matrix(runif(8 * n), nrow = n))
 x <- sensiHSIC(model = sobol.fun, X, kernelX = "raquad", kernelY = "rbf")
 print(x)

 # Test case : the Ishigami function
 # A list of kernels is given with default hyperparameter value
 n <- 100
 X <- data.frame(matrix(-pi+2*pi*runif(3 * n), nrow = n))
 x <- sensiHSIC(model = ishigami.fun, X, kernelX = c("rbf","matern3","dcov"),
                kernelY = "rbf")
 print(x)

 # A combination of kernels is given and a dummy value is passed for
 # the first hyperparameter
```

```
  x <- sensiHSIC(model = ishigami.fun, X, kernelX = c("ssanova1","matern3","dcov"),
                 paramX = c(1,2,1), kernelY = "ssanova1")
  print(x)

## End(Not run)
```

`shapleyPermEx` *Estimation of Shapley effects by examining all permutations of inputs (Agorithm of Song et al, 2016), in cases of independent or dependent inputs*

Description

`shapleyPermEx` implements the Monte Carlo estimation of the Shapley effects (Owen, 2014) by examining all permutations of inputs (Song et al., 2016). It also estimates full first order and independent total Sobol' indices (Mara et al., 2015). The function also allows the estimations of all these sensitivity indices in case of dependent inputs. The total cost of this algorithm is $Nv + d! \times (d-1) \times No \times Ni$ model evaluations.

Usage

```
shapleyPermEx(model = NULL, Xall, Xset, d, Nv, No, Ni = 3, colnames = NULL, ...)
## S3 method for class 'shapleyPermEx'
tell(x, y = NULL, return.var = NULL, ...)
## S3 method for class 'shapleyPermEx'
print(x, ...)
## S3 method for class 'shapleyPermEx'
plot(x, ylim = c(0, 1), ...)
```

Arguments

`model`	a function, or a model with a `predict` method, defining the model to analyze.
`Xall`	Xall(n) is a function to generate a n-sample of a d-dimensional input vector (following the required joint distribution).
`Xset`	Xset(n, Sj, Sjc, xjc) is a function to generate a n-sample of a d-dimensional input vector corresponding to the indices in Sj conditional on the input values xjc with the index set Sjc (following the required joint distribution).
`d`	number of inputs.
`Nv`	Monte Carlo sample size to estimate the output variance.
`No`	Outer Monte Carlo sample size to estimate the expectation of the conditional variance of the model output.
`Ni`	Inner Monte Carlo sample size to estimate the conditional variance of the model output.
`colnames`	Optional: A vector containing the names of the inputs.

x	a list of class "shapleyPermEx" storing the state of the sensitivity study (parameters, data, estimates).
y	a vector of model responses.
return.var	a vector of character strings giving further internal variables names to store in the output object x.
ylim	y-coordinate plotting limits.
...	any other arguments for model which are passed unchanged each time it is called.

Details

This function requires R package "gtools".

The computations of the standard errors (and then the confidence intervals) is not realized with this function.

The default values Ni = 3 is the optimal one obtained by the theoretical analysis of Song et al., 2016.

Value

shapleyPermEx returns a list of class "shapleyPermEx", containing all the input arguments detailed before, plus the following components:

call	the matched call.
X	a data.frame containing the design of experiments.
y	the response used.
E	the estimation of the ouput mean.
V	the estimation of the ouput variance.
Shapley	the estimations of the Shapley effects.
SobolS	the estimations of the full first-order Sobol' indices.
SobolT	the estimations of the independent total sensitivity Sobol' indices.

Users can ask more ouput variables with the argument return.var (for example, the list of permutations perms).

Author(s)

Bertrand Iooss, Eunhye Song, Barry L. Nelson, Jeremy Staum

References

B. Iooss and C. Prieur, 2017, *Shapley effects for sensitivity analysis with dependent inputs: comparisons with Sobol' indices, numerical estimation and applications*, 2017, https://hal.inria.fr/hal-01556303.

S. Kucherenko, S. Tarantola, and P. Annoni, 2012, *Estimation of global sensitivity indices for models with dependent variables*, Computer Physics Communications, 183, 937–946.

T. Mara, S. Tarantola, P. Annoni, 2015, *Non-parametric methods for global sensitivity analysis of model output with dependent inputs*, Environmental Modeling & Software 72, 173–183.

A.B. Owen, 2014, *Sobol' indices and Shapley value*, SIAM/ASA Journal of Uncertainty Quantification, 2, 245–251.

A.B. Owen and C. Prieur, 2016, *On Shapley value for measuring importance of dependent inputs*, SIAM/ASA Journal of Uncertainty Quantification, 5, 986–1002.

E. Song, B.L. Nelson, and J. Staum, 2016, *Shapley effects for global sensitivity analysis: Theory and computation*, SIAM/ASA Journal of Uncertainty Quantification, 4, 1060–1083.

See Also

shapleyPermRand

Examples

```
## Not run:

##################################
# Test case : the Ishigami function (3 uniform independent inputs)
# See Iooss and Prieur (2017)

library(gtools)

d <- 3
Xall <- function(n) matrix(runif(d*n,-pi,pi),nc=d)
Xset <- function(n, Sj, Sjc, xjc) matrix(runif(n*length(Sj),-pi,pi),nc=length(Sj))

x <- shapleyPermEx(model = ishigami.fun, Xall=Xall, Xset=Xset, d=d, Nv=1e4, No = 1e3, Ni = 3)
print(x)
plot(x)

##################################
# Test case : Linear model (3 Gaussian inputs including 2 dependent)
# See Iooss and Prieur (2017)

library(gtools)
library(mvtnorm) # Multivariate Gaussian variables
library(condMVNorm) # Conditional multivariate Gaussian variables

modlin <- function(X) apply(X,1,sum)

d <- 3
mu <- rep(0,d)
sig <- c(1,1,2)
ro <- 0.9
Cormat <- matrix(c(1,0,0,0,1,ro,0,ro,1),d,d)
Covmat <- ( sig %*% t(sig) ) * Cormat

Xall <- function(n) mvtnorm::rmvnorm(n,mu,Covmat)
```

```
Xset <- function(n, Sj, Sjc, xjc){
  if (is.null(Sjc)){
    if (length(Sj) == 1){ rnorm(n,mu[Sj],sqrt(Covmat[Sj,Sj]))
    } else{ mvtnorm::rmvnorm(n,mu[Sj],Covmat[Sj,Sj])}
  } else{ condMVNorm::rcmvnorm(n, mu, Covmat, dependent.ind=Sj, given.ind=Sjc, X.given=xjc)}}

x <- shapleyPermEx(model = modlin, Xall=Xall, Xset=Xset, d=d, Nv=1e4, No = 1e3, Ni = 3)
print(x)
plot(x)

## End(Not run)
```

---

shapleyPermRand *Estimation of Shapley effects by random permutations of inputs (Agorithm of Song et al, 2016), in cases of independent or dependent inputs*

Description

shapleyPermRand implements the Monte Carlo estimation of the Shapley effects (Owen, 2014) and their standard errors by randomly sampling permutations of inputs (Song et al., 2016). It also estimates full first order and independent total Sobol' indices (Mara et al., 2015), and their standard errors. The function also allows the estimations of all these sensitivity indices in case of dependent inputs. The total cost of this algorithm is $Nv + m \times (d-1) \times No \times Ni$ model evaluations.

Usage

```
shapleyPermRand(model = NULL, Xall, Xset, d, Nv, m, No = 1, Ni = 3, colnames = NULL, ...)
## S3 method for class 'shapleyPermRand'
tell(x, y = NULL, return.var = NULL, ...)
## S3 method for class 'shapleyPermRand'
print(x, ...)
## S3 method for class 'shapleyPermRand'
plot(x, ylim = c(0, 1), ...)
```

Arguments

model	a function, or a model with a predict method, defining the model to analyze.
Xall	Xall(n) is a function to generate a n-sample of a d-dimensional input vector (following the required joint distribution).
Xset	Xset(n, Sj, Sjc, xjc) is a function to generate a n-sample of a d-dimensional input vector corresponding to the indices in Sj conditional on the input values xjc with the index set Sjc (following the required joint distribution).
d	number of inputs.
Nv	Monte Carlo sample size to estimate the output variance.
m	Number of randomly sampled permutations.

No	Outer Monte Carlo sample size to estimate the expectation of the conditional variance of the model output.
Ni	Inner Monte Carlo sample size to estimate the conditional variance of the model output.
colnames	Optional: A vector containing the names of the inputs.
x	a list of class "shapleyPermRand" storing the state of the sensitivity study (parameters, data, estimates).
y	a vector of model responses.
return.var	a vector of character strings giving further internal variables names to store in the output object x.
ylim	y-coordinate plotting limits.
...	any other arguments for model which are passed unchanged each time it is called.

Details

This function requires R package "gtools".

The computations of the standard errors do not consider the samples to estimate expectation of conditional variances. They are only made regarding the random permutations and are based on the variance of the Monte carlo estimates divided by m. The confidence intervals at 95% correspond to +- 2 standard deviations.

The default values No = 1 and Ni = 3 are the optimal ones obtained by the theoretical analysis of Song et al., 2016.

Value

shapleyPermRand returns a list of class "shapleyPermRand", containing all the input arguments detailed before, plus the following components:

call	the matched call.
X	a data.frame containing the design of experiments.
y	the response used.
E	the estimation of the ouput mean.
V	the estimation of the ouput variance.
Shapley	the estimations of the Shapley effects.
SobolS	the estimations of the full first-order Sobol' indices.
SobolT	the estimations of the independent total sensitivity Sobol' indices.

Users can ask more ouput variables with the argument return.var (for example, the list of permutations perms).

Author(s)

Bertrand Iooss, Eunhye Song, Barry L. Nelson, Jeremy Staum

References

B. Iooss and C. Prieur, 2017, *Shapley effects for sensitivity analysis with dependent inputs: comparisons with Sobol' indices, numerical estimation and applications*, 2017, https://hal.inria.fr/hal-01556303.

S. Kucherenko, S. Tarantola, and P. Annoni, 2012, *Estimation of global sensitivity indices for models with dependent variables*, Computer Physics Communications, 183, 937–946.

T. Mara, S. Tarantola, P. Annoni, 2015, *Non-parametric methods for global sensitivity analysis of model output with dependent inputs*, Environmental Modeling & Software 72, 173–183.

A.B. Owen, 2014, *Sobol' indices and Shapley value*, SIAM/ASA Journal of Uncertainty Quantification, 2, 245–251.

A.B. Owen and C. Prieur, 2016, *On Shapley value for measuring importance of dependent inputs*, SIAM/ASA Journal of Uncertainty Quantification, 5, 986–1002.

E. Song, B.L. Nelson, and J. Staum, 2016, *Shapley effects for global sensitivity analysis: Theory and computation*, SIAM/ASA Journal of Uncertainty Quantification, 4, 1060–1083.

See Also

shapleyPermEx

Examples

```
## Not run:

