Package ‘sensitivity’

August 9, 2022

Version 1.27.1
Title Global Sensitivity Analysis of Model Outputs
Author Bertrand Iooss, Sebastien Da Veiga, Alexandre Janon and Gilles Pujol, with contributions from Baptiste Broto, Khalid Boumhaout, Thibault Delage, Reda El Amri, Jana Fruth, Laurent Gilquin, Joseph Guillaume, Mar- got Herin, Marouane Il Idrissi, Loic Le Gratiet, Paul Lemaitre, Amandine Marrel, Anouar Mey- naoui, Barry L. Nelson, Fil- ippo Monari, Roelof Oomen, Oldrich Rakovec, Bernardo Ramos, Olivier Roustant, Eun- hye Song, Jeremy Staum, Roman Sueur, Taieb Touati, Vanessa Verges, Frank Weber
Maintainer Bertrand Iooss <biooss@yahoo.fr>
Depends R (>= 3.0.0)
Imports boot, numbers, methods, ggplot2, Rcpp, foreach
LinkingTo Rcpp, RcppArmadillo
Suggests condMVNorm, DiceDesign, DiceKriging, doParallel, evd, ggExtra, grid, gplots, igraph, ks, lattice, MASS, mc2d, mvtnorm, parallel, plotrix, pracma, randtoolbox, RANN, reshape2, rgl, triangle, TSP, viridisLite, whitening
Description A collection of functions for sensitivity analysis of model outputs (factor screening, global sensitivity analysis and robustness analysis), as well as for interpretability of machine learning models. Most of the functions have to be applied on scalar output, but several functions support multi-dimensional outputs.
License GPL-2
NeedsCompilation yes
Repository CRAN
Date/Publication 2022-08-09 08:30:02 UTC

R topics documented:

sensitivity-package .......................................................... 3
addelman_const .............................................................. 7
decoupling ................................................................. 8
delsa ................................................................. 9
R topics documented:

discrepancyCriteria_cplus .................................................. 11
fast99 ................................................................. 13
lmg ................................................................. 15
maximin_cplus .......................................................... 18
morris ................................................................. 19
morrisMultOut ......................................................... 24
parameterSets ......................................................... 26
pcc ................................................................. 27
PLI ................................................................. 29
PLIquantile ......................................................... 33
PLIquantile_multivar .................................................. 37
PLIsuperquantile .................................................... 41
PLIsuperquantile_multivar ......................................... 45
plot.support ......................................................... 49
pme_knn ............................................................ 50
pmvd ............................................................... 55
PoincareChaosSqCoef ................................................ 59
PoincareConstant ..................................................... 63
PoincareOptimal ...................................................... 66
qosa ................................................................. 69
sb ................................................................. 72
sensiFdiv ............................................................ 74
sensiHSIC ............................................................ 76
shapleyBlockEstimation ............................................... 82
shapleyLinearGaussian ............................................... 86
shapleyPermEx ........................................................ 87
shapleyPermRand ...................................................... 91
shapleysobol_knn ..................................................... 94
shapleySubsetMc ..................................................... 101
sobol ............................................................... 103
sobol2002 ............................................................ 105
sobol2007 ............................................................ 107
sobolEff ............................................................. 110
sobolGP ............................................................. 112
soboljansen ........................................................... 116
sobolimara ........................................................... 119
sobolmartinez ........................................................ 121
sobolMultOut ......................................................... 124
sobolowen ........................................................... 126
sobolrank ............................................................ 128
sobolrec ............................................................ 130
sobolrep ............................................................ 133
sobolralhs ........................................................... 136
sobolroalhs ........................................................... 139
sobolSalt ............................................................. 142
sobolshap_knn ........................................................ 144
sobolSmthSpl ........................................................ 148
sobolTIIIlo ........................................................... 149
Description

Methods and functions for global sensitivity analysis of model outputs and machine learning interpretability

Details

The sensitivity package implements some global sensitivity analysis methods:

- Linear regression importance measures in regression or classification contexts (Iooss et al., 2022):
  - SRC and SRRC (`src`),
  - PCC, SPCC, PRCC and SPRCC (`pcc`),
  - LMG and LMG on ranks (`lmg`)
  - PMVD and PMVD on ranks (`pmvd`);
- 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`),
  - Squared coefficients computation in generalized chaos via Poincare differential operators (Roustant et al., 2019) (`PoincareChaosSqCoef`),
  - Distributed Evaluation of Local Sensitivity Analysis (DELSA) (Rakovec et al., 2014) (`delsa`);
- Variance-based sensitivity indices (Sobol’ indices) for independent inputs:
  - 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 permutations on a single matrix (sobolmara),
* Mighty estimator of first-order sensitivity indices based on rank statistics (correlation coefficient of Chatterjee, 2019; Gamboa et al., 2020) (sobolrank),
* 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).

– Replication-based procedures:
  * Estimation of the Sobol’ first order and closed second order indices using replicated orthogonal array-based Latin hypecube sample (Tissot and Prieur, 2015) (sobolroalhs),
  * Recursive estimation of the Sobol’ first order and closed second order indices using replicated orthogonal array-based Latin hypecube sample (Gilquin et al., 2016) (sobolrec),
  * Estimation of the Sobol’ first order, second order and total indices using the generalized method with replicated orthogonal array-based Latin hypecube sample (Tissot and Prieur, 2015) (sobolrep),
  * 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 Saltelli’s so-called "extended-FAST" method (Saltelli et al., 1999) (fast99),
  * 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 valid for dependent inputs:
  * Exact computation of Shapley effects in the linear Gaussian framework (Broto et al., 2019) (shapleyLinearGaussian),
  * Computation of Shapley effects in the Gaussian linear framework with an unknown block-diagonal covariance matrix (Broto et al., 2020) (shapleyBlockEstimation),
  * Estimation of Shapley effects by examining all permutations of inputs (Song et al., 2016) (shapleyPermEx),
– Estimation of Shapley effects by randomly sampling permutations of inputs (Song et al., 2016) (shapleyPermRand),
– Estimation of Shapley effects from data using nearest neighbors method (Broto et al., 2018) (shapleySubsetMc),
– Estimation of Shapley effects and all Sobol indices from data using nearest neighbors (Broto et al., 2018) using a fast approximate algorithm, and ranking (Gamboa et al., 2020) (shapleysobol_knn) and (sobolshap_knn),
– Estimation of Shapley effects from data using nearest neighbors method (Broto et al., 2018) with an optimized/parallelized computations and bootstrap confidence intervals estimations (shapleysobol_knn),
– Estimation of Proportional Marginal Effects (PME) (Il Idrissi et al., 2021) (pme_knn);

• 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) (Da Veiga, 2015; Meynaoui et al., 2019);
• First-order quantile-oriented sensitivity indices as defined in Fort et al. (2016) via a kernel-based estimator related (Maume-Deschamps and Niang, 2018) (qosa);
• Target Sensitivity Analysis via Hilbert-Schmidt Independence Criterion (sensiHSIC) (Spagnol et al., 2019);
• Robustness analysis by the Perturbed-Law based Indices (PLI) of Lemaitre et al. (2015), (PLIquantile) of Sueur et al. (2017), (PLIsuperquantile) of Iooss et al. (2021), and extension as (PLIquantile_multivar) and (PLIsuperquantile_multivar);
• Extensions to multidimensional outputs for:
  – Sobol’ indices (sobolMultOut): Aggregated Sobol’ indices (Lamboni et al., 2011; Gamboa et al., 2014) and functional (1D) Sobol’ indices,
  – Shapley effects and Sobol’ indices (shapleysobol_knn) and (sobolshap_knn): Functional (1D) indices,
  – HSIC indices (sensiHSIC) (Da Veiga, 2015): Aggregated HSIC, potentially via a PCA step (Da Veiga, 2015),
  – Morris method (morrisMultOut).

Moreover, some utilities are provided: standard test-cases (testmodels), weight transformation function of the output sample (weightTSA) to perform Target Sensitivity Analysis, normal and Gumbel truncated distributions (truncateddistrib), squared integral estimate (squaredIntEstim), Addelman and Kempthorne construction of orthogonal arrays of strength two (addelman_const), discrepancy criteria (discrepancyCriteria_cplus), maximin criteria (maximin_cplus) 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 function 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 is 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)

Bertrand Iooss, Sebastien Da Veiga, Alexandre Janon and Gilles Pujol with contributions from Paul Lemaitre for PLI, Thibault Delage and Roman Sueur for PLIquantile, Vanessa Verges for PLIquantile, PLIsuperquantile, PLIquantile_multivar and PLIsuperquantile_multivar, Laurent Gilquin for sobolroalhs, sobolroauc, sobolSalt, sobolrep, sobolrec, as well as addelman_const, discrepancyCriteria_cplus and maximin_cplus, Loic le Gratiet for sobolGP, Khalid Boumhaout, Taieb Touati and Bernardo Ramos for sobolwen and soboltdt, Jana Fruth for PoincareConstant, sobolTIilo and sobolTIipf, Amandine Marrel, Anouar Meynaoui and Reda El Amri for their contributions to sensiHSIC, Joseph Guillaume and Oldrich Rakovec for delsa and parameterSets, Olivier Roustant for PoincareOptimal, PoincareChaosSqCoef, squaredIntEstim and support, Eunhye Song, Barry L. Nelson and Jeremy Staum for shapleyPermEx and shapleyPermRand, Baptiste Broto for shapleySubsetMc, shapleyLinearGaussian and shapleyBlockEstimation, Filippo Monari for (sobolSmthSpl) and (morrisMultOut), Marouane II Idrissi for lmg, pmvd and shapleysobol_knn, associated to Margot Herin for pme_knn, Frank Weber and Roelof Oomen for other contributions.

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

References


B. Iooss, V. Chabridon and V. Thouvenot, Variance-based importance measures for machine learning model interpretability, Congress lambda-mu23, Saclay, France, 10-13 octobre 2022 https://hal.archives-ouvertes.fr/hal-03741384

B. Iooss, R. Kennet and P. Secchi, 2022, Different views of interpretability, In: Interpretability for Industry 4.0: Statistical and Machine Learning Approaches, A. Lepore, B. Palumbo and J-M. Poggi (Eds), Springer.


addelman_const

**Description**

addelman_const implements the Addelman and Kempthorne construction of orthogonal arrays of strength two.

**Usage**

```r
addelman_const(dimension, levels, choice="U")
```

**Arguments**

- `dimension` The number of columns of the orthogonal array.
- `levels` The number of levels of the orthogonal array. Either a prime number or a prime power number.
- `choice` A character from the list ("U", "V", "W", "X") specifying which orthogonal array to construct (see "Details").

**Details**

The method of Addelman and Kempthorne allows to construct up to four orthogonal arrays. choice specify which orthogonal array is to be constructed. Note that the four orthogonal arrays depends on each others through linear equations.

**Value**

A matrix corresponding to the orthogonal array constructed.

**Author(s)**

Laurent Gilquin

**References**


**Examples**

```r
dimension <- 6
levels <- 7
OA <- addelman_const(dimension, levels, choice="U")
```
**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. extract is used as a post-treatment of a sobolshap_knn object.

**Usage**

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

The extract method is useful if in a first step the Shapley effects have been computed and thus sensitivity indices for all possible subsets are available. The resulting sobolshap_knn object can be post-treated by extract to get first-order and total Sobol indices very easily.

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.

extract returns an object, from a sobolshap_knn object, containing first-order and total Sobol indices.
**Author(s)**

Gilles Pujol and Bertrand Iooss

**Examples**

```r
# 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)
```

**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**

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

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

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

```r
## 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.
varoutput Output variance. If "summation" is specified (default value), the output variance is computed by summing the first order effects. If "empirical" is specified, the output variance is estimated from the output sample.
...      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.
deriv      the values of derivatives for each location in X0

Author(s)

Conversion for sensitivity package by Joseph Guillaume, based on original R code by Oldrich Rakovec. Addition of the varoutput argument by Bertrand Iooss (2020).

References


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])

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)
plot(x)

discrepancyCriteria_cplus

Discrepancy measure

Description
Compute discrepancy criteria. This function uses a C++ implementation of the function discrepancyCriteria from package DiceDesign.

Usage
discrepancyCriteria_cplus(design,type='all')

Arguments
design a matrix corresponding to the design of experiments. The discrepancy criteria are computed for a design in the unit cube [0,1]^d. If this condition is not satisfied the design is automatically rescaled.
type type of discrepancies (single value or vector) to be computed:
   'all' all type of discrepancies (default)
   'C2' centered L2-discrepancy
   'L2' L2-discrepancy
   'L2star' L2star-discrepancy
   'M2' modified L2-discrepancy
   'S2' symmetric L2-discrepancy
   'W2' wrap-around L2-discrepancy
**Details**

The discrepancy measures how far a given distribution of points deviates from a perfectly uniform one. Different discrepancies are available. For example, if we denote by $Vol(J)$ the volume of a subset $J$ of $[0; 1]^d$ and $A(X; J)$ the number of points of $X$ falling in $J$, the $L_2$ discrepancy is:

$$D_{L_2}(X) = \left[ \int_{[0,1]^d} \left( \frac{A(X, J_{a,b})}{n} - Vol(J_{a,b}) \right)^2 \, dadb \right]^{1/2}$$

where $a = (a_1; ...; a_d)'$, $b = (b_1; ...; b_d)'$ and $J_{a,b} = [a_1; b_1) \times ... \times [a_d; b_d)$. The other $L_2$-discrepancies are defined according to the same principle with different form from the subset $J$. Among all the possibilities, discrepancyCriteria_cplus implements only the $L_2$ discrepancies because it can be expressed analytically even for high dimension.

Centered $L_2$-discrepancy is computed using the analytical expression done by Hickernell (1998). The user will refer to Fleming and Manteufel (2005) to have more details about the wrap around discrepancy.

**Value**

A list containing the $L_2$-discrepancies of the design.

**Author(s)**

Laurent Gilquin

**References**


**See Also**

The distance criterion provided by maximin_cplus

**Examples**

```r
dimension <- 2
n <- 40
X <- matrix(runif(n*dimension),n,dimension)
discrepancyCriteria_cplus(X)
```
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 (altogether \(2p\) indices, where \(p\) is the number of factors) at a total cost of \(n \times p\) simulations.

Usage

```r
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 $S_{iT}$), 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


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)
lmg

LMG $R^2$ decomposition for linear and logistic regression models

Description

lmg computes the Lindeman, Merenda and Gold (LMG) indices for correlated input relative importance by $R^2$ decomposition for linear and logistic regression models. These indices allocate a share of $R^2$ to each input based on the Shapley attribution system, in the case of dependent or correlated inputs.