##################################
# Test case : the Ishigami function
# See Iooss and Prieur (2017)

library(gtools)

d <- 3
Xall <- function(n) matrix(runif(d*n,-pi,pi),nc=d)
Xset <- function(n, Sj, Sjc, xjc) matrix(runif(n*length(Sj),-pi,pi),nc=length(Sj))

x <- shapleyPermRand(model = ishigami.fun, Xall=Xall, Xset=Xset, d=d, Nv=1e4, m=1e4, No = 1, Ni = 3)
print(x)
plot(x)

##################################
# Test case : Linear model (3 Gaussian inputs including 2 dependent)
# See Iooss and Prieur (2017)

library(gtools)
library(mvtnorm) # Multivariate Gaussian variables
library(condMVNorm) # Conditional multivariate Gaussian variables

modlin <- function(X) apply(X,1,sum)

d <- 3
```

```
mu <- rep(0,d)
sig <- c(1,1,2)
ro <- 0.9
Cormat <- matrix(c(1,0,0,0,1,ro,0,ro,1),d,d)
Covmat <- ( sig %*% t(sig) ) * Cormat

Xall <- function(n) mvtnorm::rmvnorm(n,mu,Covmat)

Xset <- function(n, Sj, Sjc, xjc){
  if (is.null(Sjc)){
    if (length(Sj) == 1){ rnorm(n,mu[Sj],sqrt(Covmat[Sj,Sj]))
    } else{ mvtnorm::rmvnorm(n,mu[Sj],Covmat[Sj,Sj])}
  } else{ condMVNorm::rcmvnorm(n, mu, Covmat, dependent.ind=Sj, given.ind=Sjc, X.given=xjc)}}

x <- shapleyPermRand(model = modlin, Xall=Xall, Xset=Xset, d=d, Nv=1e3, m = 1e4, No = 1, Ni = 3)
print(x)
plot(x)

##############################""
# Test case : Multiserver queue model (6 Pert inputs including two dependent pairs)
# See Song, Nelson and Staum (2016)

library(gtools)
library(mc2d) # To generate Pert random variables

d=6

model <-function(x)
{
  # x is a vector of six arrival rates
  JL = cbind(x[,1], x[,1]*0.6 + (x[,4]+x[,6])*0.3, x[,1]*0.4 + x[,2]+x[,3]+x[,5], x[,4]+x[,6],
             (x[,1]*0.4 + x[,2]+x[,3]+x[,5])*0.5
             + (x[,4]+x[,6])*0.7, (x[,1]*0.4 + x[,2]+x[,3]+x[,5])*0.5)
  mu = c(1.2, 1.5, 4, 1.8, 3.6, 1.5)

  rho = t(apply(JL,1,'/',mu))

  return(apply(cbind(rho,x), 1, function(y) sum(y[1:6]/(1-y[1:6]))/sum(y[7:12])*24))
}

Xall <- function(n)
{
  r1 = 0.5
  r2 = -0.5

  # x1 and x2 are correlated
  # convert to Pearson correlation
  r1 = 2 * sin(pi/6*r1)

  z1 = rnorm(n);
  z2 = r1 * z1 + sqrt(1-r1^2) * rnorm(n)

  x1 = qpert(pnorm(z1),0.5,0.6,0.8)
```

```
  x2 = qpert(pnorm(z2),0.5,0.6,0.8)

  # x3 and x4 are correlated
  # convert to Pearson correlation
  r2 = 2 * sin(pi/6*r2)

  z3 = rnorm(n);
  z4 = r2*z3 + sqrt(1-r2^2) * rnorm(n)

  x3 = qpert(pnorm(z3),0.5,0.6,0.8)
  x4 = qpert(pnorm(z4),0.5,0.6,0.8)

  cbind(x1,x2,x3,x4,x5=rpert(n,0.5,0.6,0.8),x6=rpert(n,0.5,0.6,0.8))
}

Xset <- function(n, Sj, Sjc, xjc)
{
  r1 = 0.5
  r2 = -0.5

  # generate a vector of dependent samples of the parameters in Sj
  # All service time distributions are Pert(0.5, 0.6, 0.8) with correlation between
  # (X1, X2) and (X3, X4).

  # Pearson correlation
  r1 = 2 * sin(pi/6*r1)
  r2 = 2 * sin(pi/6*r2)


  z1 = NULL; z2 = NULL;
  z3 = NULL; z4 = NULL;
  RV = NULL

  if(any(Sjc==1))
  {
    x1 = xjc[which(Sjc==1)]
    z1 = qnorm(ppert(x1,0.5,0.6,0.8))
  }

  if(any(Sjc==2))
  {
    x2 = xjc[which(Sjc==2)]
    z2 = qnorm(ppert(x2,0.5,0.6,0.8))
  }

  if(any(Sjc==3))
  {
    x3 = xjc[which(Sjc==3)]
    z3 = qnorm(ppert(x3,0.5,0.6,0.8))
  }

  if(any(Sjc==4))
  {
```

```
  x4 = xjc[which(Sjc==4)]
  z4 = qnorm(ppert(x4,0.5,0.6,0.8))
}

for (i in 1:length(Sj))
{
  index = Sj[i]
  val = NULL

  if(index==1)
  {
    if(is.null(z2))
    {
      val = rpert(n,0.5,0.6,0.8)
      z1 = qnorm(ppert(val,0.5,0.6,0.8))
    }
    else
    {
      z1 = r1 * z2 + sqrt(1-r1^2) * rnorm(n)
      val = qpert(pnorm(z1),0.5,0.6,0.8)
    }
  }
  else if(index ==2)
  {
    if(is.null(z1))
    {
      val = rpert(n,0.5,0.6,0.8)
      z2 = qnorm(ppert(val,0.5,0.6,0.8))
    }
    else
    {
      z2 = r1 * z1 + sqrt(1-r1^2) * rnorm(n)
      val = qpert(pnorm(z2),0.5,0.6,0.8)
    }
  }
  else if(index == 3)
  {
    if(is.null(z4))
    {
      val = rpert(n,0.5,0.6,0.8)
      z3 = qnorm(ppert(val,0.5,0.6,0.8))
    }
    else
    {
      z3 = r2 * z4 + sqrt(1-r2^2) * rnorm(n)
      val = qpert(pnorm(z3),0.5,0.6,0.8)
    }
  }
  else if(index == 4)
  {
    if(is.null(z3))
    {
      val = rpert(n,0.5,0.6,0.8)
```

```
        z4 = qnorm(ppert(val,0.5,0.6,0.8))
      }
      else
      {
        z4 = r2 * z3 + sqrt(1-r2^2) * rnorm(n)
        val = qpert(pnorm(z4),0.5,0.6,0.8)
      }
    }
    else
    {
      val = rpert(n,0.5,0.6,0.8)
    }
    RV <- cbind(RV, val)
  }
  return(RV)
}

x <- shapleyPermRand(model = model, Xall=Xall, Xset=Xset, d=d, Nv=1e3, m = 1e4, No = 1, Ni = 3)
print(x)
plot(x)


## End(Not run)
```

---

sobol *Monte Carlo Estimation of Sobol' Indices*

Description

sobol implements the Monte Carlo estimation of the Sobol' sensitivity indices (standard estimator). This method allows the estimation of the indices of the variance decomposition, sometimes referred to as functional ANOVA decomposition, up to a given order, at a total cost of $(N+1) \times n$ where N is the number of indices to estimate. This function allows also the estimation of the so-called subset (or group) indices, i.e. the first-order indices with respect to single multidimensional inputs.

Usage

```
sobol(model = NULL, X1, X2, order = 1, nboot = 0, conf = 0.95, ...)
## S3 method for class 'sobol'
tell(x, y = NULL, return.var = NULL, ...)
## S3 method for class 'sobol'
print(x, ...)
## S3 method for class 'sobol'
plot(x, ylim = c(0, 1), ...)
```

Arguments

model	a function, or a model with a predict method, defining the model to analyze.
X1	the first random sample.
X2	the second random sample.
order	either an integer, the maximum order in the ANOVA decomposition (all indices up to this order will be computed), or a list of numeric vectors, the multidimensional compounds of the wanted subset indices.
nboot	the number of bootstrap replicates.
conf	the confidence level for bootstrap confidence intervals.
x	a list of class "sobol" storing the state of the sensitivity study (parameters, data, estimates).
y	a vector of model responses.
return.var	a vector of character strings giving further internal variables names to store in the output object x.
ylim	y-coordinate plotting limits.
...	any other arguments for model which are passed unchanged each time it is called.

Value

sobol returns a list of class "sobol", containing all the input arguments detailed before, plus the following components:

call	the matched call.
X	a data.frame containing the design of experiments.
y	a vector of model responses.
V	the estimations of Variances of the Conditional Expectations (VCE) with respect to one factor or one group of factors.
D	the estimations of the terms of the ANOVA decomposition (not for subset indices).
S	the estimations of the Sobol' sensitivity indices (not for subset indices).

Users can ask more ouput variables with the argument return.var (for example, bootstrap outputs V.boot, D.boot and S.boot).

Author(s)

Gilles Pujol

References

I. M. Sobol, 1993, *Sensitivity analysis for non-linear mathematical model*, Math. Modelling Comput. Exp., 1, 407–414.

See Also

sobol2002, sobolSalt, sobol2007, soboljansen, sobolmartinez, sobolEff, sobolSmthSpl, sobolmara, sobo

Examples

```
# Test case : the non-monotonic Sobol g-function

# The method of sobol requires 2 samples
# (there are 8 factors, all following the uniform distribution on [0,1])
library(boot)
n <- 1000
X1 <- data.frame(matrix(runif(8 * n), nrow = n))
X2 <- data.frame(matrix(runif(8 * n), nrow = n))

# sensitivity analysis
x <- sobol(model = sobol.fun, X1 = X1, X2 = X2, order = 2, nboot = 100)
print(x)
#plot(x)
```

---

sobol2002 *Monte Carlo Estimation of Sobol' Indices (scheme by Saltelli 2002)*

Description

sobol2002 implements the Monte Carlo estimation of the Sobol' indices for both first-order and total indices at the same time (alltogether $2p$ indices), at a total cost of $(p+2) \times n$ model evaluations. These are called the Saltelli estimators.

Usage

```
sobol2002(model = NULL, X1, X2, nboot = 0, conf = 0.95, ...)
## S3 method for class 'sobol2002'
tell(x, y = NULL, return.var = NULL, ...)
## S3 method for class 'sobol2002'
print(x, ...)
## S3 method for class 'sobol2002'
plot(x, ylim = c(0, 1), ...)
```

Arguments

model	a function, or a model with a predict method, defining the model to analyze.
X1	the first random sample.
X2	the second random sample.
nboot	the number of bootstrap replicates.
conf	the confidence level for bootstrap confidence intervals.
x	a list of class "sobol" storing the state of the sensitivity study (parameters, data, estimates).

y	a vector of model responses.
return.var	a vector of character strings giving further internal variables names to store in the output object x.
ylim	y-coordinate plotting limits.
...	any other arguments for model which are passed unchanged each time it is called

Details

BE CAREFUL! This estimator suffers from a conditioning problem when estimating the variances behind the indices computations. This can seriously affect the Sobol' indices estimates in case of largely non-centered output. To avoid this effect, you have to center the model output before applying "sobol2002". Functions "sobolEff", "soboljansen" and "sobolmartinez" do not suffer from this problem.

Value

sobol2002 returns a list of class "sobol2002", containing all the input arguments detailed before, plus the following components:

call	the matched call.
X	a data.frame containing the design of experiments.
y	the response used
V	the estimations of Variances of the Conditional Expectations (VCE) with respect to each factor and also with respect to the complementary set of each factor ("all but X_i").
S	the estimations of the Sobol' first-order indices.
T	the estimations of the Sobol' total sensitivity indices.

Users can ask more ouput variables with the argument return.var (for example, bootstrap outputs V.boot, S.boot and T.boot).

Author(s)

Gilles Pujol

References

A. Saltelli, 2002, *Making best use of model evaluations to compute sensitivity indices*, Computer Physics Communication, 145, 580–297.

See Also

sobol, sobolSalt, sobol2007, soboljansen, sobolmartinez, sobolEff, sobolmara, sobolGP, sobolMultOut

Examples

```
# Test case : the non-monotonic Sobol g-function

# The method of sobol requires 2 samples
# There are 8 factors, all following the uniform distribution
# on [0,1]

library(boot)
n <- 1000
X1 <- data.frame(matrix(runif(8 * n), nrow = n))
X2 <- data.frame(matrix(runif(8 * n), nrow = n))

# sensitivity analysis

x <- sobol2002(model = sobol.fun, X1, X2, nboot = 100)
print(x)
plot(x)
```

`sobol2007` *Monte Carlo Estimation of Sobol' Indices (improved formulas of Mauntz: Sobol et al. (2007) and Saltelli et al. (2010))*

Description

`sobol2007` implements the Monte Carlo estimation of the Sobol' indices for both first-order and total indices at the same time (alltogether $2p$ indices), at a total cost of $(p+2) \times n$ model evaluations. These are called the Mauntz estimators.

Usage

```
sobol2007(model = NULL, X1, X2, nboot = 0, conf = 0.95, ...)
## S3 method for class 'sobol2007'
tell(x, y = NULL, return.var = NULL, ...)
## S3 method for class 'sobol2007'
print(x, ...)
## S3 method for class 'sobol2007'
plot(x, ylim = c(0, 1), ...)
```

Arguments

`model`	a function, or a model with a `predict` method, defining the model to analyze.
`X1`	the first random sample.
`X2`	the second random sample.
`nboot`	the number of bootstrap replicates.
`conf`	the confidence level for bootstrap confidence intervals.
`x`	a list of class `"sobol"` storing the state of the sensitivity study (parameters, data, estimates).

y	a vector of model responses.
return.var	a vector of character strings giving further internal variables names to store in the output object x.
ylim	y-coordinate plotting limits.
...	any other arguments for model which are passed unchanged each time it is called

Details

This estimator is good for small first-order and total indices.

BE CAREFUL! This estimator suffers from a conditioning problem when estimating the variances behind the indices computations. This can seriously affect the Sobol' indices estimates in case of largely non-centered output. To avoid this effect, you have to center the model output before applying "sobol2007". Functions "sobolEff", "soboljansen" and "sobolmartinez" do not suffer from this problem.

Value

sobol2007 returns a list of class "sobol2007", containing all the input arguments detailed before, plus the following components:

call	the matched call.
X	a data.frame containing the design of experiments.
y	the response used
V	the estimations of Variances of the Conditional Expectations (VCE) with respect to each factor and also with respect to the complementary set of each factor ("all but X_i").
S	the estimations of the Sobol' first-order indices.
T	the estimations of the Sobol' total sensitivity indices.

Users can ask more ouput variables with the argument return.var (for example, bootstrap outputs V.boot, S.boot and T.boot).

Author(s)

Bertrand Iooss

References

I.M. Sobol, S. Tarantola, D. Gatelli, S.S. Kucherenko and W. Mauntz, 2007, *Estimating the approximation errors when fixing unessential factors in global sensitivity analysis*, Reliability Engineering and System Safety, 92, 957–960.

A. Saltelli, P. Annoni, I. Azzini, F. Campolongo, M. Ratto and S. Tarantola, 2010, *Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index*, Computer Physics Communications 181, 259–270.

See Also

sobol, sobol2002, sobolSalt, soboljansen, sobolmartinez, sobolEff, sobolmara,sobolMultOut

Examples

```
# Test case : the non-monotonic Sobol g-function

# The method of sobol requires 2 samples
# There are 8 factors, all following the uniform distribution
# on [0,1]

library(boot)
n <- 1000
X1 <- data.frame(matrix(runif(8 * n), nrow = n))
X2 <- data.frame(matrix(runif(8 * n), nrow = n))

# sensitivity analysis

x <- sobol2007(model = sobol.fun, X1, X2, nboot = 100)
print(x)
plot(x)
```

---

sobolEff *Monte Carlo Estimation of Sobol' Indices (formulas of Janon-Monod)*

Description

`sobolEff` implements the Monte Carlo estimation of the Sobol' sensitivity indices using the asymptotically efficient formulas in section 4.2.4.2 of Monod et al. (2006). Either all first-order indices or all total-effect indices are estimated at a cost of $N \times (p+1)$ model calls or all closed second-order indices are estimated at a cost of $\binom{N \times p}{2}$ model calls.

Usage

```
sobolEff(model = NULL, X1, X2, order=1, nboot = 0, conf = 0.95, ...)
## S3 method for class 'sobolEff'
tell(x, y = NULL, ...)
## S3 method for class 'sobolEff'
print(x, ...)
## S3 method for class 'sobolEff'
plot(x, ylim = c(0, 1), ...)
```

Arguments

`model`	a function, or a model with a `predict` method, defining the model to analyze.
`X1`	the first random sample.
`X2`	the second random sample.
`order`	an integer specifying the indices to estimate: 0 for total effect indices,1 for first-order indices and 2 for closed second-order indices.
`nboot`	the number of bootstrap replicates, or zero to use asymptotic standard deviation estimates given in Janon et al. (2012).

conf	the confidence level for confidence intervals.
x	a list of class "sobolEff" storing the state of the sensitivity study (parameters, data, estimates).
y	a vector of model responses.
ylim	y-coordinate plotting limits.
...	any other arguments for model which are passed unchanged each time it is called.

Details

The estimator used by sobolEff is defined in Monod et al. (2006), Section 4.2.4.2 and studied under the name T_N in Janon et al. (2012). This estimator is good for large first-order indices.

Value

sobolEff returns a list of class "sobolEff", containing all the input arguments detailed before, plus the following components:

call	the matched call.
X	a data.frame containing the design of experiments.
y	a vector of model responses.
S	the estimations of the Sobol' sensitivity indices.

Author(s)

Alexandre Janon, Laurent Gilquin

References

Monod, H., Naud, C., Makowski, D. (2006), Uncertainty and sensitivity analysis for crop models in Working with Dynamic Crop Models: Evaluation, Analysis, Parameterization, and Applications, Elsevier.

A. Janon, T. Klein, A. Lagnoux, M. Nodet, C. Prieur (2014), *Asymptotic normality and efficiency of two Sobol index estimators*, ESAIM: Probability and Statistics, 18:342-364.

See Also

sobol, sobol2002, sobolSalt, sobol2007, soboljansen, sobolmartinez, sobolSmthSpl

Examples

```
# Test case : the non-monotonic Sobol g-function

# The method of sobol requires 2 samples
# (there are 8 factors, all following the uniform distribution on [0,1])
n <- 1000
X1 <- data.frame(matrix(runif(8 * n), nrow = n))
X2 <- data.frame(matrix(runif(8 * n), nrow = n))
```

```
# sensitivity analysis
x <- sobolEff(model = sobol.fun, X1 = X1, X2 = X2, nboot = 0)
print(x)
```

---

sobolGP *Kriging-based sensitivity analysis*

Description

Perform a kriging-based global sensitivity analysis taking into account both the meta-model and the Monte-Carlo errors. The Sobol indices are estimated with a Monte-Carlo integration and the true function is substituted by a kriging model. It is built thanks to the function km of the package DiceKriging. The complete conditional predictive distribution of the kriging model is considered (not only the predictive mean).

Usage

```
sobolGP(
model,
type="SK",
MCmethod="sobol",
X1,
X2,
nsim=100,
nboot=1,
conf = 0.95,
sequential = FALSE,
candidate,
sequential.tot=FALSE,
max_iter = 1000)

## S3 method for class 'sobolGP'
ask(x, tot = FALSE, ...)

## S3 method for class 'sobolGP'
tell(x,y=NULL,xpoint=NULL,newcandidate=NULL, ...)

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

## S3 method for class 'sobolGP'
plot(x,...)
```

Arguments

model	an object of class "km" specifying the kriging model built from package "DiceKriging" (see km).

type	a character string giving the type of the considered kriging model. "SK" refers to simple kriging and "UK" refers to universal kriging (see km).
MCmethod	a character string specifying the Monte-Carlo procedure used to estimate the Sobol indices. The avaible methods are : "sobol", "sobol2002", "sobol2007", "sobolEff" and "soboljansen".
X1	a matrix representing the first random sample.
X2	a matrix representing the second random sample.
nsim	an integer giving the number of samples for the conditional Gaussian process. It is used to quantify the uncertainty due to the kriging approximation.
nboot	an integer representing the number of bootstrap replicates. It is used to quantify the uncertainty due to the Monte-Carlo integrations. We recommend to set nboot = 100.
conf	a numeric representing the confidence intervals taking into account the uncertainty due to the bootstrap procedure and the Gaussian process samples.
sequential	a boolean. If sequential=TRUE, the procedure provides a new point where to perform a simulation. It is the one minimizing the sum of the MAIN effect estimate variances. The variance is taken with respect to the conditional Gaussian process. The new point is selected in the points candidate.
candidate	a matrix representing the candidate points where the best new point to be simulated is selected. The lines represent the points and the columns represent the dimension.
sequential.tot	a boolean. If sequential.tot=TRUE, the procedure provides a new point where to perform the simulation. It is the one minimizing the sum of the TOTAL effect estimate. The variance is taken with respect to the conditional Gaussian process. The new point is selected in the points candidate.
max_iter	a numeric giving the maximal number of iterations for the propagative Gibbs sampler. It is used to simulate the realizations of the Gaussian process.
x	an object of class S3 "sobolGP" obtaining from the procedure sobolGP. It stores the results of the Kriging-based global sensitivity analysis.
tot	a boolean. If tot=TRUE, the procedure ask provides a point relative to the uncertainty of the total Sobol' indices (instead of first order' ones).
xpoint	a matrix representing a new point added to the kriging model.
y	a numeric giving the response of the function at xpoint.
newcandidate	a matrix representing the new candidate points where the best point to be simulated is selected. If newcandidate=NULL, these points correspond to candidate without the new point xpoint.
...	any other arguments to be passed

Details

The function ask provides the new point where the function should be simulated. Furthermore, the function tell performs a new kriging-based sensitivity analysis when the point x with the corresponding observation y is added.

Value

An object of class S3 `sobolGP`.

- call : a list containing the arguments of the function `sobolGP` :
  - X1 : X1
  - X2 : X2
  - conf : conf
  - nboot : nboot
  - candidate : candidate
  - sequential : sequential
  - max_iter : max_iter
  - sequential.tot : sequential.tot
  - model : model
  - tot : tot
  - method : MCmethod
  - type : type
  - nsim : nsim
- S : a list containing the results of the kriging-based sensitivity analysis for the MAIN effects:
  - mean : a matrix giving the mean of the Sobol index estimates.
  - var : a matrix giving the variance of the Sobol index estimates.
  - ci : a matrix giving the confidence intervals of the Sobol index estimates according to `conf`.
  - varPG : a matrix giving the variance of the Sobol index estimates due to the Gaussian process approximation.
  - varMC : a matrix giving the variance of the Sobol index estimates due to the Monte-Carlo integrations.
  - xnew : if `sequential=TRUE`, a matrix giving the point in `candidate` which is the best to simulate.
  - xnewi : if `sequential=TRUE`, an integer giving the index of the point in `candidate` which is the best to simulate.
- T : a list containing the results of the kriging-based sensitivity analysis for the TOTAL effects:
  - mean : a matrix giving the mean of the Sobol index estimates.
  - var : a matrix giving the variance of the Sobol index estimates.
  - ci : a matrix giving the confidence intervals of the Sobol index estimates according to `conf`.
  - varPG : a matrix giving the variance of the Sobol index estimates due to the Gaussian process approximation.
  - varMC : a matrix giving the variance of the Sobol index estimates due to the Monte-Carlo integrations.
  - xnew : if `sequential.tot=TRUE`, a matrix giving the point in `candidate` which is the best to simulate.
  - xnewi : if `sequential.tot=TRUE`, an integer giving the index of the point in `candidate` which is the best to simulate.

Author(s)

Loic Le Gratiet, EDF R&D - CNRS, I3S

References

L. Le Gratiet, C. Cannamela and B. Iooss (2014), A Bayesian approach for global sensitivity analysis of (multifidelity) computer codes, SIAM/ASA J. Uncertainty Quantification 2-1, pp. 336-363.

See Also

sobol, sobol2002, sobol2007, sobolEff, soboljansen,sobolMultOut, km

Examples