Usage

```r
lmg(X, y, logistic = FALSE, rank = FALSE, nboot = 0,
    conf = 0.95, max.iter = 1000, parl = NULL)
```

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

## S3 method for class 'lmg'
plot(x, ylim = c(0,1), ...)

Arguments

- **X**
  - a matrix or data frame containing the observed covariates (i.e., features, input variables...).
- **y**
  - a numeric vector containing the observed outcomes (i.e., dependent variable). If logistic=TRUE, can be a numeric vector of zeros and ones, or a logical vector, or a factor.
- **logistic**
  - logical. If TRUE, the analysis is done via a logistic regression (binomial GLM).
- **rank**
  - logical. If TRUE, the analysis is done on the ranks.
- **nboot**
  - the number of bootstrap replicates for the computation of confidence intervals.
- **conf**
  - the confidence level of the bootstrap confidence intervals.
- **max.iter**
  - if logistic=TRUE, the maximum number of iterative optimization steps allowed for the logistic regression. Default is 1000.
- **parl**
  - number of cores on which to parallelize the computation. If NULL, then no parallelization is done.
- **x**
  - the object returned by lmg.
- **ylim**
  - the y-coordinate limits of the plot.
- **...**
  - arguments to be passed to methods, such as graphical parameters (see par).

Details

The computation is done using the subset procedure, defined in Broto, Bachoc and Depecker (2020), that is computing all the $R^2$ for all possible sub-models first, and then affecting the Shapley weights according to the Lindeman, Merenda and Gold (1980) definition.
For logistic regression (logistic=TRUE), the $R^2$ value is equal to:

$$R^2 = 1 - \frac{\text{model deviance}}{\text{null deviance}}$$

If either a logistic regression model (logistic = TRUE), or any column of $X$ is categorical (i.e., of class factor), then the rank-based indices cannot be computed. In both those cases, rank = FALSE is forced by default (with a warning).

If too many cores for the machine are passed on to the parl argument, the chosen number of cores is defaulted to the available cores minus one.

Value

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

- **call**: the matched call.
- **lm**: the estimations of the LMG indices.
- **R2s**: the estimations of the $R^2$ for all possible sub-models.
- **indices**: list of all subsets corresponding to the structure of R2s.
- **w**: the Shapley weights.
- **conf_int**: a matrix containing the estimations, bias and confidence intervals by bootstrap (if nboot>0).
- **X**: the observed covariates.
- **y**: the observed outcomes.
- **logistic**: logical. TRUE if the analysis has been made by logistic regression.
- **boot**: logical. TRUE if bootstrap estimates have been produced.
- **nboot**: number of bootstrap replicates.
- **rank**: logical. TRUE if a rank analysis has been made.
- **parl**: number of chosen cores for the computation.
- **conf**: level for the confidence intervals by bootstrap.

Author(s)

Marouane Il Idrissi

References


B. Iooss, V. Chabridon and V. Thouvenot, *Variance-based importance measures for machine learning model interpretability*, Congres lambda-mu23, Saclay, France, 10-13 octobre 2022 [https://hal.archives-ouvertes.fr/hal-03741384](https://hal.archives-ouvertes.fr/hal-03741384)


**See Also**

`pcc`, `src`, `shapleyPermEx`, `shapleysobol_knn`, `pmvd`, `pme_knn`

**Examples**

```r
library(parallel)
library(doParallel)
library(foreach)
library(gtools)
library(boot)
library(mvtnorm)

set.seed(1234)
n <- 1000
beta<-c(1,-1,0.5)
sigma<-matrix(c(1,0,0, 
0,1,-0.8, 
0,-0.8,1),
nrow=3, 
ncol=3)

# Gaussian correlated inputs
X <- rmvnorm(n, rep(0,3), sigma)
colnames(X)<-c("X1","X2","X3")

# Linear Model
y <- X%*%beta + rnorm(n,0,2)

# Without Bootstrap confidence intervals
x<-lmg(X, y)
print(x)
plot(x)
```
# With Bootstrap confidence intervals
x<-lmg(X, y, nboot=100, conf=0.95)
print(x)
plot(x)

# Rank-based analysis
x<-lmg(X, y, rank=TRUE, nboot=100, conf=0.95)
print(x)
plot(x)

# Logistic Regression
y<-as.numeric(X%*%beta + rnorm(n)>0)
x<-lmg(X,y, logistic = TRUE)
plot(x)
print(x)

# Parallel computing
#x<-lmg(X,y, logistic = TRUE, parl=2)
#plot(x)
#print(x)

maximin_cplus

---

**maximin_cplus**

*Maximin criterion*

**Description**

Compute the maximin criterion (also called mindist). This function uses a C++ implementation of the function mindist from package `DiceDesign`.

**Usage**

maximin_cplus(design)

**Arguments**

design

a matrix representing the design of experiments in the unit cube $[0,1]^d$. If this last condition is not fulfilled, a transformation into $[0,1]^d$ is applied before the computation of the criteria.

**Details**

The maximin criterion is defined by:

$$\text{maximin} = \min_{x_i \in X} (\gamma_i)$$

where $\gamma_i$ is the minimal distance between the point $x_i$ and the other points $x_k$ of the design. A higher value corresponds to a more regular scaterring of design points.
Value

A real number equal to the value of the maximin criterion for the design.

Author(s)

Laurent Gilquin

References


See Also
discrepancy measures provided by discrepancyCriteria_cplus.

Examples

dimension <- 2
n <- 40
X <- matrix(runif(n*dimension),n,dimension)
maximin_cplus(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

```r
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 \((\mu^*, \sigma)\) 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 $\mu^*$ and $\sigma$ (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 $\mu^*$ and $\sigma$ (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 $(\mu^*, \sigma)$ graph.

plot3d.morris draws the $(\mu, \mu^*, \sigma)$ 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 (\(\mu, \mu^*\) and \(\sigma\)) 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:

```r
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:

```r
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:

```r
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


See Also

morrisMultOut

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)

library(rgl)
plot3d.morris(x)  # (requires the package 'rgl')

# 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")
morrisMultOut

Morris’s Elementary Effects Screening Method for Multidimensional Outputs

Description

morrisMultOut extend the Morris’s elementary effects screening method (Morris 1991) to model with multidimensional outputs.

Usage

```r
call(morrisMultOut(model = NULL, factors, r, design, binf = 0, bsup = 1, scale = TRUE, ...)
## S3 method for class 'morrisMultOut'
tell(x, y = NULL, ...)
```

Arguments

- **model**: NULL or a function returning a matrix having as columns the model outputs.
- **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.
... for morrisMultOut: 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

All the methods available for object of class "morris" are available also for objects of class "morrisMultOut". See the documentation relative to the function "morris" for more details.

Value

morrisMultOut returns a list of class "c(morrisMultOut, 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 a matrix having as columns the model responses.
e a vector of aggregated elementary effects.

Author(s)

Filippo Monari

References


See Also

morris

Examples

mdl <- function (X) t(atantemp.fun(X))
x = morrisMultOut(model = mdl, factors = 4, r = 50, design = list(type = "oat", levels = 5, grid.jump = 3), binf = -1, bsup = 5, scale = FALSE)
print(x)
plot(x)
parameterSets

Generate parameter sets

Description

Generate parameter sets from given ranges, with chosen sampling scheme

Usage

```r
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

```r
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")

library(randtoolbox)
X.sobol<-parameterSets(par.ranges=list(V1=c(1,1000),V2=c(1,4)),
                       samples=100,method="sobol")
plot(X.sobol)
```

---

**pcc**  
*Partial Correlation Coefficients*

**Description**

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

**Usage**

```r
pcc(X, y, rank = FALSE, semi = FALSE, logistic = FALSE, nboot = 0, conf = 0.95)
## S3 method for class 'pcc'
print(x, ...)  
## S3 method for class 'pcc'
plot(x, ylim = c(-1,1), ...)  
## S3 method for class 'pcc'
ggplot(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.
- **semi**: logical. If `TRUE`, semi-PCC are computed.
- **logistic**: logical. If `TRUE`, the analysis is done via a logistic regression (binomial GLM).
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).

Details

Logistic regression model (logistic = TRUE) and rank-based indices (rank = TRUE) are incompat-
ible.

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 and semi = FALSE).
PRCC   a data frame containing the estimations of the PRCC indices, bias and confidence
       intervals (if rank = TRUE and semi = FALSE).
SPCC   a data frame containing the estimations of the PCC indices, bias and confidence
       intervals (if rank = TRUE and semi = TRUE).
SPRCC  a data frame containing the estimations of the PRCC indices, bias and confidence
       intervals (if rank = TRUE and semi = TRUE).

Author(s)

Gilles Pujol and Bertrand Iooss

References


B. Iooss, V. Chabridon and V. Thouvenot, Variance-based importance measures for machine learn-
ing model interpretability, Congres lambda-mu23, Saclay, France, 10-13 octobre 2022 https://hal.archives-ouvertes.fr/hal-03741384


See Also

src, lmg, pmvd
Examples

```r
# 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 = X1^2 + X2 + X3
y <- with(X, X1^2 + X2 + X3)

# sensitivity analysis
x <- pcc(X, y, nboot = 100)
print(x)
plot(x)

library(ggplot2)
ggplot(x)

x <- pcc(X, y, semi = TRUE, nboot = 100)
print(x)
plot(x)
```

---

**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 robustness indices related to a probability of exceedance of a model output (i.e. a failure probability), estimated by a Monte Carlo method. See Lemaitre et al. (2015).

**Usage**

```r
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 Truncated Gaussian, Left Truncated Gumbel. Using Gumbel 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 matrices, containing:

- 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


See Also

PLIquantile, PLIquantile_multivar, PLIsuperquantile, PLIsuperquantile_multivar
Examples