```
## Not run:
library(DiceKriging)

#--------------------------------------#
# kriging model building
#--------------------------------------#

d <- 2; n <- 16
design.fact <- expand.grid(x1=seq(0,1,length=4), x2=seq(0,1,length=4))
y <- apply(design.fact, 1, branin)

m <- km(design=design.fact, response=y)

#--------------------------------------#
# sobol samples & candidate points
#--------------------------------------#

n <- 1000
X1 <- data.frame(matrix(runif(d * n), nrow = n))
X2 <- data.frame(matrix(runif(d * n), nrow = n))

candidate <- data.frame(matrix(runif(d * 100), nrow = 100))

#--------------------------------------#
# Kriging-based Sobol
#--------------------------------------#

res <- sobolGP(
model = m,
type="UK",
MCmethod="sobol",
X1,
X2,
nsim = 100,
conf = 0.95,
nboot=100,
sequential = TRUE,
```

```
candidate,
sequential.tot=FALSE,
max_iter = 1000
)

res
plot(res)
x <- ask(res)
y <- branin(x)
res.new <- tell(res,y,x)
res.new

## End(Not run)
```

`soboljansen` *Monte Carlo Estimation of Sobol' Indices (improved formulas of Jansen (1999) and Saltelli et al. (2010))*

Description

`soboljansen` implements the Monte Carlo estimation of the Sobol' indices for both first-order and total indices at the same time (alltogether $2p$ indices), at a total cost of $(p+2) \times n$ model evaluations. These are called the Jansen estimators.

Usage

```
soboljansen(model = NULL, X1, X2, nboot = 0, conf = 0.95, ...)
## S3 method for class 'soboljansen'
tell(x, y = NULL, return.var = NULL, ...)
## S3 method for class 'soboljansen'
print(x, ...)
## S3 method for class 'soboljansen'
plot(x, ylim = c(0, 1), y_col = NULL, y_dim3 = NULL, ...)
```

Arguments

`model`	a function, or a model with a `predict` method, defining the model to analyze.
`X1`	the first random sample.
`X2`	the second random sample.
`nboot`	the number of bootstrap replicates.
`conf`	the confidence level for bootstrap confidence intervals.
`x`	a list of class `"sobol"` storing the state of the sensitivity study (parameters, data, estimates).
`y`	a vector of model responses.
`return.var`	a vector of character strings giving further internal variables names to store in the output object `x`.

ylim	y-coordinate plotting limits.
y_col	an integer defining the index of the column of x$y to be used for plotting the corresponding sensitivity indices (only applies if x$y is a matrix or an array). If set to NULL (as per default) and x$y is a matrix or an array, the first column (respectively the first element in the second dimension) of x$y is used (i.e. y_col = 1).
y_dim3	an integer defining the index in the third dimension of x$y to be used for plotting the corresponding sensitivity indices (only applies if x$y is an array). If set to NULL (as per default) and x$y is a three-dimensional array, the first element in the third dimension of x$y is used (i.e. y_dim3 = 1).
...	for soboljansen: any other arguments for model which are passed unchanged each time it is called.

Details

This estimator is good for large first-order indices, and (large and small) total indices.

This version of soboljansen also supports matrices and three-dimensional arrays as output of model. If the model output is a matrix or an array, V, S and T are matrices or arrays as well (depending on the type of y and the value of nboot).

The bootstrap outputs V.boot, S.boot and T.boot can only be returned if the model output is a vector (using argument return.var). For matrix or array output, these objects can't be returned.

Value

soboljansen returns a list of class "soboljansen", containing all the input arguments detailed before, plus the following components:

call	the matched call.
X	a data.frame containing the design of experiments.
y	either a vector, a matrix or a three-dimensional array of model responses (depends on the output of model).
V	the estimations of Variances of the Conditional Expectations (VCE) with respect to each factor and also with respect to the complementary set of each factor ("all but X_i").
S	the estimations of the Sobol' first-order indices.
T	the estimations of the Sobol' total sensitivity indices.

Users can ask more ouput variables with the argument return.var (for example, bootstrap outputs V.boot, S.boot and T.boot).

Author(s)

Bertrand Iooss, with contributions from Frank Weber (2016)

References

M.J.W. Jansen, 1999, *Analysis of variance designs for model output*, Computer Physics Communication, 117, 35–43.

A. Saltelli, P. Annoni, I. Azzini, F. Campolongo, M. Ratto and S. Tarantola, 2010, *Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index*, Computer Physics Communications 181, 259–270.

See Also

sobol, sobol2002, sobolSalt, sobol2007, sobolmartinez, sobolEff, sobolmara,sobolMultOut

Examples

```
# Test case : the non-monotonic Sobol g-function

# The method of sobol requires 2 samples
# There are 8 factors, all following the uniform distribution
# on [0,1]

library(boot)
n <- 1000
X1 <- data.frame(matrix(runif(8 * n), nrow = n))
X2 <- data.frame(matrix(runif(8 * n), nrow = n))

# sensitivity analysis

x <- soboljansen(model = sobol.fun, X1, X2, nboot = 100)
print(x)
plot(x)

# Only for demonstration purposes: a model function returning a matrix
sobol.fun_matrix <- function(X){
  res_vector <- sobol.fun(X)
  cbind(res_vector, 2 * res_vector)
}
x_matrix <- soboljansen(model = sobol.fun_matrix, X1, X2)
plot(x_matrix, y_col = 2)
title(main = "y_col = 2")

# Also only for demonstration purposes: a model function returning a
# three-dimensional array
sobol.fun_array <- function(X){
  res_vector <- sobol.fun(X)
  res_matrix <- cbind(res_vector, 2 * res_vector)
  array(data = c(res_matrix, 5 * res_matrix),
        dim = c(length(res_vector), 2, 2))
}
x_array <- soboljansen(model = sobol.fun_array, X1, X2)
plot(x_array, y_col = 2, y_dim3 = 2)
title(main = "y_col = 2, y_dim3 = 2")
```

sobolmara *Monte Carlo Estimation of Sobol' Indices via matrix permutations*

Description

sobolmara implements the Monte Carlo estimation of the first-order Sobol' sensitivity indices using the formula of Mara and Joseph (2008), called the Mara estimator. This method allows the estimation of all first-order p indices at a cost of 2N model calls (the random sample size), then independently of p (the number of inputs).

Usage

```
sobolmara(model = NULL, X1, ...)
## S3 method for class 'sobolmara'
tell(x, y = NULL, return.var = NULL, ...)
## S3 method for class 'sobolmara'
print(x, ...)
## S3 method for class 'sobolmara'
plot(x, ylim = c(0, 1), ...)
```

Arguments

model	a function, or a model with a predict method, defining the model to analyze.
X1	the random sample.
x	a list of class "sobolEff" storing the state of the sensitivity study (parameters, data, estimates).
y	a vector of model responses.
return.var	a vector of character strings giving further internal variables names to store in the output object x.
ylim	y-coordinate plotting limits.
...	any other arguments for model which are passed unchanged each time it is called.

Details

The estimator used by sobolmara is based on rearragement of a unique matrix via random permutations (see Mara and Joseph, 2008). Bootstrap confidence intervals are not available.

Value

sobolmara returns a list of class "sobolmara", containing all the input arguments detailed before, plus the following components:

call	the matched call.
X	a data.frame containing the design of experiments.
y	a vector of model responses.
S	the estimations of the Sobol' sensitivity indices.

Author(s)

Bertrand Iooss

References

Mara, T. and Joseph, O.R. (2008), *Comparison of some efficient methods to evaluate the main effect of computer model factors*, Journal of Statistical Computation and Simulation, 78:167–178

See Also

sobolroalhs, sobol, sobol2002, sobolSalt, sobol2007, soboljansen, sobolmartinez, sobolEff, sobolMultOu

Examples

```
# Test case : the non-monotonic Sobol g-function

# The method of sobolmara requires 1 sample
# (there are 8 factors, all following the uniform distribution on [0,1])
n <- 1000
X1 <- data.frame(matrix(runif(8 * n), nrow = n))

# sensitivity analysis
x <- sobolmara(model = sobol.fun, X1 = X1)
print(x)
plot(x)
```

---

sobolmartinez *Monte Carlo Estimation of Sobol' Indices (formulas of Martinez (2011))*

Description

sobolmartinez implements the Monte Carlo estimation of the Sobol' indices for both first-order and total indices using correlation coefficients-based formulas, at a total cost of $(p + 2) \times n$ model evaluations. These are called the Martinez estimators.

Usage