```r
# Model: Ishigami function with a threshold 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 = 100000
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(mfrow=c(1,1), mar=c(4,5,1,1))
plot(v_delta, BIshm[,2], ylim=c(-4,4), xlab=expression(delta), 
ylab=expression(hat(PLI[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)
```
PLI

BIshv=Toto[[1]]
SImshv=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(PLI[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*SImshv[,2],col="black")
lines(v_delta,BIshv[,2]-1.96*SImshv[,2],col="black")
lines(v_delta,BIshv[,1]+1.96*SImshv[,1],col="darkgreen")
lines(v_delta,BIshv[,1]-1.96*SImshv[,1],col="darkgreen")
lines(v_delta,BIshv[,3]+1.96*SImshv[,3],col="red")
lines(v_delta,BIshv[,3]-1.96*SImshv[,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(PLI[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*SImshv[,2],col="black")
lines(v_delta,BIshv[,2]-1.96*SImshv[,2],col="black")
lines(v_delta,BIshv[,1]+1.96*SImshv[,1],col="darkgreen")
lines(v_delta,BIshv[,1]-1.96*SImshv[,1],col="darkgreen")
lines(v_delta,BIshv[,3]+1.96*SImshv[,3],col="red")
lines(v_delta,BIshv[,3]-1.96*SImshv[,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)

# Example with an inverse probability transform
# (to obtain Gaussian inputs from Uniform ones)

# Monte Carlo sampling (the inputs are Uniform)
N = 100000
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

# Empirical transform (applied on the sample)
Xn <- matrix(0,nrow=N,ncol=3)
for (i in 1:3){
  ecdfx <- ecdf(X[,i])
  q <- ecdfx(X[,i])
  Xn[,i] <- qnorm(q) # Gaussian anamorphosis
  if (any(is.infinite(q)))
    Xn[which(is.infinite(q)),i] <- -Xn[which.min(Xn[,i]),i]
}

# Visualization of a perturbed density (the one of X1 perturbed on the mean)
delta_mean_gauss <- 1  # perturbed value on the mean of the Gaussian transform
Xtr <- quantile(ecdfx,pnorm(Xn[,1] + delta_mean_gauss))  # backtransform
par(mfrow=c(1,1))
plot(density(Xtr), col="red"); lines(density(X[,1]))

# sensitivity indices with perturbation of the mean

distributionIshigami = list()
for (i in 1:3){
distributionIshigami[[i]]=list("norm",c(0,1))
distributionIshigami[[i]]$r="rnorm"
}
ptsdef = Xn[T <- -7,]  # Failure points # failure points with Gaussian distrib.

v_delta = seq(-1.5,1.5,1/20)
Toto = PLI(failurepoints=ptsdef,failureprobabilityhat=pdefchap,samplesize=N,
deltasvector=v_delta,InputDistributions=distributionIshigami,type="MOY",
samedelta=TRUE)
BIshm = Toto[[1]]
SIshm = Toto[[2]]

par(mfrow=c(1,1),mar=c(4,5,1,1))
plot(v_delta,BIshm[,2],ylim=c(-4,4),xlab=expression(delta),
ylab=expression(hat(PLI[1+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)

---

**PLIquantile**

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

**Description**

PLIquantile computes the Perturbed-Law based Indices (PLI) for quantile, which are robustness indices related to a quantile of a model output, estimated by a Monte Carlo method, See Sueur et al. (2017) and Iooss et al. (2020).
Usage

\[
\text{PLIquantile(order, } x, y, \text{deltasvector, InputDistributions, type="MOY", same}\text{delta=TRUE, percentage=TRUE, nboot=0, conf=0.95, bootsample=TRUE)}
\]

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.
- **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 Truncated Gaussian, Left Truncated Gumbel. Using Gumbel 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.
- **percentage**: a boolean that defines the formula used for the PLI.
  - If it is set at FALSE, the initially proposed formula is used (see Sueur et al., 2017).
  - If not (set as TRUE), the PLI is given in percentage of variation of the quantile (see Iooss et al., 2020).
- **nboot**: the number of bootstrap replicates.
- **conf**: the confidence level for bootstrap confidence intervals.
- **bootsample**: If TRUE, the uncertainty about the original quantile estimation is taken into account in the PLI confidence intervals (see Iooss et al., 2021). If FALSE, standard confidence intervals are computed for the PLI. It mainly changes the CI at small delta values.
PLIquantile returns a list of matrices (each column corresponds to an input, each line corresponds to a twist of amplitude delta) containing the following components:

- **PLI**: the PLI.
- **PLICIinf**: the bootstrap lower confidence interval values of the PLI.
- **PLICIsup**: the bootstrap upper confidence interval values of the PLI.
- **quantile**: the perturbed quantile.
- **quantileCIinf**: the bootstrap lower confidence interval values of the perturbed quantile.
- **quantileCIsup**: the bootstrap upper confidence interval values of the perturbed quantile.

**Author(s)**

Paul Lemaitre, Bertrand Iooss, Thibault Delage and Roman Sueur

**References**


**See Also**

- PLI
- PLIsuperquantile
- PLIquantile_multivar
- PLIsuperquantile_multivar

**Examples**

```R
definition = list()```

# Model: 3D function

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

# Monte Carlo sampling

N = 5000
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
alpha <- 0.95  # quantile order

q95 = quantile(Y, alpha)
nboot=20  # put nboot=200 for consistency

# sensitivity indices with perturbation of the mean

v_delta = seq(-1, 1, 1/10)
toto = PLIquantile(alpha, X, Y, deltasvector=v_delta,
                   InputDistributions=distribution,type="MOY",samedelta=TRUE,
                   percentage=FALSE,nboot=nboot)

# Plotting the PLI

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

# Plotting the perturbed quantiles

par(mar=c(4, 5, 1, 1))
plot(v_delta, toto$quantile[, 2], ylim=c(1.5, 6.5), xlab=expression(delta),
     ylab=expression(hat(q[i*delta])), pch=19, cex=1.5)
points(v_delta, toto$quantile[, 1], col="darkgreen", pch=15, cex=1.5)
points(v_delta, toto$quantile[, 3], col="red", pch=17, cex=1.5)
lines(v_delta, toto$quantileCIinf[, 2], col="black")
lines(v_delta, toto$quantileCIinf[, 1], col="darkgreen")
lines(v_delta, toto$quantileCIinf[, 3], col="red")
lines(v_delta, toto$quantileCIsup[, 2], col="black")
lines(v_delta, toto$quantileCIsup[, 1], col="darkgreen")
lines(v_delta, toto$quantileCIsup[, 3], col="red")
abline(h=q95, lty=2)

PLIquantile_multivar

Perturbed-Law based sensitivity Indices (PLI) for quantile and simultaneous perturbations of 2 inputs

```r
legend(0.5, 2.4, legend = c("X1", "X2", "X3"),
      col = c("darkgreen", "black", "red"), pch = c(15, 19, 17), cex = 1.5)

# Plotting the PLI in percentage with refined confidence intervals
toto = PLIquantile(alpha, X, Y, deltasvector = v_delta,
                  InputDistributions = distribution, type = "MOY", samedelta = TRUE,
                  percentage = TRUE, nboot = nboot, bootsample = FALSE)

par(mar = c(4, 5, 1, 1))
plot(v_delta, toto$PLI[, 2], ylim = c(-0.6, 0.6), xlab = expression(delta),
     ylab = expression(hat(PLI[i*delta])), pch = 19, cex = 1.5)
points(v_delta, toto$PLI[, 1], col = "darkgreen", pch = 15, cex = 1.5)
points(v_delta, toto$PLI[, 3], col = "red", pch = 17, cex = 1.5)
lines(v_delta, toto$PLICIinf[, 2], col = "black")
lines(v_delta, toto$PLICIsup[, 2], col = "black")
lines(v_delta, toto$PLICIinf[, 1], col = "darkgreen")
lines(v_delta, toto$PLICIsup[, 1], col = "darkgreen")
lines(v_delta, toto$PLICIinf[, 3], col = "red")
lines(v_delta, toto$PLICIsup[, 3], col = "red")
abline(h = 0, lty = 2)
legend(0, 0.6, legend = c("X1", "X2", "X3"),
       col = c("darkgreen", "black", "red"), pch = c(15, 19, 17), cex = 1.5)

# another visualization by using the plotCI() fct
# (from plotrix package) for the CI plotting(from Vanessa Verges)

library(plotrix)
parameters = list(colors = c("darkgreen", "black", "red"),
                  symbols = c(15, 19, 17), overlay = c(FALSE, TRUE, TRUE))
par(mar = c(4, 5, 1, 1), xpd = TRUE)
for (i in 1:3){
  plotCI(v_delta, toto$PLI[, i], ui = toto$PLICIsup[, i], li = toto$PLICIinf[, i],
         cex = 1.5, col = parameters$colors[i], pch = parameters$symbols[i],
         add = parameters$overlay[i], xlab = "", ylab = "")
}
title(xlab = expression(delta), ylab = expression(hat(PLI[i*delta])),
abline(h = 0, lty = 2)
legend("topleft", legend = c("X1", "X2", "X3"), col = parameters$colors,
        pch = parameters$symbols, cex = 1.5)
```

PLIquantile_multivar
**Description**

`PLIquantile_multivar` computes the Perturbed-Law based Indices (PLI) for quantile and simultaneous perturbations of the means of 2 inputs, estimated by a Monte Carlo method.

**Usage**

```r
PLIquantile_multivar(order,x,y,inputs,deltasvector,InputDistributions,samedelta=TRUE,percentage=TRUE,nboot=0,conf=0.95,bootsample=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.
- `inputs` the vector of the two inputs' indices for which the indices will be computed.
- `deltasvector` a vector containing the values of the perturbed means for which the indices will be computed. Warning: if `samedelta=FALSE`, `deltasvector` has to be the vector of deltas (mean perturbations).
- `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 Truncated Gaussian, Left Truncated Gumbel. Using Gumbel requires the package `evd`.
- `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`.
- `percentage` a boolean that defines the formula used for the PLI.
  - If it is set at FALSE, the initially proposed formula is used (see Sueur et al., 2017).
  - If not (set as TRUE), the PLI is given in percentage of variation of the quantile (see Iooss et al., 2021).
- `nboot` the number of bootstrap replicates.
- `conf` the confidence level for bootstrap confidence intervals.
- `bootsample` If TRUE, the uncertainty about the original quantile estimation is taken into account in the PLI confidence intervals (see Iooss et al., 2021). If FALSE, standard confidence intervals are computed for the PLI. It mainly changes the CI at small delta values.

**Details**

This function does not allow perturbations on the variance of the inputs' distributions.
PLIquantile_multivar

Value

PLIquantile_multivar returns a list of matrices (delta twist of input 1 (in rows) vs. delta twist of input 2 (in columns)) containing the following components:

- PLI the PLI.
- PLICIinf the bootstrap lower confidence interval values of the PLI.
- PLICIsup the bootstrap upper confidence interval values of the PLI.
- quantile the perturbed quantile.
- quantileCIinf the bootstrap lower confidence interval values of the perturbed quantile.
- quantileCIsup the bootstrap upper confidence interval values of the perturbed quantile.

Author(s)

Bertrand Iooss

References


See Also

PLI, PLIquantile, PLIsuperquantile, PLIsuperquantile_multivar

Examples

```R
# Model: 3D function
distribution = list()
for (i in 1:3) distribution[[i]]=list("norm",c(0,1))
N = 5000
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
alpha <- 0.95
nboot <- 20 # put nboot=200 for consistency
q95 = quantile(Y,alpha)
v_delta = seq(-1,1,1/10)
toto12 = PLIquantile_multivar(alpha,X,Y,c(1,2),deltasvector=v_delta,
   InputDistributions=distribution,samedelta=TRUE)
toto = PLIquantile(alpha,X,Y,deltasvector=v_delta,InputDistributions=distribution,
   type="MOY",samedelta=TRUE,nboot=0)

par(mar=c(4,5,1,1))
plot(v_delta,diag(toto12$PLI),ylim=c(-1,1),xlab=expression(delta),
ylab=expression(hat(PLI[i*delta])),pch=16,cex=1.5,col="blue")
points(v_delta,toto$PLI[,1],col="darkgreen",pch=15,cex=1.5)
points(v_delta,toto$PLI[,2],col="black",pch=19,cex=1.5)
points(v_delta,toto$PLI[,3],col="red",pch=17,cex=1.5)
abline(h=0,lty=2)
legend(-1,1,legend=c("X1","X2","X3","X1X2"),col=c("darkgreen","black","red","blue"),
pch=c(15,19,17,16),cex=1.5)

# with bootstrap
v_delta = seq(-1,1,2/10)
toto12 = PLIquantile_multivar(alpha,X,Y,c(1,2),deltasvector=v_delta,
   InputDistributions=distribution,samedelta=TRUE,nboot=nboot,bootsample=FALSE)
toto = PLIquantile(alpha,X,Y,deltasvector=v_delta,InputDistributions=distribution,
   type="MOY",samedelta=TRUE,nboot=0,bootsample=FALSE)

par(mar=c(4,5,1,1))
plot(v_delta,diag(toto12$PLI),ylim=c(-1,1),xlab=expression(delta),
ylab=expression(hat(PLI[i*delta])),pch=16,cex=1.5,col="blue")
points(v_delta,toto$PLI[,1],col="darkgreen",pch=15,cex=1.5)
points(v_delta,toto$PLI[,2],col="black",pch=19,cex=1.5)
points(v_delta,toto$PLI[,3],col="red",pch=17,cex=1.5)
lines(v_delta,diag(toto12$PLICIinf),col="blue")
lines(v_delta,diag(toto12$PLICIsup),col="blue")
lines(v_delta,toto$PLICIinf[,2],col="black")
lines(v_delta,toto$PLICIsup[,2],col="black")
lines(v_delta,toto$PLICIinf[,1],col="darkgreen")
lines(v_delta,toto$PLICIsup[,1],col="darkgreen")
lines(v_delta,toto$PLICIinf[,3],col="red")
lines(v_delta,toto$PLICIsup[,3],col="red")
abline(h=0,lty=2)
legend(-1,1,legend=c("X1","X2","X3","X1X2"),col=c("darkgreen","black","red","blue"),
pch=c(15,19,17,16),cex=1.5)

# another visualizations by using the plotrix,
# viridisLite, lattice and grid packages (from Vanessa Verges)
library(plotrix)

parameters = list(colors=c("darkgreen","black","red"), symbols=c(15,19,17))
par(mar=c(4,5,1,1), xpd=TRUE)
plotCI(v_delta, diag(toto12$PLI), ui=diag(toto12$PLICIsup), li=diag(toto12$PLICIinf),
    xlab=expression(delta), ylab=expression(hat(PLI[i*delta]),
    cex=1.5, col="blue", pch=16)
for (i in 1:3){
    plotCI(v_delta, toto$PLI[,i], ui=toto$PLICIsup[,i], li=toto$PLICIinf[,i],
        cex=1.5, col=parameters$colors[i], pch=parameters$symbols[i],
        add=TRUE)
}
abline(h=0, lty=2)
legend("topleft", legend=c("X1", "X2", "X3", "X1X2"),
    col=c(parameters$colors,"blue"), pch=c(parameters$symbols,16), cex=1.5)

# Visu of all the PLIs (at any paired combinations of deltas)
library(viridisLite)
library(lattice)
library(grid)

colnames(toto12$PLI) = round(v_delta,2)
rownames(toto12$PLI) = round(v_delta,2)
coul = viridis(100)
levelplot(toto12$PLI, col.regions = coul, xlab=bquote(delta[X~.(1)]), ylab=bquote(delta[X~.(2)]),
    main=bquote(hat(PLI)[quantile[X[-(1)]-X[-(2)]]]))

### PLIsuperquantile

**Perturbed-Law based sensitivity Indices (PLI) for superquantile**

**Description**

PLIsuperquantile computes the Perturbed-Law based Indices (PLI) for superquantile, which are robustness indices related to a superquantile of a model output, estimated by a Monte Carlo method. See Iooss et al. (2020).