```
sobolmartinez(model = NULL, X1, X2, nboot = 0, conf = 0.95, ...)
## S3 method for class 'sobolmartinez'
tell(x, y = NULL, return.var = NULL, ...)
## S3 method for class 'sobolmartinez'
print(x, ...)
## S3 method for class 'sobolmartinez'
plot(x, ylim = c(0, 1), y_col = NULL, y_dim3 = NULL, ...)
```

Arguments

model	a function, or a model with a predict method, defining the model to analyze.
X1	the first random sample.
X2	the second random sample.
nboot	the number of bootstrap replicates, or zero to use theoretical formulas based on confidence interfaces of correlation coefficient (Martinez, 2011).
conf	the confidence level for bootstrap confidence intervals.
x	a list of class "sobol" storing the state of the sensitivity study (parameters, data, estimates).
y	a vector of model responses.
return.var	a vector of character strings giving further internal variables names to store in the output object x.
ylim	y-coordinate plotting limits.
y_col	an integer defining the index of the column of x$y to be used for plotting the corresponding sensitivity indices (only applies if x$y is a matrix or an array). If set to NULL (as per default) and x$y is a matrix or an array, the first column (respectively the first element in the second dimension) of x$y is used (i.e. y_col = 1).
y_dim3	an integer defining the index in the third dimension of x$y to be used for plotting the corresponding sensitivity indices (only applies if x$y is an array). If set to NULL (as per default) and x$y is a three-dimensional array, the first element in the third dimension of x$y is used (i.e. y_dim3 = 1).
...	for sobolmartinez: any other arguments for model which are passed unchanged each time it is called

Details

This estimator supports missing values (NA or NaN) which can occur during the simulation of the model on the design of experiments (due to code failure) even if Sobol' indices are no more rigorous variance-based sensitivity indices if missing values are present. In this case, a warning is displayed.

This version of sobolmartinez also supports matrices and three-dimensional arrays as output of model. Bootstrapping (including bootstrap confidence intervals) is also supported for matrix or array output. However, theoretical confidence intervals (for nboot = 0) are only supported for vector output. If the model output is a matrix or an array, V, S and T are matrices or arrays as well (depending on the type of y and the value of nboot).

The bootstrap outputs V.boot, S.boot and T.boot can only be returned if the model output is a vector (using argument return.var). For matrix or array output, these objects can't be returned.

Value

sobolmartinez returns a list of class "sobolmartinez", containing all the input arguments detailed before, plus the following components:

call	the matched call.
X	a data.frame containing the design of experiments.

y	either a vector, a matrix or a three-dimensional array of model responses (depends on the output of model).
V	the estimations of normalized variances of the Conditional Expectations (VCE) with respect to each factor and also with respect to the complementary set of each factor ("all but X_i").
S	the estimations of the Sobol' first-order indices.
T	the estimations of the Sobol' total sensitivity indices.

Users can ask more ouput variables with the argument return.var (for example, bootstrap outputs V.boot, S.boot and T.boot).

Author(s)

Bertrand Iooss, with contributions from Frank Weber (2016)

References

J-M. Martinez, 2011, *Analyse de sensibilite globale par decomposition de la variance*, Presentation in the meeting of GdR Ondes and GdR MASCOT-NUM, January, 13th, 2011, Institut Henri Poincare, Paris, France.

M. Baudin, K. Boumhaout, T. Delage, B. Iooss and J-M. Martinez, 2016, Numerical stability of Sobol' indices estimation formula, Proceedings of the SAMO 2016 Conference, Reunion Island, France, December 2016

See Also

sobol, sobol2002, sobolSalt, sobol2007, soboljansen, soboltouati, sobolEff, sobolmara, sobolMult

Examples

```
# Test case : the non-monotonic Sobol g-function

# The method of sobol requires 2 samples
# There are 8 factors, all following the uniform distribution
# on [0,1]

library(boot)
n <- 1000
X1 <- data.frame(matrix(runif(8 * n), nrow = n))
X2 <- data.frame(matrix(runif(8 * n), nrow = n))

# sensitivity analysis

x <- sobolmartinez(model = sobol.fun, X1, X2, nboot = 0)
print(x)
plot(x)

# Only for demonstration purposes: a model function returning a matrix
sobol.fun_matrix <- function(X){
  res_vector <- sobol.fun(X)
```

```
  cbind(res_vector, 2 * res_vector)
}
x_matrix <- sobolmartinez(model = sobol.fun_matrix, X1, X2)
plot(x_matrix, y_col = 2)
title(main = "y_col = 2")

# Also only for demonstration purposes: a model function returning a
# three-dimensional array
sobol.fun_array <- function(X){
  res_vector <- sobol.fun(X)
  res_matrix <- cbind(res_vector, 2 * res_vector)
  array(data = c(res_matrix, 5 * res_matrix),
        dim = c(length(res_vector), 2, 2))
}
x_array <- sobolmartinez(model = sobol.fun_array, X1, X2)
plot(x_array, y_col = 2, y_dim3 = 2)
title(main = "y_col = 2, y_dim3 = 2")
```

`sobolMultOut` *Monte Carlo Estimation of Aggregated Sobol' Indices for multiple and functional outputs*

Description

`sobolMultOut` implements the aggregated Sobol' indices for multiple outputs. It consists in averaging all the Sobol indices weighted by the variance of their corresponding output. Moreover, this function computes and plots the functional (unidimensional) Sobol' indices for functional (unidimensional) model output. Sobol' indices for both first-order and total indices are estimated by Monte Carlo formulas.

Usage

```
sobolMultOut(model = NULL, q = 1, X1, X2, MCmethod = "sobol",
                         plotFct=FALSE, ...)
## S3 method for class 'sobolMultOut'
print(x, ...)
## S3 method for class 'sobolMultOut'
plot(x, ylim = c(0, 1), ...)
```

Arguments

`model`	a function, or a model with a `predict` method, defining the model to analyze.
`q`	dimension of the model output vector.
`X1`	the first random sample.
`X2`	the second random sample.
`MCmethod`	a character string specifying the Monte-Carlo procedure used to estimate the Sobol indices. The avaible methods are : `"sobol"`, `"sobol2002"`, `"sobol2007"`, `"soboljansen"`, `sobolmara` and `sobolGP`.

plotFct	if TRUE, 1D functional Sobol indices are computed and plotted in an external window (default=FALSE).
x	a list of class MCmethod storing the state of the sensitivity study (parameters, data, estimates).
ylim	y-coordinate plotting limits.
...	any other arguments for model which are passed unchanged each time it is called

Details

For this function, there are several gaps: the bootstrap estimation of confidence intervals is not avalaible and the tell function does not work.

Value

sobolMultOut returns a list of class MCmethod, containing all its input arguments, plus the following components:

call	the matched call.
X	a data.frame containing the design of experiments.
y	the response used
V	the estimations of the aggregated Variances of the Conditional Expectations (VCE) with respect to each factor and also with respect to the complementary set of each factor ("all but X_i").
S	the estimations of the aggregated Sobol' first-order indices.
T	the estimations of the aggregated Sobol' total sensitivity indices.
Sfct	the estimations of the functional Sobol' first-order indices (if PlotFct=TRUE).
Tfct	the estimations of the functional Sobol' total sensitivity indices (if PlotFct=TRUE).

Author(s)

Bertrand Iooss

References

M. Lamboni, H. Monod and D. Makowski, 2011, *Multivariate sensitivity analysis to measure global contribution of input factors in dynamic models*, Reliability Engineering and System Safety, 96:450-459.

F. Gamboa, A. Janon, T. Klein and A. Lagnoux, 2014, *Sensitivity indices for multivariate outputs*, Electronic Journal of Statistics, 8:575-603.

See Also

sobol, sobol2002, sobol2007, soboljansen, sobolmara, sobolGP

Examples

```
## Not run:
# Functional toy function: Arctangent temporal function (Auder, 2011)
# X: input matrix (in [-7,7]^2)
# q: number of discretization steps of [0,2pi] interval
# output: vector of q values

atantemp <- function(X, q = 100){

  n <- dim(X)[[1]]
  t <- (0:(q-1)) * (2*pi) / (q-1)

  res <- matrix(0,ncol=q,nrow=n)
  for (i in 1:n) res[i,] <- atan(X[i,1]) * cos(t) + atan(X[i,2]) * sin(t)

  return(res)
}

# Tests functional toy fct

y0 <- atantemp(matrix(c(-7,0,7,-7,0,7),ncol=2))
#plot(y0[1,],type="l")
#apply(y0,1,lines)

n <- 100
X <- matrix(c(runif(2*n,-7,7)),ncol=2)
y <- atantemp(X)
x11()
plot(y0[2,],ylim=c(-2,2),type="l")
apply(y,1,lines)

# Sobol indices computations

n <- 1000
X1 <- data.frame(matrix(runif(2*n,-7,7), nrow = n))
X2 <- data.frame(matrix(runif(2*n,-7,7), nrow = n))

x11()
sa <- sobolMultOut(model=atantemp, q=100, X1, X2,
                   MCmethod="soboljansen", plotFct=T)
print(sa)
x11()
plot(sa)


## End(Not run)
```

---

sobolowen *Monte Carlo Estimation of Sobol' Indices (improved formulas of Owen (2013)*

Description

sobolowen implements the Monte Carlo estimation of the Sobol' indices for both first-order and total indices at the same time (alltogether $2p$ indices). Take as input 3 independent matrices. These are called the Owen estimators.

Usage

```
sobolowen(model = NULL, X1, X2, X3, nboot = 0, conf = 0.95, varest = 2, ...)
## S3 method for class 'sobolowen'
tell(x, y = NULL, return.var = NULL, varest = 2, ...)
## S3 method for class 'sobolowen'
print(x, ...)
## S3 method for class 'sobolowen'
plot(x, ylim = c(0, 1), ...)
```

Arguments

model	a function, or a model with a predict method, defining the model to analyze.
X1	the first random sample.
X2	the second random sample.
X3	the third random sample.
nboot	the number of bootstrap replicates.
conf	the confidence level for bootstrap confidence intervals.
varest	choice for the variance estimator for the denominator of the Sobol' indices. varest=1 is for a classical estimator. varest=2 (default) is for the estimator proposed in Janon et al. (2012).
x	a list of class "sobol" storing the state of the sensitivity study (parameters, data, estimates).
y	a vector of model responses.
return.var	a vector of character strings giving further internal variables names to store in the output object x.
ylim	y-coordinate plotting limits.
...	any other arguments for model which are passed unchanged each time it is called

Value

sobolowen returns a list of class "sobolowen", containing all the input arguments detailed before, plus the following components:

call	the matched call.
X	a data.frame containing the design of experiments.
y	the response used
V	the estimations of Variances of the Conditional Expectations (VCE) with respect to each factor and also with respect to the complementary set of each factor ("all but X_i").

S	the estimations of the Sobol' first-order indices.
T	the estimations of the Sobol' total sensitivity indices.

Users can ask more ouput variables with the argument `return.var` (for example, bootstrap outputs `V.boot`, `S.boot` and `T.boot`).

Author(s)

Taieb Touati and Bernardo Ramos

References

A. Owen, 2013, *Better estimations of small Sobol' sensitivity indices*, ACM Transactions on Modeling and Computer Simulations (TOMACS), 23(2), 11.

Janon, A., Klein T., Lagnoux A., Nodet M., Prieur C. (2012), Asymptotic normality and efficiency of two Sobol index estimators. Accepted in ESAIM: Probability and Statistics.

See Also

`sobol`, `sobol2002`, `sobolSalt`, `sobol2007`, `soboljansen`, `sobolmartinez`, `sobolEff`, `sobolmara`, `sobolGP`

Examples

```
# Test case : the non-monotonic Sobol g-function

# The method of sobolowen requires 3 samples
# There are 8 factors, all following the uniform distribution
# on [0,1]

library(boot)
n <- 1000
X1 <- data.frame(matrix(runif(8 * n), nrow = n))
X2 <- data.frame(matrix(runif(8 * n), nrow = n))
X3 <- data.frame(matrix(runif(8 * n), nrow = n))