**Usage**

```r
PLIsuperquantile(order, x, y, deltasvector, InputDistributions, type="MOY", samedelta=TRUE, percentage=TRUE, nboot=0, conf=0.95, bootsample=TRUE, bias=TRUE)
```

**Arguments**

- **order**
  - the order of the superquantile to estimate.
the matrix of simulation points coordinates, one column per variable.
y the vector of model outputs.
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 Trucated Gumbel. Using Gumbel 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.
percentage a boolean that defines the formula used for the PLI.
  • If it is set at FALSE, the classical formula used in the bibliographical references is used.
  • If not (set as TRUE), the PLI is given in percentage of variation of the superquantile (even if it is negative).
nboot the number of bootstrap replicates.
conf the confidence level for bootstrap confidence intervals.
bootsample If TRUE, the uncertainty about the original quantile estimation is taken into account in the PLI confidence intervals (see Iooss et al., 2020). If FALSE, standard confidence intervals are computed for the PLI. It mainly changes the CI at small delta values.
bias defines the version of PLI-superquantile:
  • If it is set at "TRUE", it gives the mean of outputs above the perturbed quantile (alternative formula)
  • If it is set at "FALSE", it gives the mean of perturbed outputs above the perturbed quantile (original formula)

**Value**

PLIsuperquantile returns a list of matrices (each column corresponds to an input, each line corresponds to a twist of amplitude delta) containing the following components:

PLI the PLI.
PLI superquantile

PLICI\text{inf} \quad \text{the bootstrap lower confidence interval values of the PLI.}
PLICISup \quad \text{the bootstrap upper confidence interval values of the PLI.}
superquantile \quad \text{the perturbed superquantile.}
superquantileCI\text{inf} \quad \text{the bootstrap lower confidence interval values of the perturbed superquantile.}
superquantileCISup \quad \text{the bootstrap upper confidence interval values of the perturbed superquantile.}

Author(s)

Bertrand Iooss

References


See Also

PLI, PLIquantile, PLIsuperquantile\_multivar

Examples

```
# Model: 3D function

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

# Monte Carlo sampling

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
alpha <- 0.95

q95 = quantile(Y,alpha)
sq95a <- mean(Y*(Y>q95)/(1-alpha)) ; sq95b <- mean(Y[Y>q95])
nboot=20 # change to nboot=200 for consistency

# sensitivity indices with perturbation of the mean
```
v_delta = seq(-1,1,1/10)
toto = PLIsuperquantile(alpha,X,Y,deltasvector=v_delta, 
InputDistributions=distribution,type="MOY",samedelta=TRUE, 
percentage=FALSE,nboot=nboot,bias=TRUE)

# Plotting the PLI
par(mar=c(4,5,1,1))
plot(v_delta,toto$PLI[,2],ylim=c(-0.5,0.5),xlab=expression(delta), 
ylab=expression(hat(PLI[i*delta])),pch=19,cex=1.5)
points(v_delta,toto$PLI[,1],col="darkgreen",pch=15,cex=1.5)
points(v_delta,toto$PLI[,3],col="red",pch=17,cex=1.5)
lines(v_delta,toto$PLICIinf[,2],col="black")
lines(v_delta,toto$PLICIsup[,2],col="black")
lines(v_delta,toto$PLICIinf[,1],col="darkgreen")
lines(v_delta,toto$PLICIsup[,1],col="darkgreen")
lines(v_delta,toto$PLICIinf[,3],col="red")
lines(v_delta,toto$PLICIsup[,3],col="red")
abline(h=0,lty=2)
legend(-1,0.5,legend=c("X1","X2","X3"), 
col=c("darkgreen","black","red"),pch=c(15,19,17),cex=1.5)

# Plotting the perturbed superquantiles
par(mar=c(4,5,1,1))
plot(v_delta,toto$superquantile[,2],ylim=c(3,7),xlab=expression(delta), 
ylab=expression(hat(q[i*delta])),pch=19,cex=1.5)
points(v_delta,toto$superquantile[,1],col="darkgreen",pch=15,cex=1.5)
points(v_delta,toto$superquantile[,3],col="red",pch=17,cex=1.5)
lines(v_delta,toto$superquantileCIinf[,2],col="black")
lines(v_delta,toto$superquantileCIsup[,2],col="black")
lines(v_delta,toto$superquantileCIinf[,1],col="darkgreen")
lines(v_delta,toto$superquantileCIsup[,1],col="darkgreen")
lines(v_delta,toto$superquantileCIinf[,3],col="red")
lines(v_delta,toto$superquantileCIsup[,3],col="red")
abline(h=q95,lty=2)
legend(-1,7,legend=c("X1","X2","X3"), 
col=c("darkgreen","black","red"),pch=c(15,19,17),cex=1.5)

# Plotting the unbiased PLI in percentage with refined confidence intervals
toto = PLIsuperquantile(alpha,X,Y,deltasvector=v_delta, 
InputDistributions=distribution,type="MOY",samedelta=TRUE,percentage=TRUE, 
nboot=nboot,bootsample=FALSE,bias=FALSE)

par(mar=c(4,5,1,1))
plot(v_delta,toto$PLI[,2],ylim=c(-0.4,0.5),xlab=expression(delta), 
ylab=expression(hat(PLI[i*delta])),pch=19,cex=1.5)
points(v_delta,toto$PLI[,1],col="darkgreen",pch=15,cex=1.5)
points(v_delta,toto$PLI[,3],col="red",pch=17,cex=1.5)
lines(v_delta,toto$PLICIinf[,2],col="black")
lines(v_delta,toto$PLICIsup[,2],col="black")
lines(v_delta,toto$PLICIinf[,1],col="darkgreen")
lines(v_delta,toto$PLICIsup[,1],col="darkgreen")
lines(v_delta,toto$PLICIinf[,3],col="red")
lines(v_delta,toto$PLICIsup[,3],col="red")
**PLIsuperquantile_multivar**

**Perturbed-Law based sensitivity Indices (PLI) for superquantile and simultaneous perturbations of 2 inputs**

**Description**

PLIsuperquantile_multivar computes the Perturbed-Law based Indices (PLI) for superquantile and simultaneous perturbations of the means of 2 inputs, estimated by a Monte Carlo method.

**Usage**

```r
PLIsuperquantile_multivar(order,x,y,inputs,deltasvector,InputDistributions,samedelta=TRUE, percentage=TRUE,nboot=0,conf=0.95,bootsample=TRUE,bias=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.
- **inputs** the vector of the two inputs' indices for which the indices will be computed.
deltasvector  a vector containing the values of the perturbed means for which the indices will be computed. Warning: if samedelta=FALSE, deltasvector has to be the vector of deltas (mean perturbations)

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 Truncated Gaussian, Left Truncated Gumbel. Using Gumbel requires the package evd.

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.

percentage  a boolean that defines the formula used for the PLI.

• If it is set at FALSE, the initially proposed formula is used (see Sueur et al., 2017).
• If not (set as TRUE), the PLI is given in percentage of variation of the superquantile (see Iooss et al., 2021).

nboot  the number of bootstrap replicates.

conf  the confidence level for bootstrap confidence intervals.

bootsample  If TRUE, the uncertainty about the original quantile estimation is taken into account in the PLI confidence intervals (see Iooss et al., 2021). If FALSE, standard confidence intervals are computed for the PLI. It mainly changes the CI at small delta values.

bias  defines the version of PLI-superquantile:

• If it is set at "TRUE", it gives the mean of outputs above the perturbed quantile (alternative formula)
• If it is set at "FALSE", it gives the mean of perturbed outputs above the perturbed quantile (original formula)

Details

This function does not allow perturbations on the variance of the inputs’ distributions.

Value

PLIsuperquantile_multivar returns a list of matrices (delta twist of input 1 (in rows) vs. delta twist of input 2 (in columns)) containing the following components:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLI</td>
<td>the PLI.</td>
</tr>
<tr>
<td>PLICInf</td>
<td>the bootstrap lower confidence interval values of the PLI.</td>
</tr>
<tr>
<td>PLICIsup</td>
<td>the bootstrap upper confidence interval values of the PLI.</td>
</tr>
<tr>
<td>quantile</td>
<td>the perturbed quantile.</td>
</tr>
<tr>
<td>quantileCInf</td>
<td>the bootstrap lower confidence interval values of the perturbed superquantile.</td>
</tr>
<tr>
<td>quantileCIsup</td>
<td>the bootstrap upper confidence interval values of the perturbed superquantile.</td>
</tr>
</tbody>
</table>
Author(s)

Bertrand Iooss

References


See Also

*PLI, PLIquantile, PLIsuperquantile, PLIquantile_multivar*

Examples

# Model: 3D function
distribution = list()
for (i in 1:3) distribution[[i]]=list("norm",c(0,1))
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
alpha <- 0.95
nboot <- 20 # put nboot=200 for consistency
q95 = quantile(Y, alpha)
sq95a <- mean(Y*(Y>q95)/(1-alpha)) ; sq95b <- mean(Y[Y>q95])
v_delta = seq(-1,1,1/10)
toto12 = PLIsuperquantile_multivar(alpha,X,Y,c(1,2),deltasvector=v_delta,
   InputDistributions=distribution,samedelta=TRUE,bias=FALSE)
toto = PLIsuperquantile(alpha,X,Y,deltasvector=v_delta,InputDistributions=distribution,
   type="MOY",samedelta=TRUE,nboot=0,bias=FALSE)
par(mar=c(4,5,1,1))
plot(v_delta,diag(toto12$PLI),ylim=c(-1,1),xlab=expression(delta),
     ylab=expression(hat(PLI[i*delta])),pch=16,cex=1.5,col="blue")
points(v_delta,toto$PLI[,1],col="darkgreen",pch=15,cex=1.5)
points(v_delta,toto$PLI[,2],col="black",pch=19,cex=1.5)
points(v_delta,toto$PLI[,3],col="red",pch=17,cex=1.5)
abline(h=0,lty=2)
legend(-1.1,1,legend=c("X1","X2","X3","X1X2"),col=c("darkgreen","black","red","blue"),
pch=c(15,19,17,16),cex=1.5)

# with bootstrap (put in comment because too long for the CRAN tests)

v_delta = seq(-1,1,2/10)
toto12 = PLIsuperquantile_multivar(alpha,X,Y,c(1,2),deltasvector=v_delta,
    InputDistributions=distribution,samedelta=TRUE,nboot=nboot,bootsample=FALSE,bias=FALSE)
toto = PLIsuperquantile(alpha,X,Y,deltasvector=v_delta,InputDistributions=distribution,
    type="MOY",samedelta=TRUE,nboot=nboot,bootsample=FALSE,bias=FALSE)
par(mar=c(4,5,1,1))
plot(v_delta,diag(toto12$PLI),ylim=c(-1,1),xlab=expression(delta),
     ylab=expression(hat(PLI[i*delta])),pch=16,cex=1.5,col="blue")
points(v_delta,toto$PLI[,1],col="darkgreen",pch=15,cex=1.5)
points(v_delta,toto$PLI[,2],col="black",pch=19,cex=1.5)
points(v_delta,toto$PLI[,3],col="red",pch=17,cex=1.5)
lines(v_delta,diag(toto12$PLICIinf),col="blue")
lines(v_delta,diag(toto12$PLICIsup),col="blue")
lines(v_delta,toto$PLICIinf[,2],col="black")
lines(v_delta,toto$PLICIsup[,2],col="black")
lines(v_delta,toto$PLICIinf[,1],col="darkgreen")
lines(v_delta,toto$PLICIsup[,1],col="darkgreen")
lines(v_delta,toto$PLICIinf[,3],col="red")
lines(v_delta,toto$PLICIsup[,3],col="red")
abline(h=0,lty=2)
legend(-1.1,1,legend=c("X1","X2","X3","X1X2"),col=c("darkgreen","black","red","blue"),
pch=c(15,19,17,16),cex=1.5)

# another visualizations by using the plotrix,
# viridisLite, lattice and grid packages (from Vanessa Verges)

library(plotrix)
parameters = list(colors=c("darkgreen","black","red"),symbols=c(15,19,17))
par(mar=c(4,5,1,1),xpd=TRUE)
plotCI(v_delta,diag(toto12$PLI),ui=diag(toto12$PLICIsup),li=diag(toto12$PLICIinf),
       xlab=expression(delta),ylab=expression(hat(PLI[i*delta])),
       main=bquote("PLI-superquantile (N =.(N) ~ \alpha =.(alpha) ~ "
       cex=1.5,col="blue",pch=16)
for (i in 1:3){
    plotCI(v_delta,toto$PLI[,i],ui=toto$PLICIsup[,i],li=toto$PLICIinf[,i],
           cex=1.5,col=parameters$colors[i],pch=parameters$symbols[i],
           add=TRUE)
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