# sensitivity analysis

## Not run:
x <- sobolowen(model = sobol.fun, X1, X2, X3, nboot = 100)
print(x)
plot(x)

## End(Not run)
```

`sobolroalhs` *Sobol' Indices Estimation Using Replicated OA-based LHS*

Description

`sobolroalhs` implements the estimation of the Sobol' sensitivity indices introduced by Tissot & Prieur (2015) using two replicated designs (Latin hypercubes or orthogonal arrays). This function estimates either all first-order indices or all closed second-order indices at a total cost of $2 \times N$ model evaluations. For closed second-order indices $N = q^2$ where $q \geq d - 1$ is a prime number corresponding to the number of levels of the orthogonal array, and where d indicates the number of factors.

Usage

```
sobolroalhs(model = NULL, factors, N, p=1, order, tail=TRUE, conf=0.95, nboot=0, ...)
## S3 method for class 'sobolroalhs'
tell(x, y = NULL, ...)
## S3 method for class 'sobolroalhs'
print(x, ...)
## S3 method for class 'sobolroalhs'
plot(x, ylim = c(0,1), ...)
```

Arguments

`model`	a function, or a model with a `predict` method, defining the model to analyze.
`factors`	an integer giving the number of factors, or a vector of character strings giving their names.
`N`	an integer giving the size of each replicated design (for a total of $2 \times N$ model evaluations).
`p`	an integer giving the number of model outputs.
`order`	an integer giving the order of the indices (1 or 2).
`tail`	a boolean specifying the method used to choose the number of levels of the orthogonal array (see "Warning messages").
`conf`	the confidence level for confidence intervals.
`nboot`	the number of bootstrap replicates.
`x`	a list of class `"sobolroalhs"` storing the state of the sensitivity study (parameters, data, estimates).
`y`	a vector of model responses.
`ylim`	y-coordinate plotting limits.
`...`	any other arguments for `model` which are passed unchanged each time it is called.

Details

`sobolroalhs` automatically assigns a uniform distribution on [0,1] to each input. Transformations of distributions (between U[0,1] and the wanted distribution) have to be realized before the call to tell() (see "Examples").

Missing values (i.e `NA` values) in outputs are automatically handled by the function.

This function also supports multidimensional outputs (matrices in `y` or as output of `model`). In this case, aggregated Sobol' indices are returned (`see sobolMultOut`).

Value

`sobolroalhs` returns a list of class `"sobolroalhs"`, containing all the input arguments detailed before, plus the following components:

`call`	the matched call.
`X`	a `data.frame` containing the design of experiments (row concatenation of the two replicated designs).
`y`	the responses used.
`OA`	the orthogonal array constructed (`NULL` if `order=1`).
`V`	the estimations of Variances of the Conditional Expectations (VCE) with respect to each factor.
`S`	the estimations of the Sobol' indices.

Warning messages

"The value entered for `N` is not the square of a prime number. It has been replaced by: " when `order`$= 2$, the number of levels of the orthogonal array must be a prime number. If `N` is not a square of a prime number, then this warning message indicates that it was replaced depending on the value of `tail`. If `tail=TRUE` (resp. `tail=FALSE`) the new value of `N` is equal to the square of the prime number preceding (resp. following) the square root of `N`.

"The value entered for `N` is not satisfying the constraint $N \geq (d-1)^2$. It has been replaced by: " when `order`$= 2$, the following constraint must be satisfied $N \geq (d-1)^2$ where d is the number of factors. This warning message indicates that `N` was replaced by the square of the prime number following (or equals to) $d-1$.

Author(s)

Laurent Gilquin

References

A.S. Hedayat, N.J.A. Sloane and J. Stufken, 1999, *Orthogonal Arrays: Theory and Applications*, Springer Series in Statistics.

F. Gamboa, A. Janon, T. Klein and A. Lagnoux, 2014, *Sensitivity indices for multivariate outputs*, Electronic Journal of Statistics, 8:575-603.

J.Y. Tissot and C. Prieur, 2015, *Estimating Sobol's indices combining Monte Carlo integration and Latin hypercube sampling*, J. Statist. Comput. Simulation, 85:1358-1381.

See Also

sobolmara, sobolroauc, sobolMultOut

Examples

```
library(boot)
library(numbers)

####################
# Test case: the non-monotonic Sobol g-function

# The method of sobol requires 2 samples
# (there are 8 factors, all following the uniform distribution on [0,1])

# first-order sensitivity indices
x <- sobolroalhs(model = sobol.fun, factors = 8, N = 1000, order = 1, nboot=100)
print(x)
plot(x)

# closed second-order sensitivity indices
x <- sobolroalhs(model = sobol.fun, factors = 8, N = 1000, order = 2, nboot=100)
print(x)
plot(x)

####################
# Test case: dealing with non-uniform distributions

x <- sobolroalhs(model = NULL, factors = 3, N = 1000, order =1, nboot=0)

# X1 follows a log-normal distribution:
x$X[,1] <- qlnorm(x$X[,1])

# X2 follows a standard normal distribution:
x$X[,2] <- qnorm(x$X[,2])

# X3 follows a gamma distribution:
x$X[,3] <- qgamma(x$X[,3],shape=0.5)

# toy example
toy <- function(x){rowSums(x)}
y <- toy(x$X)
tell(x, y)
print(x)
plot(x)

####################
# Test case : multidimensional outputs

toy <- function(x){cbind(x[,1]+x[,2]+x[,1]*x[,2],2*x[,1]+3*x[,1]*x[,2]+x[,2])}
x <- sobolroalhs(model = toy, factors = 3, N = 1000, p=2, order =1, nboot=100)
print(x)
plot(x)
```

`sobolroauc` *Sobol' Indices estimation under inequality constraints*

Description

`sobolroauc` deals with the estimation of Sobol' sensitivity indices when there exists one or multiple sets of constrained factors. Constraints within a set are expressed as inequality constraints (simplex constraint). This function generalizes the procedure of Tissot and Prieur (2015) to estimate either all first-order indices or all closed second-order indices at a total cost of $2 \times N$ model evaluations. For closed second-order indices $N = q^2$ where $q \geq d - 1$ is a prime number denoting the number of levels of the orthogonal array, and where d indicates the number of independent factors or sets of factors.

Usage

```
sobolroauc(model = NULL, factors, constraints = NULL, N, p = 1, order,
           tail = TRUE, conf = 0.95, nboot = 0, ...)
## S3 method for class 'sobolroauc'
tell(x, y = NULL, ...)
## S3 method for class 'sobolroauc'
print(x, ...)
## S3 method for class 'sobolroauc'
plot(x, ylim = c(0,1), ...)
```

Arguments

`model`	a function, or a model with a `predict` method, defining the model to analyze.
`factors`	an integer giving the number of factors, or a vector of character strings giving their names.
`constraints`	a list giving the sets of constrained factors (see "Details").
`N`	an integer giving the size of each replicated design (for a total of $2 \times N$ model evaluations).
`p`	an integer giving the number of model outputs.
`order`	an integer giving the order of the indices (1 or 2).
`tail`	a boolean specifying the method used to choose the number of levels of the orthogonal array (see "Warning messages").
`conf`	the confidence level for confidence intervals.
`nboot`	the number of bootstrap replicates.
`x`	a list of class `"sobolroauc"` storing the state of the sensitivity study (parameters, data, estimates).
`y`	a vector of model responses.
`ylim`	y-coordinate plotting limits.
`...`	any other arguments for `model` which are passed unchanged each time it is called.

Details

constraints list the sets of factors depending on each other through inequality constraints (see "Examples"). A same factor is not allowed to appear in multiple sets. Factors not appearing in constraints are assumed to be independent and follow each a uniform distribution on [0,1]. One Sobol' index is estimated for each independent factor or set of factors.

Missing values (i.e NA values) in the model responses are automatically handled by the function.

This function also supports multidimensional outputs (matrices in y or as output of model). In this case, aggregated Sobol' indices are returned (see sobolMultOut).

Value

sobolroauc returns a list of class "sobolroauc", containing all the input arguments detailed before, plus the following components:

call	the matched call.
X	a data.frame containing the design of experiments (concatenation of two replicated designs).
y	the responses used.
OA	the orthogonal array constructed (NULL if order=1).
V	the estimations of Variances of the Conditional Expectations (VCE) with respect to each factor.
S	the estimations of the Sobol' indices.

Warning messages

"The value entered for N is not the square of a prime number. It has been replaced by: " when order$= 2$, the number of levels of the orthogonal array must be a prime number. If N is not a square of a prime number, then this warning message indicates that it was replaced depending on the value of tail. If tail=TRUE (resp. tail=FALSE) the new value of N is equal to the square of the prime number preceding (resp. following) the square root of N.

"The value entered for N is not satisfying the constraint $N \geq (d-1)^2$. It has been replaced by: " when order$= 2$, the following constraint must be satisfied $N \geq (d-1)^2$ where d is the number of independent factors or sets of factors. This warning message indicates that N was replaced by the square of the prime number following (or equals to) $d-1$.

Author(s)

Laurent Gilquin

References

L. Devroye, 1986, Non-Uniform Random Variate Generation. Springer-Verlag.

J. Jacques, C. Lavergne and N. Devictor, 2006, Sensitivity Analysis in presence of model uncertainty and correlated inputs. Reliability Engineering & System Safety, 91:1126-1134.

L. Gilquin, C. Prieur and E. Arnaud, 2015, *Replication procedure for grouped Sobol' indices estimation in dependent uncertainty spaces*, Information and Inference, 4:354-379.

J.Y. Tissot and C. Prieur, 2015, *Estimating Sobol's indices combining Monte Carlo integration and Latin hypercube sampling*, J. Statist. Comput. Simulation, 85:1358-1381.

See Also

sobolroalhs, sobolmara

Examples

```
library(boot)
library(numbers)

# Test case: the non-monotonic Sobol g-function
# (there are 8 factors, all following the uniform distribution on [0,1])

# Suppose we have the inequality constraints: X1 <= X3 and X4 <= X6.

# first-order sensitivity indices
x <- sobolroauc(model = sobol.fun, factors = 8, constraints = list(c(1,3),c(4,6)),
                N = 1000, order = 1, nboot=100)
print(x)
plot(x)

# closed second-order sensitivity indices
x <- sobolroauc(model = sobol.fun, factors = 8, constraints = list(c(1,3),c(4,6)),
                N = 1000, order = 2, nboot=100)
print(x)
plot(x)
```

---

sobolSalt *Monte Carlo Estimation of Sobol' Indices based on Saltelli schemes*

Description

`sobolSalt` implements the Monte Carlo estimation of the Sobol' indices for either both first-order and total effect indices at the same time (alltogether $2p$ indices) at a total cost of $n \times (p+2)$ model evaluations; or first-order, second-order and total indices at the same time (alltogether $2p+p \times (p-1)/2$ indices) at a total cost of $n \times (2 \times p + 2)$ model evaluations.

Usage

```
sobolSalt(model = NULL, X1, X2, scheme="A", nboot = 0, conf = 0.95, ...)
## S3 method for class 'sobolSalt'
tell(x, y = NULL, ...)
## S3 method for class 'sobolSalt'
print(x, ...)
## S3 method for class 'sobolSalt'
plot(x, ylim = c(0, 1), choice, ...)
```

Arguments

model	a function, or a model with a predict method, defining the model to analyze.
X1	the first random sample (containing n points).
X2	the second random sample (containing n points).
scheme	a letter "A" or "B" indicating which scheme to use (see "Details")
nboot	the number of bootstrap replicates.
conf	the confidence level for bootstrap confidence intervals.
x	a list of class "sobol" storing the state of the sensitivity study (parameters, data, estimates).
y	a vector of model responses.
ylim	y-coordinate plotting limits.
choice	an integer specifying which indices to plot: 1 for first-order and total effect indices, 2 for second-order indices.
...	any other arguments for model which are passed unchanged each time it is called

Details

The estimators used are the one implemented in "sobolEff".

scheme specifies which Saltelli's scheme is to be used: "A" to estimate both first-order and total effect indices, "B" to estimate first-order, second-order and total effect indices.

Value

sobolSalt returns a list of class "sobolSalt", containing all the input arguments detailed before, plus the following components:

call	the matched call.
X	a data.frame containing the design of experiments.
y	the response used.
V	the model variance.
S	the estimations of the Sobol' first-order indices.
S2	the estimations of the Sobol' second-order indices (only for scheme "B").
T	the estimations of the Sobol' total sensitivity indices.

Author(s)

Laurent Gilquin

References

A. Janon, T. Klein, A. Lagnoux, M. Nodet, C. Prieur (2014), *Asymptotic normality and efficiency of two Sobol index estimators*, ESAIM: Probability and Statistics, 18:342-364.

A. Saltelli, 2002, *Making best use of model evaluations to compute sensitivity indices*, Computer Physics Communication, 145:580-297.

See Also

sobol, sobol2007, soboljansen, sobolmartinez, sobolEff

Examples

```
# Test case : the non-monotonic Sobol g-function

# The method of sobol requires 2 samples
# There are 8 factors, all following the uniform distribution
# on [0,1]

library(boot)
n <- 1000
X1 <- data.frame(matrix(runif(8 * n), nrow = n))
X2 <- data.frame(matrix(runif(8 * n), nrow = n))

# sensitivity analysis

x <- sobolSalt(model = sobol.fun, X1, X2, scheme="A", nboot = 100)
print(x)
plot(x, choice=1)
```

---

sobolSmthSpl *Estimation of Sobol' First Order Indices with B-spline Smoothing*

Description

Determines the Si coefficient for singular parameters through B-spline smoothing with roughness penalty.

Usage

```
sobolSmthSpl(Y, X)
```

Arguments

Y	vector of model responses.
X	matrix having as rows the input vectors corresponding to the responses in Y.

Value

sobolSmthSpl returns a list of class "sobolSmthSpl" containing the following components:

call:	the matched call.
X:	the provided input matrix.
Y:	the provided matrix of model responses.

S: a matrix having the following columns: Si (the estimated first order Sobol' indices), Si.e (the standard errors for the estimated first order Sobol' indices) and q0.05 (the 0.05 quantiles assuming for the Si indices Normal distributions centred on the Si estimates and with standard deviations the calculated standard errors)

Author(s)

Filippo Monari

References

Saltelli, A; Ratto, M; Andres, T; Campolongo, F; Cariboni, J; Gatelli, D; Saisana, M & Tarantola, S. *Global Sensitivity Analysis: The Primer Wiley-Interscience*, 2008

M Ratto and A. Pagano, 2010, *Using recursive algorithms for the efficient identification of smoothing spline ANOVA models*, Advances in Statistical Analysis, 94, 367–388.

See Also

sobol2002, sobol2007, soboljansen, sobolmartinez, sobolEff, sobolmara, sobolroalhs, fast99, sobol0

Examples

```
X = matrix(runif(10000), ncol = 10)
Y = sobol.fun(X)
sa = sobolSmthSpl(Y, X)
plot(sa)
```

---

sobolTIIlo *Liu and Owen Estimation of Total Interaction Indices*

Description

sobolTIIlo implements the asymptotically efficient formula of Liu and Owen (2006) for the estimation of total interaction indices as described e.g. in Section 3.4 of Fruth et al. (2014). Total interaction indices (TII) are superset indices of pairs of variables, thus give the total influence of each second-order interaction. The total cost of the method is $\binom{(1+N+}{(N,2))\times n}$ where N is the number of indices to estimate. Asymptotic confidence intervals are provided. Via plotFG (which uses functions of the package igraph), the TIIs can be visualized in a so-called FANOVA graph as described in section 2.2 of Muehlenstaedt et al. (2012).

Usage

```
sobolTIIlo(model = NULL, X1, X2, conf = 0.95, ...)
## S3 method for class 'sobolTIIlo'
tell(x, y = NULL, ...)
## S3 method for class 'sobolTIIlo'
print(x, ...)
```

```
## S3 method for class 'sobolTIIlo'
plot(x, ylim = NULL, ...)
## S3 method for class 'sobolTIIlo'
plotFG(x)
```

Arguments

model	a function, or a model with a predict method, defining the model to analyze.
X1	the first random sample.
X2	the second random sample.
conf	the confidence level for asymptotic confidence intervals, defaults to 0.95.
x	a list of class "sobolTIIlo" storing the state of the sensitivity study (parameters, data, estimates).
y	a vector of model responses.
...	any other arguments for model which are passed unchanged each time it is called.
ylim	optional, the y limits of the plot.

Value

sobolTIIlo returns a list of class "sobolTIIlo", containing all the input arguments detailed before, plus the following components:

call	the matched call.
X	a data.frame containing the design of experiments.
y	a vector of model responses.
V	the estimation of the overall variance.
tii.unscaled	the unscaled estimations of the TIIs.
tii.scaled	the scaled estimations of the TIIs together with asymptotic confidence intervals.

Author(s)

Jana Fruth

References

R. Liu, A. B. Owen, 2006, *Estimating mean dimensionality of analysis of variance decompositions*, JASA, 101 (474), 712–721.

J. Fruth, O. Roustant, S. Kuhnt, 2014, *Total interaction index: A variance-based sensitivity index for second-order interaction screening*, J. Stat. Plan. Inference, 147, 212–223.

T. Muehlenstaedt, O. Roustant, L. Carraro, S. Kuhnt, 2012, *Data-driven Kriging models based on FANOVA-decomposition*, Stat. Comput., 22 (3), 723–738.

See Also

sobolTIIpf

Examples

```
# Test case : the Ishigami function

# The method requires 2 samples
n <- 1000
X1 <- data.frame(matrix(runif(3 * n, -pi, pi), nrow = n))
X2 <- data.frame(matrix(runif(3 * n, -pi, pi), nrow = n))

# sensitivity analysis (the true values of the scaled TIIs are 0, 0.244, 0)
x <- sobolTIIlo(model = ishigami.fun, X1 = X1, X2 = X2)
print(x)

# plot of tiis and FANOVA graph
plot(x)

## Not run:
library(igraph)
plotFG(x)

## End(Not run)
```

---

sobolTIIpf *Pick-freeze Estimation of Total Interaction Indices*

Description

sobolTIIpf implements the pick-freeze estimation of total interaction indices as described in Section 3.3 of Fruth et al. (2014). Total interaction indices (TII) are superset indices of pairs of variables, thus give the total influence of each second-order interaction. The pick-freeze estimation enables the strategy to reuse evaluations of Saltelli (2002). The total costs are $(1+N) \times n$ where N is the number of indices to estimate. Via plotFG, the TIIs can be visualized in a so-called FANOVA graph as described in section 2.2 of Muehlenstaedt et al. (2012).

Usage

```
sobolTIIpf(model = NULL, X1, X2, ...)
## S3 method for class 'sobolTIIpf'
tell(x, y = NULL, ...)
## S3 method for class 'sobolTIIpf'
print(x, ...)
## S3 method for class 'sobolTIIpf'
plot(x, ylim = NULL, ...)
## S3 method for class 'sobolTIIpf'
plotFG(x)
```

Arguments

model	a function, or a model with a predict method, defining the model to analyze.
X1	the first random sample.
X2	the second random sample.
x	a list of class "sobolTIIpf" storing the state of the sensitivity study (parameters, data, estimates).
y	a vector of model responses.
...	any other arguments for model which are passed unchanged each time it is called.
ylim	optional, the y limits of the plot.

Value

sobolTIIpf returns a list of class "sobolTIIpf", containing all the input arguments detailed before, plus the following components:

call	the matched call.
X	a data.frame containing the design of experiments.
y	a vector of model responses.
V	the estimation of the overall variance.
tii.unscaled	the unscaled estimations of the TIIs together.
tii.scaled	the scaled estimations of the TIIs.

Author(s)

Jana Fruth

References

J. Fruth, O. Roustant, S. Kuhnt, 2014, *Total interaction index: A variance-based sensitivity index for second-order interaction screening*, J. Stat. Plan. Inference, 147, 212–223.

A. Saltelli, 2002, *Making best use of model evaluations to compute sensitivity indices*, Comput. Phys. Commun., 145, 580-297.

T. Muehlenstaedt, O. Roustant, L. Carraro, S. Kuhnt, 2012, *Data-driven Kriging models based on FANOVA-decomposition*, Stat. Comput., 22 (3), 723–738.

See Also

sobolTIIlo

Examples

```
# Test case : the Ishigami function

# The method requires 2 samples
n <- 1000
X1 <- data.frame(matrix(runif(3 * n, -pi, pi), nrow = n))
X2 <- data.frame(matrix(runif(3 * n, -pi, pi), nrow = n))

# sensitivity analysis (the true values are 0, 0.244, 0)
x <- sobolTIIpf(model = ishigami.fun, X1 = X1, X2 = X2)
print(x)

# plot of tiis and FANOVA graph
plot(x)

## Not run:
library(igraph)
plotFG(x)

## End(Not run)
```

`soboltouati` *Monte Carlo Estimation of Sobol' Indices (formulas of Martinez (2011) and Touati (2016))*

Description

`soboltouati` implements the Monte Carlo estimation of the Sobol' indices for both first-order and total indices using correlation coefficients-based formulas, at a total cost of $(p + 2) \times n$ model evaluations. These are called the Martinez estimators. It also computes their confidence intervals based on asymptotic properties of empirical correlation coefficients.

Usage