```r
## 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`: .

```r
colnames(toto12$PLI) = round(v_delta,2)
rownames(toto12$PLI) = round(v_delta,2)
coul = viridis(100)
levelplot(toto12$PLI,col.regions=coul,main=bquote(hat(PLI)[superquantile[X1-X2]]),
xlab=bquote(delta[X1]),ylab=bquote(delta[X2]))
```
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 $F_n(x)$, where $F_n$ 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

pme_knn

Data-given proportional marginal effects estimation via nearest-neighbors procedure

Description

pme_knn computes the proportional marginal effects (PME), from Herin et al. (2022) via a nearest neighbor estimation. Parallelized computations are possible to accelerate the estimation process. It can be used with categorical inputs (which are transformed with one-hot encoding before computing the nearest-neighbors), dependent inputs and multiple outputs. For large sample sizes, the nearest neighbour algorithm can be significantly accelerated by using approximate nearest neighbour search.
**Usage**

```r
pme_knn(model=NULL, X, method = "knn", tol = NULL, marg = T, n.knn = 2,
        n.limit = 2000, noise = F, rescale = F, nboot = NULL, boot.level = 0.8,
        conf=0.95, parl=NULL, ...)
```

## S3 method for class 'pme_knn'
tell(x, y, ...)

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

## S3 method for class 'pme_knn'
plot(x, ylim = c(0,1), ...)

## S3 method for class 'pme_knn'
ggplot(x, ylim = c(0, 1), ...)

**Arguments**

- `model` a function defining the model to analyze, taking `X` as an argument.
- `X` a matrix or data frame containing the observed inputs.
- `method` the algorithm to be used for estimation, either "rank" or "knn", see details. Default is `method="knn"`.
- `tol` tolerance under which an input is considered as being a zero input. See details.
- `marg` whether to chose the closed Sobol' (`FALSE`) or total Sobol' (`TRUE`) indices as value functions.
- `n.knn` the number of nearest neighbours used for estimation.
- `n.limit` sample size limit above which approximate nearest neighbour search is activated.
- `noise` a logical which is `TRUE` if the model or the output sample is noisy. See details.
- `rescale` a logical indicating if continuous inputs must be rescaled before distance computations. If `TRUE`, continuous inputs are first whitened with the ZCA-cor whitening procedure (cf. `whiten()` function in package `whitening`). If the inputs are independent, this first step will have a very limited impact. Then, the resulting whitened inputs are individually modified via a copula transform such that each input has the same scale.
- `n.boot` the number of bootstrap resamples for the bootstrap estimate of confidence intervals. See details.
- `boot.level` a numeric between 0 and 1 for the proportion of the bootstrap sample size.
- `conf` the confidence level of the bootstrap confidence intervals.
- `parl` number of cores on which to parallelize the computation. If `NULL`, then no parallelization is done.
- `x` the object returned by `pme_knn`.
- `y` a numeric univariate vector containing the observed outputs.
- `ylim` the y-coordinate limits for plotting.
- `...` additional arguments to be passed to `model`, or to the methods, such as graphical parameters (see `par`).
Details

For method="rank", the estimator is defined in Gamboa et al. (2020) following Chatterjee (2019). For first-order indices it is based on an input ranking (same algorithm as in sobol.rank) while for higher orders, it uses an approximate heuristic solution of the traveling salesman problem applied to the input sample distances (cf. TSP() function in package TSP). For method="knn", ranking and TSP are replaced by a nearest neighbour search as proposed in Broto et al. (2020) and in Azadkia & Chatterjee (2020) for a similar coefficient.

The computation is done using the subset procedure, defined in Broto, Bachoc and Depecker (2020), that is computing all the Sobol' closed indices for all possible sub-models first, and then computing the proportional values recursively, as detailed in Feldman (2005), but using an extension to non strictly positive games (Margot Herin 2021).

Since bootstrap creates ties which are not accounted for in the algorithm, confidence intervals are obtained by sampling without replacement with a proportion of the total sample size boot.level, drawn uniformly.

If the outputs are noisy, the argument noise can be used: it only has an impact on the estimation of one specific sensitivity index, namely \( \frac{Var(E(Y|X_1, \ldots, X_p))}{Var(Y)} \). If there is no noise this index is equal to 1, while in the presence of noise it must be estimated.

The distance used for subsets with mixed inputs (continuous and categorical) is the Euclidean distance, thanks to a one-hot encoding of categorical inputs.

If too many cores for the machine are passed on to the par1 argument, the chosen number of cores is defaulted to the available cores minus one.

If marg = TRUE (default), the chosen value function to compute the proportional values are the total Sobol' indices (dual of the underlying cooperative game). If marg = FALSE, then the closed Sobol' indices are used instead. Differences may appear between the two.

Zero inputs are defined by the tol argument. If null, then inputs with:

\[ S_{(i)}^T = 0 \]

are considered as zero input in the detection of spurious variables. If provided, zero inputs are detected when:

\[ S_{(i)}^T \leq tol \]

Value

pme_knn returns a list of class "pme_knn":

- call: the matched call.
- PME: the estimations of the PME indices.
- VE: the estimations of the closed Sobol' indices for all possible sub-models.
- indices: list of all subsets corresponding to the structure of VE.
- method: which estimation method has been used.
- conf_int: a matrix containing the estimations, biais and confidence intervals by bootstrap (if nboot>0).
- X: the observed covariates.
y the observed outcomes.
n.knn value of the n.knn argument.
rescale wheter the design matrix has been rescaled.
n.limit value of the n.limit argument.
boot.level value of the boot.level argument.
noise wheter the PME must sum up to one or not.
boot logical, wheter bootstrap confidence interval estimates have been performed.
nboot value of the nboot argument.
parl value of the parl argument.
conf value of the conf argument.
marg value of the marg argument.
tol value of the tol argument.

Author(s)

Marouane Il Idrissi, Margot Herin

References


B. Iooss, V. Chabridon and V. Thouvenot, Variance-based importance measures for machine learning model interpretability, Congres lambda-mu23, Saclay, France, 10-13 octobre 2022 https://hal.archives-ouvertes.fr/hal-03741384


See Also

sobolrank, shapleysobol_knn, shapleyPermEx, shapleySubsetMc, lmg, pmvd
Examples

```r
library(parallel)
library(doParallel)
library(foreach)
library(gtools)
library(boot)
library(RANN)

###########################################################
# Linear Model with Gaussian correlated inputs
###########################################################
library(mvtnorm)
set.seed(1234)
n <- 1000
beta<-c(1,-1,0.5)
sigma<-matrix(c(1,0,0,
0,1,-0.8,
0,-0.8,1),
ncol=3)
X <-rmvnorm(n, rep(0,3), sigma)
colnames(X)<-c("X1","X2", "X3")
y <- X%*%beta + rnorm(n,0,2)

# Without Bootstrap confidence intervals
x<-pme_knn(model=NULL, X=X,
n.knn=3,
noise=TRUE)
tell(x,y)
print(x)
plot(x)

# With Bootstrap confidence intervals
x<-pme_knn(model=NULL, X=X,
nboot=10,
n.knn=3,
noise=TRUE,
boot.level=0.7,
conf=0.95)
tell(x,y)
print(x)
plot(x)

###########################################################
# Test case: the Ishigami function
# Example with given data and the use of approximate nearest neighbour search
```
n <- 5000
X <- data.frame(matrix(-pi+2*pi*runif(3 * n), nrow = n))
Y <- ishigami.fun(X)
x <- pme_knn(model = NULL, X = X, method = "knn", n.knn = 5, n.limit = 2000)
tell(x, Y)
plot(x)

library(ggplot2); ggplot(x)

#########################################################
# Test case : Linear model (3 Gaussian inputs including 2 dependent) with scaling
# See Iooss and Prieur (2019)
library(mvtnorm) # Multivariate Gaussian variables
library(whitening) # For scaling
modlin <- function(X) apply(X, 1, sum)
d <- 3
n <- 10000
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, Cormat)
X <- Xall(n)
x <- pme_knn(model = modlin, X = X, method = "knn", n.knn = 5, rescale = TRUE, n.limit = 2000)
print(x)
plot(x)

pmvd

Proportional Marginal Variance Decomposition indices for linear and logistic models

Description

pmvd computes the PMVD indices derived from Feldman (2005) applied to the explained variance \( R^2 \) as a performance metric. They allow for relative importance indices by \( R^2 \) decomposition for linear and logistic regression models. These indices allocate a share of \( R^2 \) to each input based on a Proportional attribution system, allowing for covariates with null regression coefficients to have indices equal to 0, despite their potential dependence with other covariates (Exclusion principle).

Usage

pmvd(X, y, logistic = FALSE, tol = NULL, rank = FALSE, nboot = 0, conf = 0.95, max.iter = 1000, parl = NULL)
## S3 method for class 'pmvd'
print(x, ...)
## S3 method for class 'pmvd'
plot(x, ylim = c(0, 1), ...)
Arguments

- **X**: a matrix or data frame containing the observed covariates (i.e., features, input variables...).
- **y**: a numeric vector containing the observed outcomes (i.e., dependent variable). If logistic=TRUE, can be a numeric vector of zeros and ones, or a logical vector, or a factor.
- **logistic**: logical. If TRUE, the analysis is done via a logistic regression (binomial GLM).
- **tol**: covariates with absolute marginal contributions less or equal to tol are omitted. By default, if tol=NULL, only covariates with no marginal contribution are omitted.
- **rank**: logical. If TRUE, the analysis is done on the ranks.
- **nboot**: the number of bootstrap replicates for the computation of confidence intervals.
- **conf**: the confidence level of the bootstrap confidence intervals.
- **max.iter**: if logistic=TRUE, the maximum number of iterative optimization steps allowed for the logistic regression. Default is 1000.
- **parl**: number of cores on which to parallelize the computation. If NULL, then no parallelization is done.
- **xlim**: the object returned by lmg.
- **ylim**: the y-coordinate limits of the plot.
- **...**: arguments to be passed to methods, such as graphical parameters (see par).

Details

The computation of the PMVD is done using the recursive method defined in Feldman (2005), but using the subset procedure defined in Broto, Bachoc and Depecker (2020), that is computing all the $R^2$ for all possible sub-models first, and then computing $P(\cdot)$ recursively for all subsets of covariates. See Il Idrissi et al. (2021).

For logistic regression (logistic=TRUE), the $R^2$ value is equal to:

$$R^2 = 1 - \frac{\text{model deviance}}{\text{null deviance}}$$

If either a logistic regression model (logistic = TRUE), or any column of X is categorical (i.e., of class factor), then the rank-based indices cannot be computed. In both those cases, rank = FALSE is forced by default (with a warning).

If too many cores for the machine are passed on to the parl argument, the chosen number of cores is defaulted to the available cores minus one.

Spurious covariates are defined by the tol argument. If null, then covariates with:

$$w(\{i\}) = 0$$

are omitted, and their pmvd index is set to zero. In other cases, the spurious covariates are detected by:

$$|w(\{i\})| \leq \text{tol}$$
Value

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

- `call`: the matched call.
- `pmvd`: the estimations of the PMVD indices.
- `R2s`: the estimations of the $R^2$ for all possible sub-models.
- `indices`: list of all subsets corresponding to the structure of R2s.
- `P`: the values of $P(.)$ of all subsets for recursive computing. Equal to NULL if bootstrap estimates are made.
- `conf_int`: a matrix containing the estimations, biais and confidence intervals by bootstrap (if `nboot>0`).
- `X`: the observed covariates.
- `y`: the observed outcomes.
- `logistic`: logical. TRUE if the analysis has been made by logistic regression.
- `boot`: logical. TRUE if bootstrap estimates have been produced.
- `nboot`: number of bootstrap replicates.
- `rank`: logical. TRUE if a rank analysis has been made.
- `parl`: number of chosen cores for the computation.
- `conf`: level for the confidence intervals by bootstrap.

Author(s)

Marouane Il Idrissi

References


B. Iooss, V. Chabridon and V. Thouvenot, Variance-based importance measures for machine learning model interpretability, Congres lambda-mu23, Saclay, France, 10-13 octobre 2022 https://hal.archives-ouvertes.fr/hal-03741384

See Also

`pcc`, `src`, `lmg`, `pme_knn`
Examples

```r
library(parallel)
library(gtools)
library(boot)
library(mvtnorm)

set.seed(1234)
n <- 100
beta <- c(1,-2,3)
sigma <- matrix(c(1,0,0,
                 0,1,-0.8,
                 0,-0.8,1),
                 nrow=3,
                 ncol=3)

############################
# Gaussian correlated inputs
X <- rmvnorm(n, rep(0,3), sigma)
############################
# Linear Model
y <- X%*%beta + rnorm(n)
# Without Bootstrap confidence intervals
x <- pmvd(X, y)
print(x)
plot(x)
# With Bootstrap confidence intervals
x <- pmvd(X, y, nboot=100, conf=0.95)
print(x)
plot(x)
# Rank-based analysis
x <- pmvd(X, y, rank=TRUE, nboot=100, conf=0.95)
print(x)
plot(x)

############################
# Logistic Regression
y <- as.numeric(X%*%beta + rnorm(n)>0)
x <- pmvd(X,y, logistic = TRUE)
plot(x)
print(x)
# Parallel computing
# x <- pmvd(X,y, logistic = TRUE, par=2)
# plot(x)
# print(x)
```
Description

This program computes the squared coefficient of the function decomposition in the tensor basis formed by eigenfunctions of Poincare differential operators. After division by the variance of the model output, it provides lower bounds of first-order and total Sobol’ indices.

Usage

\[ \text{PoincareChaosSqCoef}(\text{PoincareEigen}, \text{multiIndex}, \text{design}, \text{output}, \text{outputGrad} = \text{NULL}, \text{inputIndex} = 1, \text{der} = \text{FALSE}, \text{method} = "\text{unbiased}" ) \]

Arguments

- **PoincareEigen**: output list from PoincareOptimal() function
- **multiIndex**: vector of indices \((l_1, ..., l_d)\). A coordinate equal to 0 corresponds to the constant basis function 1
- **design**: design of experiments (matrix of size \(n \times d\)) with \(d\) the number of inputs and \(n\) the number of observations
- **output**: vector of length \(n\) \((y_1, ..., y_n)\) of output values at design points
- **outputGrad**: matrix \(n \times d\) whose columns contain the output partial derivatives at design points
- **inputIndex**: index of the input variable (between 1 and \(d\))
- **der**: logical (default=FALSE): should we use the formula with derivatives to compute the squared coefficient?
- **method**: "biased" or "unbiased" formula when estimating the squared integral. See \(\text{squaredIntEstim}\)

Details

Similarly to polynomial chaos, where tensors of polynomials are used, we consider here tensor basis formed by eigenfunctions of Poincare differential operators. This basis is also orthonormal, and Parseval formula lead to lower bound for (unnormalized) Sobol, total Sobol indices, and any variance-based index. Denoting by \( (e_{1,l_1}...e_{d,l_d}) \) one tensor basis, the corresponding coefficient is equal to

\[ c_{l_1,...,l_d} = \langle f, e_{1,l_1}...e_{d,l_d} \rangle. \]

For a given input variable (say \(x_1\) to simplify notations), it can be rewritten with derivatives as:

\[ c_{l_1,...,l_d} = \langle df/dx_1, de_{1,l_1}/dx_1e_{2,l_2}...e_{d,l_d} \rangle / \text{eigenvalue}_{1,l_1} \]

The function returns an estimate of \(c_{l_1,...,l_d}^2\), corresponding to one of these two forms (derivative-free, or derivative-based).
Value
An estimate of the squared coefficient.

Author(s)
Olivier Roustant and Bertrand Iooss

References

See Also
PoincareOptimal

Examples

# A simple example

g <- function(x, a){
  res <- x[, 1] + a*x[, 1]*x[, 2]
  attr(res, "grad") <- cbind(1 + a * x[, 2], a * x[, 1])
  return(res)
}

n <- 1e3
set.seed(0)
X <- matrix(runif(2*n, min = -1/2, max = 1/2), nrow = n, ncol = 2)
a <- 3
fX <- g(X, a = a)

out_1 <- out_2 <- PoincareOptimal(distr = list("unif", -1/2, 1/2),
only.values = FALSE, der = TRUE,
method = "quad")

out <- list(out_1, out_2)

# Lower bounds for X1

c2_10 <- PoincareChaosSqCoef(PoincareEigen = out, multiIndex = c(1, 0),
design = X, output = fX, outputGrad = attr(fX, "grad"),
inputIndex = 1, der = FALSE)
c2_11 <- PoincareChaosSqCoef(PoincareEigen = out, multiIndex = c(1, 1),
design = X, output = fX, outputGrad = attr(fX, "grad"),
inputIndex = 1, der = FALSE)
c2_10_der <- PoincareChaosSqCoef(PoincareEigen = out, multiIndex = c(1, 0),
design = X, output = fX, outputGrad = attr(fX, "grad"),
inputIndex = 1, der = TRUE)
c2_11_der <- PoincareChaosSqCoef(PoincareEigen = out, multiIndex = c(1, 1),
design = X, output = fX, outputGrad = attr(fX, "grad"),
inputIndex = 1, der = TRUE)
LB1 <- c(8/pi^4, c2_10, c2_10_der)
LB1tot <- LB1 + c(64/pi^8 * a^2, c2_11, c2_11_der)
LB <- cbind(LB1, LB1tot)
rownames(LB) <- c("True lower bound value",
            "Estimated, no derivatives", "Estimated, with derivatives")
colnames(LB) <- c("D1", "D1tot")
cat("True values of D1 and D1tot: ", c(1/12, 1/12 + a^2 / 144),"\n")
cat("Sample size: ", n, "\n")
cat("Lower bounds computed with the first Poincare eigenvalue:\n")
print(LB)
cat("\nN.B. Increase the sample size to see the convergence to true lower bound values.\n")

############################################################
# Flood model example (see Roustant et al., 2017, 2019)

library(evd) # Gumbel law
library(triangle) # Triangular law

# Flood model
Fcrues_full2=function(X,ans=0){
  # ans=1 gives Overflow output; ans=2 gives Cost output; ans=0 gives both
  mat=matrix(X,nrow=8);
  if (ans==0){ repone=matrix(NA,nrow(mat),2);}
  else{ repone=rep(NA,nrow(mat));}
  for (i in 1:nrow(mat)) {
    H = (mat[i,1] / (mat[i,2]*mat[i,8]*sqrt((mat[i,4] - mat[i,3])/mat[i,7])))^(0.6) ;
    if (S > 0){ Cp = 1 ;}
    else{ Cp = 0.2 + 0.8 * (1 - exp(-1000 / S^4));}
    if (mat[i,5]>8){ Cp = Cp + mat[i,5]/20 ;}
    else{ Cp = Cp + 8/20 ;}
    if (ans==0){
      repone[i,1] = S ;
      repone[i,2] = Cp ;
    }
    if (ans==1){ repone[i] = S ;}
    if (ans==2){ repone[i] = Cp ;}
  }  
  return(RES=repone)
}

# Flood model derivatives (by finite-differences)
dFcrues_full2 <- function(X, i, ans, eps){
  der = X
  X1 = X
  X1[,i] = X[,i]+eps
  der = (Fcrues_full2(X1,ans) - Fcrues_full2(X,ans))/(eps)
  return(der)
}
# Function for flood model inputs sampling

```r
EchantFcrues_full2<-function(taille){
  X = matrix(NA,taille,8)
  X[,1] = rgumbel.trunc(taille,loc=1013.0,scale=558.0,min=500,max=3000)
  X[,2] = rnorm.trunc(taille,mean=30.0,sd=8,min=15.)
  X[,3] = rtriangle(taille,a=49,b=51,c=50)
  X[,4] = rtriangle(taille,a=54,b=56,c=55)
  X[,5] = runif(taille,min=7,max=9)
  X[,6] = rtriangle(taille,a=55,b=56,c=55.5)
  X[,7] = rtriangle(taille,a=4990,b=5010,c=5000)
  X[,8] = rtriangle(taille,a=295,b=305,c=300)
  return(X)
}
```

```r
d <- 8
n <- 1e3
eps <- 1e-7 # finite-differences for derivatives
x <- EchantFcrues_full2(n)
yy <- Fcrues_full2(x, ans=2)
y <- scale(yy, center = TRUE, scale = FALSE)[,1]
dy <- NULL
for (i in 1:d) dy <- cbind(dy, dFcrues_full2(x, i, ans=2, eps))
```

```r
method <- "quad"
out_1 <- PoincareOptimal(distr = list("gumbel", 1013, 558), min=500,max=3000,
 only.values = FALSE, der = TRUE, method = method)
out_2 <- PoincareOptimal(distr = list("norm", 30, 8), min=15, max=200,
 only.values = FALSE, der = TRUE, method = method)
out_3 <- PoincareOptimal(distr = list("triangle", 49, 51, 50),
 only.values = FALSE, der = TRUE, method = method)
out_4 <- PoincareOptimal(distr = list("triangle", 54, 56, 55),
 only.values = FALSE, der = TRUE, method = method)
out_5 <- PoincareOptimal(distr = list("unif", 7, 9),
 only.values = FALSE, der = TRUE, method = method)
out_6 <- PoincareOptimal(distr = list("triangle", 55, 56, 55.5),
 only.values = FALSE, der = TRUE, method = method)
out_7 <- PoincareOptimal(distr = list("triangle", 4990, 5010, 5000),
 only.values = FALSE, der = TRUE, method = method)
out_8 <- PoincareOptimal(distr = list("triangle", 295, 305, 300),
 only.values = FALSE, der = TRUE, method = method)
out_ <- list(out_1,out_2,out_3,out_4,out_5,out_6,out_7,out_8)
```

```r
c2 <- c2der <- c2tot <- c2totder <- rep(0,d)
for (i in 1:d){
  m <- diag(1,d,d) ; m[,i] <- 1
  for (j in 1:d){
    cc <- PoincareChaosSqCoef(PoincareEigen = out_, multiIndex = m[j,],
      design = x, output = y, outputGrad = NULL,
      inputIndex = i, der = FALSE)
    c2tot[j] <- c2tot[j] + cc
    if (j == i) c2[i] <- cc
  }
}
PoincareConstant

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 \([\text{min}, \text{max}]\) interval: The optimal Poincare constant is \(\frac{(\text{max}-\text{min})^2}{\text{max}}\).
- 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 (analytical 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 logis-
tic transport". Requires argument 'dfct' and 'pfct'
optimize.interval In the transport-based case (logconcave=FALSE), a vector containing the end-
points 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 logcon-
cave=FALSE, - the non-truncated distribution name has to be passed in the 'dfct' and 'qfct' argu-
ments 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**


**See Also**

`PoincareOptimal`

**Examples**

# Exponential law (log-concave)

```r
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)

```r
PoincareConstant(dfct=dweibull,qfct=NULL,pfct=pweibull,optimize.interval=c(0, 15),shape=1,scale=1) # logistic transport approach
```

# Triangular law (log-concave)

```r
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,transport="double_exp", optimize.interval=c(-1.1)) # Logistic transport calculation
```

# Normal N(0,1) law truncated on [-1.87,+infty]

```r
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=dnorm.trunc,qfct=qnorm.trunc,pfct=pnorm.trunc,mean=0,sd=1, truncated=TRUE, min=-1.87, max=999, transport="double_exp", optimize.interval=c(-1.87,20)) # Logistic transport approach
```

# Gumbel law (log-concave)

```r
library(evd)
```
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 eigenvalues are sorted in ascending order, starting from zero. The information corresponding to the optimal constant is thus given in the second 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, der = FALSE, plot = FALSE, ...)```

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)
# Double-exponential transport approach
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=dgumbel.trunc, qfct=NULL, pfct=pgumbel.trunc, loc=0, scale=1, truncated=TRUE, min=-0.92, max=3.56, transport="double_exp", optimize.interval=c(-0.92,3.56)) # Logistic transport approach
PoincareConstant(dfct=dgumbel.trunc, qfct=qgumbel.trunc, pfct=pgumbel.trunc, loc=0, scale=1, truncated=TRUE, min=-0.92, max=3.56, optimize.interval=c(-0.92,3.56))
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 omitted.

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)

der if TRUE, compute the eigenfunction derivatives (default value is FALSE)

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 corresponding distributions (for a better numerical efficiency). In these cases, the return optimal constant and eigenvalues correspond to original distributions.