```
soboltouati(model = NULL, X1, X2, conf = 0.95, ...)
## S3 method for class 'soboltouati'
tell(x, y = NULL, return.var = NULL, ...)
## S3 method for class 'soboltouati'
print(x, ...)
## S3 method for class 'soboltouati'
plot(x, ylim = c(0, 1), ...)
```

Arguments

`model`	a function, or a model with a `predict` method, defining the model to analyze.
`X1`	the first random sample.
`X2`	the second random sample.

conf	the confidence level for confidence intervals, or zero to avoid their computation if they are not needed.
x	a list of class "sobol" storing the state of the sensitivity study (parameters, data, estimates).
y	a vector of model responses.
return.var	a vector of character strings giving further internal variables names to store in the output object x.
ylim	y-coordinate plotting limits.
...	any other arguments for model which are passed unchanged each time it is called

Details

This estimator supports missing values (NA or NaN) which can occur during the simulation of the model on the design of experiments (due to code failure) even if Sobol' indices are no more rigorous variance-based sensitivity indices if missing values are present. In this case, a warning is displayed.

Value

soboltouati returns a list of class "soboltouati", containing all the input arguments detailed before, plus the following components:

call	the matched call.
X	a data.frame containing the design of experiments.
y	the response used
V	the estimations of normalized variances of the Conditional Expectations (VCE) with respect to each factor and also with respect to the complementary set of each factor ("all but X_i").
S	the estimations of the Sobol' first-order indices.
T	the estimations of the Sobol' total sensitivity indices.

Author(s)

Taieb Touati, Khalid Boumhaout

References

J-M. Martinez, 2011, *Analyse de sensibilite globale par decomposition de la variance*, Presentation in the meeting of GdR Ondes and GdR MASCOT-NUM, January, 13th, 2011, Institut Henri Poincare, Paris, France.

T. Touati, 2016, Confidence intervals for Sobol' indices. Proceedings of the SAMO 2016 Conference, Reunion Island, France, December 2016.

T. Touati, 2016, *Proprietes asymptotiques du coefficient de correlation empirique*, draft.

See Also

sobol, sobol2002, sobolSalt, sobol2007, soboljansen, sobolEff, sobolmara, sobolmartinez

Examples

```
# Test case : the non-monotonic Sobol g-function

# The method of sobol requires 2 samples
# There are 8 factors, all following the uniform distribution
# on [0,1]

library(boot)
n <- 1000
X1 <- data.frame(matrix(runif(8 * n), nrow = n))
X2 <- data.frame(matrix(runif(8 * n), nrow = n))

# sensitivity analysis

x <- soboltouati(model = sobol.fun, X1, X2)
print(x)
plot(x)
```

---

src *Standardized Regression Coefficients*

Description

src computes the Standardized Regression Coefficients (SRC), or the Standardized Rank Regression Coefficients (SRRC), which are sensitivity indices based on linear or monotonic assumptions in the case of independent factors.

Usage

```
src(X, y, rank = FALSE, nboot = 0, conf = 0.95)
## S3 method for class 'src'
print(x, ...)
## S3 method for class 'src'
plot(x, ylim = c(-1,1), ...)
```

Arguments

X	a data frame (or object coercible by as.data.frame) containing the design of experiments (model input variables).
y	a vector containing the responses corresponding to the design of experiments (model output variables).
rank	logical. If TRUE, the analysis is done on the ranks.
nboot	the number of bootstrap replicates.
conf	the confidence level of the bootstrap confidence intervals.
x	the object returned by src.
ylim	the y-coordinate limits of the plot.
...	arguments to be passed to methods, such as graphical parameters (see par).

Value

src returns a list of class "src", containing the following components:

call	the matched call.
SRC	a data frame containing the estimations of the SRC indices, bias and confidence intervals (if rank = FALSE).
SRRC	a data frame containing the estimations of the SRRC indices, bias and confidence intervals (if rank = TRUE).

Author(s)

Gilles Pujol

References

A. Saltelli, K. Chan and E. M. Scott eds, 2000, *Sensitivity Analysis*, Wiley.

See Also

pcc

Examples

```
# a 100-sample with X1 ~ U(0.5, 1.5)
#                   X2 ~ U(1.5, 4.5)
#                   X3 ~ U(4.5, 13.5)

library(boot)
n <- 100
X <- data.frame(X1 = runif(n, 0.5, 1.5),
                X2 = runif(n, 1.5, 4.5),
                X3 = runif(n, 4.5, 13.5))

# linear model : Y = X1 + X2 + X3

y <- with(X, X1 + X2 + X3)

# sensitivity analysis

x <- src(X, y, nboot = 100)
print(x)
plot(x)
```

support *Support index functions: Measuring the effect of input variables over their support*

Description

Function to estimate the first-order and total support index functions (Fruth et al., 2016).

Usage

```
support(model, X, Xnew = NULL, fX = NULL, gradfX = NULL, h = 1e-06, ...)
```

Arguments

model	a function, or a model with a predict method, defining the model to analyze.
X	a random sample.
Xnew	an optional set of points where to visualize the support indices. If missing, X is used.
fX	an optional vector containing the evaluations of model at X. If missing, fX is computed by evaluating model at X.
gradfX	an optional vector containing the evaluations of the gradient of model at X. If missing, gradfX is approximated by finite differences of model at X.
h	a small number for computing finite differences (f(X_i + h) - f(X_i))/h. Default is 1e-6.
...	optional arguments to be passed to model.

Details

The first-order support index of f(X) relative to X_i is the squared conditional expectation of its partial derivative with respect to X_i.

The total support index of f(X) relative to X_i is the conditional expectation of its squared partial derivative with respect to X_i.

These two functions measure the local influence of X_i, in the global space of the other input variables. Up to square transformations, support indices can be viewed as regression curves of partial derivatives df(X)/dX_i with respect to X_i. Estimation is performed by smoothing from the diagonal scatterplots (X_i, df/dX_i) with the function smooth.spline{stats} with the default options.

For the sake of comparison, support index functions may be normalized. The proposed normalization is the sum of the DGSM, equal to the sum of the overall means of total support functions. Normalized support index functions can be plotted with the S3 method plot, as well as the underlying diagonal scatterplots of derivatives (S3 method scatterplot).

Value

main	a matrix whose columns contain the first-order support index functions, estimated at Xnew.
total	a matrix whose columns contain the total support index functions, estimated at Xnew.
DGSM	a vector containing an estimation of DGSM.
X	...
Xnew	...
fX	...
gradfX	... see 'arguments' section.

Author(s)

O. Roustant

References

J. Fruth, O. Roustant, S. Kuhnt, *Support indices: Measuring the effects of input variables over their support*, 2016, https://hal.archives-ouvertes.fr/hal-01113555.

See Also

S3 methods plot and scatterplot: plot.support

Examples

```
# -----------------
# ishigami function
# -----------------
n <- 5000
n.points <- 1000
d <- 3

set.seed(0)
X <- matrix(runif(d*n, min = -pi, max = pi), n, d)
Xnew <- matrix(seq(from = -pi, to = pi, length=n.points), n.points, d)

b <- support(model = ishigami.fun, X, Xnew)

# plot method (x-axis in probability scale), of the normalized support index functions
plot(b, col = c("lightskyblue4", "lightskyblue1", "black"),
     xprob = TRUE, p = 'punif', p.arg = list(min = -pi, max = pi), ylim = c(0, 2))

# below : diagonal scatterplots of the gradient,
# on which are based the estimation by smoothing
scatterplot(b, xprob = TRUE)

# now with normal margins
```

```
# -----------------------
X <- matrix(rnorm(d*n), n, d)
Xnew <- matrix(rnorm(d*n.points), n.points, d)
b <- support(model = ishigami.fun, X, Xnew)

plot(b, col = c("lightskyblue4", "lightskyblue1", "black"), xprob = FALSE)
scatterplot(b, xprob = FALSE, type = "histogram", bins = 10, cex = 1, cex.lab = 1.5)
```

`template.replace` *Replace Values in a Template Text*

Description

`template.replace` replaces keys within special markups with values in a so-called template file. Pieces of R code can be put into the markups of the template file, and are evaluated during the replacement.

Usage

```
template.replace(text, replacement, eval = FALSE,
                 key.pattern = NULL, code.pattern = NULL)
```

Arguments

`text`	vector of character strings, the template text.
`replacement`	the list values to replace in `text`.
`eval`	boolean, `TRUE` if the code within `code.pattern` has to be evaluated, `FALSE` otherwise.
`key.pattern`	custom pattern for key replacement (see below)
`code.pattern`	custom pattern for code replacement (see below)

Details

In most cases, a computational code reads its inputs from a text file. A template file is like an input file, but where some missing values, identified with generic keys, will be replaced by specific values.

By default, the keys are enclosed into markups of the form `$(KEY)`.

Code to be interpreted with R can be put in the template text. Pieces of code must be enclosed into markups of the form `@{CODE}`. This is useful for example for formating the key values (see example). For interpreting the code, set `eval = TRUE`.

Users can define custom patterns. These patterns must be perl-compatible regular expressions (see `regexpr`. The default ones are:

```
key.pattern = "\$\\(KEY\\)"
code.pattern = "@\\{CODE\\}"
```

Note that special characters have to be escaped both (one for perl, one for R).

Author(s)

Gilles Pujol

Examples

```
txt <- c("Hello $(name)!", "$(a) + $(b) = @{$(a)+$(b)}",
         "pi = @{format(pi,digits=5)}")
replacement <- list(name = "world", a = 1, b = 2)
# 1. without code evaluation:
txt.rpl1 <- template.replace(txt, replacement)
print(txt.rpl1)
# 2. with code evalutation:
txt.rpl2 <- template.replace(txt, replacement, eval = TRUE)
print(txt.rpl2)
```

---

testmodels *Test Models for Sensitivity Analysis*

Description

These functions are standard testcase for sensitivity analysis benchmarks. For a scalar output (see Saltelli et al. 2000, section 2.9):

- the g-function of Sobol' with 8 inputs, X ~ U[0,1];
- the function of Ishigami with 3 inputs, X ~ U[-pi,pi];
- the function of Morris with 20 inputs, X ~ U[0,1].

For functional output cases:

- the Arctangent temporal function with 2 inputs, X ~ U[-7,7] (Auder, 2011). The functional support is on [0,2pi];
- the Cambell1D function with 4 nputs, X ~U[-1,5] (Campbell et al. 2006). The functional support is on [-90,90].

Usage

```
sobol.fun(X)
ishigami.fun(X)
morris.fun(X)
atantemp.fun(X, q = 100)
campbell1D.fun(X, theta = -90:90)
```

Arguments

X	a matrix (or data.frame) containing the input sample.
q	for the atantemp() function: the number of discretization steps of the functional output
theta	for the campbell1D() function: the discretization steps (angles in degrees)

Value

A vector of function responses.

Author(s)

Gilles Pujol

References

A. Saltelli, K. Chan and E. M. Scott eds, 2000, *Sensitivity Analysis*, Wiley.

Examples

```
## Not run:

# Examples for the functional toy fonctions

# atantemp function

y0 <- atantemp.fun(matrix(c(-7,0,7,-7,0,7),ncol=2))
plot(y0[1,],type="l")
apply(y0,1,lines)

n <- 100
X <- matrix(c(runif(2*n,-7,7)),ncol=2)
y <- atantemp.fun(X)
x11()
plot(y0[2,],ylim=c(-2,2),type="l")
apply(y,1,lines)

# campbell1D function

N1=100         # nombre de simulations pour courbes 1D
min=-1 ; max=5
nominal=(max+min)/2

X1 = NULL ; y1 = NULL
Xnom=matrix(nominal,nr=1,nc=4)
ynom=campbell1D.fun(Xnom,theta=-90:90)
x11()
plot(ynom,ylim=c(8,30),type="l",col="red")
for (i in 1:N1){
  X=matrix(runif(4,min=min,max=max),nr=1,nc=4)
  rbind(X1,X)
  y=campbell1D.fun(X,theta=-90:90)
  rbind(y1,y)
  lines(y)
}


## End(Not run)
```

Index