Value

PoincareOptimal returns a list containing:

opt the optimal Poincare constant

values the eigenvalues in increasing order, starting from 0. Thus, the second one is the spectral gap, equal to the inverse of the Poincare constant

vectors the values of eigenfunctions at knots

der the values of eigenfunction derivatives at knots

knots a sequence of length n formed by equally spaced real numbers in the support of the probability distribution, used for discretization

Author(s)

Olivier Roustant and Bertrand Iooss
References


O Roustant, F. Gamboa, B Iooss. Parseval inequalities and lower bounds # for variance-based sensitivity indices. 2019. hal-02140127

See Also

PoincareConstant, PoincareChaosSqCoef

Examples

# uniform on [a, b]
a <- -1 ; b <- 1
out <- PoincareOptimal(distr = list("unif", a, b))
cat("Poincare constant (theory -- estimated):", (b-a)^2/pi^2, "--", out$opt, "\n")

# truncated standard normal on [-1, 1]
# the optimal Poincare constant is then equal to 1/3,
# as -1 and 1 are consecutive roots of the 2nd Hermite polynomial X*X - 1.
out <- PoincareOptimal(distr = dnorm, min = -1, max = 1,
plot = TRUE, only.values = FALSE)
cat("Poincare constant (theory -- estimated):", 1/3, "--", out$opt, "\n")

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

# symmetric triangular [-1,1]
library(triangle)
out <- PoincareOptimal(distr = list("triangle", -1, 1, 0), min = NULL, max = NULL)
cat("Poincare constant (theory -- estimated):", 0.1729, "--", out$opt, "\n")

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

## -------------------------------
## Illustration for eigenfunctions on the uniform distribution

### (corresponds to Fourier series)

```r
b <- 1
a <- -b
out <- PoincareOptimal(distr = list("unif", a, b),
                        only.values = FALSE, der = TRUE, method = "quad")
```

# Illustration for 3 eigenvalues

```r
par(mfrow = c(3,2))
eigenNumber <- 1:3  # eigenvalue number
for (k in eigenNumber[1:3]){  # keep the 3 first ones (for graphics)
  plot(out$knots, out$vectors[, k + 1], type = "l",
       ylab = "", xlab = "Eigenvalue:",
       main = paste("Eigenfunction", k),
       col = "red", lty = "dotted")
  sgn <- sign(out$vectors[1, k + 1])
  lines(out$knots, sgn * sqrt(2) * cos(pi * k * (out$knots/(b-a) + 0.5)),
        col = "red", lty = "dotted")

  plot(out$knots, out$der[, k + 1], type = "l",
       ylab = "", xlab = "Eigenfunction derivative",
       main = paste("Eigenfunction derivative", k),
       col = "red", lty = "dotted")
  sgn <- sign(out$der[1, k + 1])
  lines(out$knots, - sgn * sqrt(2) / (b-a) * pi * k * sin(pi * k * (out$knots/(b-a) + 0.5)),
        col = "red", lty = "dotted")
}
```

# how to create a function for one eigenfunction and eigenvalue, # given N values
```r
eigenFun <- approxfun(x = out$knots, y = out$vectors[, 2])
eigenDerFun <- approxfun(x = out$knots, y = out$der[, 2])
x <- runif(n = 3, min = -1/2, max = 1/2)
eigenFun(x)
eigenDerFun(x)
```

---

**qosa**

**Quantile-oriented sensitivity analysis**

### Description

qosa implements the estimation of first-order quantile-oriented sensitivity indices as defined in Fort et al. (2016) with a kernel-based estimator of conditional probability density functions closely related to the one proposed by Maume-Deschamps and Niang (2018). qosa also supports a kernel-based estimation of Sobol first-order indices (i.e. Nadaraya-Watson).
Usage

qosa(model = NULL, X1, X2 = NULL, type = "quantile", alpha = 0.1, split.sample = 2/3, nsample = 1e4, nboot = 0, conf = 0.95, ...)

## S3 method for class 'qosa'
tell(x, y = NULL, ...)
## S3 method for class 'qosa'
print(x, ...)
## S3 method for class 'qosa'
plot(x, ylim = c(0, 1), ...)
## S3 method for class 'qosa'
ggplot(x, ylim = c(0, 1), ...)

Arguments

- **model**: a function, or a model with a predict method, defining the model to analyze.
- **X1**: a random sample of the inputs used for the estimation of conditional probability density functions. If X2 is NULL, X1 is split in two samples, with the first split.sample proportion of observations assigned to X1 and the rest to X2.
- **X2**: a random sample of the inputs used to evaluate the conditional probability density functions. If NULL, it is constructed with the last (1-split.sample) proportion of observations from X1, see above.
- **type**: a string specifying which first-order sensitivity indices must be estimated: quantile-oriented indices (type="quantile") or Sobol’ indices (type="mean").
- **alpha**: if type="quantile" the quantile level.
- **split.sample**: if X2=NULL the proportion of observations from X1 assigned to the estimation of conditional probability density functions.
- **nsample**: the number of samples from the conditional probability density functions used to estimate the conditional quantiles (if type="quantile") or the conditional means (if type="mean").
- **nboot**: the number of bootstrap replicates.
- **conf**: the confidence level for confidence intervals.
- **x**: a list of class "sobolrank" 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

Quantile-oriented sensitivity indices were defined as a special case of sensitivity indices based on contrast functions in Fort et al. (2016). The estimator used by qosa follows closely the one proposed by Maume-Deschamps & Niang (2018). The only difference is that Maume-Deschamps and Niang (2018) use the following kernel-based estimate of the conditional cumulative distribution function:

\[ \hat{F}(y|X=x) = \frac{\sum_{i=1}^{n} K_{h_{x}}(x-X_{i})1\{Y_{i} < y\}}{\sum_{i=1}^{n} K_{h_{x}}(x-X_{i})} \]
whereas we use
\[ \hat{F}(y|X = x) = \frac{\sum_{i=1}^{n} K_{h_x}(x - X_i) \int_{-\infty}^{y} K_{h_y}(t - Y_i) dt}{\sum_{i=1}^{n} K_{h_x}(x - X_i)}, \]
meaning that \(1\{Y_i < y\}\) is replaced by \(\int_{-\infty}^{y} K_{h_y}(t - Y_i) dt = \Phi\left(\frac{y - Y_i}{h_y}\right)\) where \(\Phi\) is the cumulative distribution function of the standard normal distribution (since kernel \(K\) is Gaussian). The two definitions thus coincide when \(h_y \to 0\). Our formula arises from a kernel density estimator of the joint pdf with a diagonal bandwidth. In a future version, it will be genralized to a general bandwidth matrix for improved performance.

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

- **call**: the matched call.
- **X**: a data.frame containing the design of experiments.
- **X1**: a data.frame containing the design of experiments used for the estimation of conditional probability density functions.
- **X**: a data.frame containing the design of experiments used for the evaluation of conditional probability density functions.
- **y**: a vector of model responses.
- **S**: the estimations of the Sobol’ sensitivity indices.

Author(s)
Sebastien Da Veiga

References


Examples

```r
library(ks)
library(ggplot2)
library(boot)

# Test case : difference of two exponential distributions (Fort et al. (2016))
# We use two samples with different sizes
n1 <- 5000
X1 <- data.frame(matrix(rexp(2 * n1,1), nrow = n1))
n2 <- 1000
X2 <- data.frame(matrix(rexp(2 * n2,1), nrow = n2))
```
Y1 <- X1[,1] - X1[,2]
Y2 <- X2[,1] - X2[,2]
x <- qosa(model = NULL, X1, X2, type = "quantile", alpha = 0.1)
tell(x,c(Y1,Y2))
print(x)
ggplot(x)

# Test case : difference of two exponential distributions (Fort et al. (2016))
# We use only one sample
n <- 1000 # put n=10000 for more consistency
X <- data.frame(matrix(rexp(2 * n,1), nrow = n))
Y <- X[,1] - X[,2]
x <- qosa(model = NULL, X1 = X, type = "quantile", alpha = 0.7)
tell(x,Y)
print(x)
ggplot(x)

# Test case : the Ishigami function
# We estimate first-order Sobol' indices (by specifying 'mean')
# Next lines are put in comment because too long fro CRAN tests
#n <- 5000
#nboot <- 50
#X <- data.frame(matrix(-pi+2*pi*runif(3 * n), nrow = n))
#x <- qosa(model = ishigami.fun, X1 = X, type = "mean", nboot = nboot)
#print(x)
#ggplot(x)

---

**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 of 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 split 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

Examples

```r
# 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

```r
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), ...)
## S3 method for class 'sensiFdiv'
geomplot(x, ylim = c(0, 1), ...)
```
Arguments

```r
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, `S` is a list where each element encompasses the estimations of the sensitivity indices for one of the divergence.

Author(s)

Sebastien Da Veiga, Snecma

References


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. The use of universal kernels ensures equivalence between HSIC nullity and independence of X and Y. In this case, a statistical test of independence with HSIC measure as statistic can be built. H0 is "X and Y are independent", against H1: X and Y are dependent. P-value can be computed either with asymptotic approximation (Gamma approximation), either with permutation method. See Meynaoui et al. (2019) for details. sensiHSIC can also be used to perform a Target Sensitivity Analysis (TSA). The basic idea of TSA is to measure the influence of the inputs on a critical domain of the model output, see Spagnol et al. (2019), Marrel and Chabridon (2020) for details. Aggregated HSIC indices can be readily computed for multiple outputs, but for functional outputs a first PCA step is possible and recommended, see Da Veiga (2015) for an illustration.

Usage

sensiHSIC(model = NULL, X, target = NULL, cond = NULL, kernelX = "rbf", paramX = NA, kernelY = "rbf", paramY = NA, nboot = 0, conf = 0.95, estimator.type = "V-stat", test.method = "Asymptotic", B = 5000, crit.option = list(stop.criterion = "screening", alpha = 0.05,
### Arguments

- **model**: a function, or a model with a `predict` method, defining the model to analyze.
- **X**: a matrix or a `data.frame` representing the input random sample.
- **target**: list of parameters to perform Target Sensitivity Analysis (TSA). `c`: threshold. `upper`: `TRUE` for upper threshold and `FALSE` for lower threshold. `type`: the weight function type ("indicTh", "logistic", "exp1side"). `param`: the parameter value for "logistic" and "exp1side" types.
- **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 or a list of strings specifying the reproducing kernel to be used for the output variables. Available choices are "rbf" (Gaussian), "laplace" (exponential), "dcov" (distance covariance, see details), "raquad" (rationale quadratic), "categ" (categorical kernel; Applied for TSA with "indicTh" weight function), "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). If there is only one output, a categorical kernel "categ" can also be used for (multiclass) classification problems. In this case the output must take numerical values only, each unique one corresponding to a class.
paramY  a scalar or a vector of hyperparameters 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 "decov". Kernels "linear", "ssanova1" and "ssanova2" do not involve hyperparameters. If kernelY is a combination of kernels with and without hyperparameters, paramY must have a (dummy) value for the hyperparameter-free kernels. If kernelY is equal to "categ", paramY can be equal to NA, "normal" and "weighted". Default choice is "normal", which means that we use a dirac kernel. For "weighted", the dirac kernel is weighted by the inverse of the number of occurrence of each class.

nboot  the number of bootstrap replicates

conf  the confidence level for confidence intervals

estimator.type  a string specifying the type of HSIC estimator. Two types of estimators are available, V-statistic of U-statistic. If estimator.type = "V-stat" (default value), the HSIC is estimated with a biased (but asymptotically unbiased) estimator, more practical for numerical implementation. If estimator.type = "U-stat", the unbiased estimator is used. The variance is of order o(1/n) for both estimators (n being the sample size, i.e. number of rows of X). More details in Meynaoui et al., 2019

test.method  a string specifying the method used to compute the p-value of the HSIC-based independence test. If test.method = "No" then no test performed. If test.method = "Asymptotic" (default value), an asymptotic approximation with Gamma distribution is used. If test.method = "Permutation", a permutation method based on B bootstrap samples is used to estimate the p-value in a non-asymptotic framework. Permutation-test are recommended for low sample size n. More details in Meynaoui et al., 2019. If test.method = "Seq_Permutation", an iterative permutation method is used to estimate the p-value. This approach bypasses the a-priori choice of the number of permutations (B) and the sequential estimation is stopped according to the user's objective (see crit.option for details).

B  number of bootstrap samples used in the permutation method for the estimation of P-values in independence test. B is used only if test.method is "Permutation"

crit.option  a list of parameters used only if test.method = "Seq_Permutation". stop.criterion: "ranking", "screening" or "both". alpha: significance level of the test. Bstart: initial number of bootstrap samples used for P-values estimation. Bfinal: final number of bootstrap samples. Bbatch: batch of bootstrap samples used in the iterative estimation. lo: parameter depending on the stability we want to achieve. graph: logical; if TRUE plot the sequential estimation of the pvalues.

expl.var.PCA  lower bound on the percentage of explained variance to be used in the selection of PCA components. Default is NULL, meaning that no PCA step is performed if there are multiple outputs.

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.
sensiHSIC

Details

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

\[ S_{HSIC}^i = R_{HSIC}^2 = \frac{HSIC(X_i, Y)}{\sqrt{HSIC(X_i, X_i)} \sqrt{HSIC(Y, Y)}} \]

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). The Target Sensitivity measures are defined via weight functions \( w \) which depend on \( c = \text{threshold} \). Indicator function \( 1_{c} \) and smooth relaxations are available (according to target$type\):

- if type = "indicTh" ; \( w = 1_{Y > c} \),
- if type = "logistic" ; \( w = \frac{1}{1 + \exp(-\text{param}.\frac{|Y - c|}{\sigma(Y)})} \),
- if type = "exp1side" ; \( w = \exp(-\text{max}(c - Y, 0) / \text{param}.\sigma(Y)^5) \),

where \( \sigma(Y) \) is an estimation of the standard deviation of \( Y \) and \( \text{param} = 1 \) is a parameter tuning the smoothness. The Conditional Sensitivity Analysis evaluates the influence of the inputs on the output within a critical domain only, for instance given by \( Y > c \) (more details can be found in Marrel and Chabridon (2020)).

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 normalized HSIC sensitivity indices (also denoted R2HSIC),
- HSICXY the estimation of HSIC sensitivity indices (numerator in S formula),
- Pvalue the estimation of P-value of the independence test based on HSIC statistic,
- SeqPvalue a matrix containing the sequential p-values estimation of the independence test based on HSIC statistic.

Author(s)

Sebastien Da Veiga, Amandine Marrel, Anouar Meynaoui, Reda El Amri
References


See Also

  * kde, sensiFdiv, weightTSA

Examples

```
library(ggplot2)
library(boot)

# Test case : the non-monotonic Sobol g-function
# Only one kernel is provided with default hyperparameter value

n <- 50
X <- data.frame(matrix(runif(8 * n), nrow = n))

# HSIC-based GSA and asymptotic independence test
x <- sensiHSIC(model = sobol.fun, X, kernelX = "rbf", kernelY = "rbf",
               test.method = "Asymptotic")
print(x)

# HSIC-based GSA and permutation independence test
x <- sensiHSIC(model = sobol.fun, X, kernelX = "rbf", kernelY = "rbf",
               estimator.type = "U-stat", test.method = "Permutation")
print(x)

# HSIC-based GSA and independence test with optimized number of permutations
x <- sensiHSIC(model = sobol.fun, X, kernelX = "rbf", kernelY = "rbf",
               test.method = "Seq_Permutation",
               crit.option = list(stop.criterion = "ranking", alpha = 0.05,
               Bstart = 100, Bfinal = 3000, Bbatch = 100, lo = 100,
               graph = TRUE))
```
print(x)

# Target-HSIC GSA in case of given model
x <- sensiHSIC(model = sobol.fun, X, target = list(c = 0.4, upper = TRUE,
    type = "indicTh", param = 1), kernelX = "rbf",
    kernelY = "categ", test.method = "Permutation")

# Target-HSIC GSA in case of given data
x <- sensiHSIC(model = NULL, X, target = list(c = 0.4, upper = TRUE,
    type = "indicTh", param = 1), kernelX = "rbf",
    kernelY = "categ", test.method = "Permutation")
y <- sobol.fun(X)
tell(x,y)

# Conditional-HSIC GSA in case of given model
x <- sensiHSIC(model = sobol.fun, X, cond = list(c = 0.4, upper = TRUE,
    type = "indicTh", param = 1), kernelX = "rbf", kernelY = "rbf",
    test.method = "Permutation",B=3000)

# Conditional-HSIC GSA in case of given data
x <- sensiHSIC(model = NULL, X, cond = list(c = 0.4, upper = TRUE,
    type = "indicTh", param = 1), kernelX = "rbf", kernelY = "rbf",
    test.method = "Seq_Permutation", crit.option =
    list(stop.criterion = "ranking", alpha = 0.05, Bstart = 100,
    Bfinal = 3000, Bbatch = 100, lo = 100, graph = TRUE))
y <- sobol.fun(X)
tell(x,y)

# 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)
ggplot(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)
ggplot(x)

# Example in case of given data
n <- 100
X <- data.frame(matrix(-pi+2*pi*runif(3 * n), nrow = n))
Y <- ishigami.fun(X)
x <- sensiHSIC(model = NULL, X)
tell(x,Y)
shapleyBlockEstimation

Computation of the Shapley effects in the Gaussian linear framework with an unknown block-diagonal covariance matrix

Description

shapleyBlockEstimation estimates the Shapley effects of a Gaussian linear model when the parameters are unknown and when the number of inputs is large, choosing the most likely block-diagonal structure of the covariance matrix.

Usage

shapleyBlockEstimationS(Beta, S, kappa=0, M=20, tol=10^(-6))
shapleyBlockEstimationX(X, Y, delta=NULL, M=20, tol=10^(-6))
Arguments

- **Beta**: A vector containing the (estimated) coefficients of the linear model.
- **S**: Empirical covariance matrix of the inputs. Has to be positive semi-definite matrix with same size that Beta.
- **X**: Matrix containing an i.i.d. sample of the inputs.
- **Y**: Vector containing the corresponding i.i.d. sample of the (noisy) output.
- **kappa**: The positive penalization coefficient that promotes block-diagonal matrices. It is advised to choose kappa=0 to get the largest block structure such that the maximal block size is $M$.
- **delta**: Positive number that fixes the positive penalization coefficient kappa to $1/(p n^\delta)$. It is advised to choose delta to 2/3 for a positive penalisation or delta=NULL to get the largest block structure such that the maximal block size is $M$.
- **M**: Maximal size of the estimate of the block-diagonal structure. The computation time grows exponentially with $M$.
- **tol**: A relative tolerance to detect zero singular values of Sigma.

Details

- If kappa = 0 or if delta = NULL, there is no penalization.
- It is advised to choose $M$ smaller or equal than 20. For $M$ larger or equal than 25, the computation is very long.

Value

shapleyBlockEstimationS and shapleyblockEstimationX return a list containing the following components:

- **label**: a vector containing the label of the group of each input variable.
- **S_B**: the block-diagonal estimated covariance matrix of the inputs.
- **Shapley**: a vector containing all the estimated Shapley effects.

Author(s)

Baptiste Broto, CEA LIST

References


See Also

shapleyLinearGaussian, shapleyPermEx, shapleyPermRand, shapleySubsetMc

Examples

# packages for the plots of the matrices
library(gplots)
library(graphics)

# the following function improves the plots of the matrices
sig=function(x,alpha=0.4)
{
  return(1/(1+exp(-x/alpha)))
}

# 1) we generate the parameters by groups in order
K=4 # number or groups
pk=rep(0,K)
for(k in 1:K)
{
  pk[k]=round(6+4*runif(1))
}
p=sum(pk)
Sigma_ord=matrix(0,nrow=p, ncol=p)
ind_min=0
L=5
for(k in 1:K)
{
  p_k=pk[k]
  ind=ind_min+(1:p_k)
  ind_min=ind_min+p_k
  A=2*matrix(runif(p_k*L),nrow=L,ncol=p_k)-1
  Sigma_ord[ind,ind]=t(A)%*%A + 0.2*diag(rep(1,p_k))
}

image((0:p)+0.5,(0:p)+0.5,z=sig(Sigma_ord),col=cm.colors(100), zlim=c(0,1),
  ylim=c(p+0.5,0.5), main=expression(Sigma["order"])),
  cex.main=3,ylab = "", xlab = "",axes=FALSE)
box()

Beta_ord=3*runif(p)+1
eta_ord=shapleyLinearGaussian(Beta=Beta_ord, Sigma=Sigma_ord)
barplot(eta_ord,main=expression(eta["order"])),cex.axis = 2,cex.main=3)
# 2) We sample the input variables to get an input vector more general

```r
samp = sample(1:p)
Sigma = Sigma_ord[samp, samp]
```

```r
image((0:p)+0.5,(0:p)+0.5,z=sig(Sigma),col=cm.colors(100), zlim=c(0,1),
ylim=c(p+0.5,0.5), main=expression(Sigma),
cex.main=3,ylab = "",xlab = "",axes=FALSE)
box()
```

```r
Beta = Beta_ord[samp]
eta = shapleyLinearGaussian(Beta=Beta, Sigma=Sigma)
barplot(eta,main=expression(eta),cex.axis = 2,cex.main=3)
```

# 3) We generate the observations with these parameters

```r
n = 5*p # sample size

C = chol(Sigma)
X0 = matrix(rnorm(p*n),ncol=p)
X = X0 %*% C
S = var(X) # empirical covariance matrix
image((0:p)+0.5,(0:p)+0.5,z=sig(S),col=cm.colors(100), zlim=c(0,1),
ylim=c(p+0.5,0.5), main=expression(S),
cex.main=3,ylab = "",xlab = "",axes=FALSE)
box()

beta0 = rnorm(1)
Y = X %*% as.matrix(Beta) + beta0 + 0.2 * rnorm(p)
```

# 4) We estimate the block-diagonal covariance matrix
# and the Shapley effects using the observations
# We assume that we know that the groups are smaller than 15

```r
Estim = shapleyBlockEstimationX(X,Y,delta=3/4, M=15, tol=10^(-6))
eta_hat = Estim$Shapley
S_B = Estim$S_B
image((0:p)+0.5,(0:p)+0.5,z=sig(S_B),col=cm.colors(100), zlim=c(0,1),
ylim=c(p+0.5,0.5), main=expression(S[hat(B)]),
cex.main=3,ylab = "",xlab = "",axes=FALSE)
box()
```
shapleyLinearGaussian

\begin{verbatim}
barplot(eta_hat,main=expression(hat(eta)),cex.axis = 2,cex.main=3)

sum(abs(eta_hat-eta))
\end{verbatim}

\textbf{Description}

shapleyLinearGaussian implements the computation of the Shapley effects in the linear Gaussian framework, using the linear model (without the value at zero) and the covariance matrix of the inputs. It uses the block-diagonal covariance trick of Broto et al. (2019) which allows to go through high-dimensional cases (nb of inputs > 25). It gives a warning in case of dim(block) > 25.

\textbf{Usage}

\begin{verbatim}
shapleyLinearGaussian(Beta, Sigma, tol=10^(-6))
\end{verbatim}

\textbf{Arguments}

- **Beta**: a vector containing the coefficients of the linear model (without the value at zero).
- **Sigma**: covariance matrix of the inputs. Has to be positive semi-definite matrix with same size that Beta.
- **tol**: a relative tolerance to detect zero singular values of Sigma.

\textbf{Value}

shapleyLinearGaussian returns a numeric vector containing all the Shapley effects.

\textbf{Author(s)}

Baptiste Broto

\textbf{References}


See Also

shapleyBlockEstimation, shapleyPermEx, shapleyPermRand, shapleySubsetMc

Examples

library(MASS)
library(igraph)

# First example:

p=5 #dimension
A=matrix(rnorm(p^2),nrow=p,ncol=p)
Sigma=t(A)%*%A
Beta=runif(p)
Shapley=shapleyLinearGaussian(Beta,Sigma)
plot(Shapley)

# Second Example, block-diagonal:

K=5 #number of groups
m=5 # number of variables in each group
p=K*m
Sigma=matrix(0,ncol=p,nrow=p)

for(k in 1:K)
{
  A=matrix(rnorm(m^2),nrow=m,ncol=m)
  Sigma[(m*(k-1)+1):(m*k),(m*(k-1)+1):(m*k)]=t(A)%*%A
}

# we mix the variables:
samp=sample(1:p,p)
Sigma=Sigma[samp,samp]

Beta=runif(p)
Shapley=shapleyLinearGaussian(Beta,Sigma)
plot(Shapley)

shapleyPermEx     Estimation of Shapley effects by examining all permutations of inputs (Algorithm 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) and their standard errors by examining all permutations of inputs (Song et al., 2016; Iooss and Prieur,
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**

```r
shapleyPermEx(model = NULL, Xall, Xset, d, Nv, No, Ni = 3, colnames = NULL, ...)
```

## S3 method for class 'shapleyPermEx'

```r
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), ...)
## S3 method for class 'shapleyPermEx'

ggplot(x, ylim = c(0, 1), title = NULL, ...)
```

**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`: \( Xset(n, S_j, S_{jc}, x_{jc}) \) is a function to generate a \( n \)-sample of a \( d \)-dimensional input vector corresponding to the indices in \( S_j \) conditional on the input values \( x_{jc} \) with the index set \( S_{jc} \) (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.
- `title`: a title of the plot with `ggplot()` function.
- `...`: any other arguments for `model` which are passed unchanged each time it is called.

**Details**

This function requires R package "gtools".

The default values \( Ni = 3 \) is the optimal one obtained by the theoretical analysis of Song et al., 2016.
The computations of the standard errors (and then the confidence intervals) come from Iooss and Prieur (2019). Based on the outer Monte Carlo loop (calculation of expectation of conditional variance), the variance of the Monte Carlo estimate is divided by No. The standard error is then averaged over the exact permutation loop. The confidence intervals at 95% correspond to ± 1.96 standard deviations.

**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 output mean.
- `V` the estimation of the output 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 output 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**


**See Also**

`shapleyPermRand`, `shapleyLinearGaussian`, `shapleySubsetMc`, `shapleysobol_knn`, `lmg`
Examples

#################################
# Test case : the Ishigami function (3 uniform independent inputs)
# See Iooss and Prieur (2019)

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)

library(ggplot2)
ggplot(x)

#################################
# Test case : Linear model (3 Gaussian inputs including 2 dependent)
# See Iooss and Prieur (2019)

library(ggplot2)
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)
ggplot(x)
**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 \( N_v + m \times (d - 1) \times N_o \times N_i \) model evaluations.

**Usage**

```r
shapleyPermRand(model = NULL, Xall, Xset, d, Nv, m, No = 1, Ni = 3, colnames = NULL, ...)
```

**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 \( S_j \) conditional on the input values \( x_{jc} \) with the index set \( S_{jc} \) (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.
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 output mean.
- `V` the estimation of the output 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 output 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


**See Also**

*shapleyPermEx, shapleyLinearGaussian, shapleySubsetMc, shapleysobol_knn*

**Examples**

```r
# Test case: the Ishigami function
# See Iooss and Prieur (2019)
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)

library(ggplot2)
ggplot(x)
```

```r
# Test case: Linear model (3 Gaussian inputs including 2 dependent)
# See Iooss and Prieur (2019)
library(ggplot2)
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)
```
shapleysobol_knn

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))
  }
}

m <- 1e3 # put m)1e4 for more precised results
x <- shapleyPermRand(model = modlin, Xall=Xall, Xset=Xset, d=d, Nv=1e3, m = m, No = 1, Ni = 3)
print(x)
ggplot(x)

shapleysobol_knn  
Data given Shapley effects estimation via nearest-neighbors procedure

Description

shapleysobol_knn implements the estimation of several sensitivity indices using only N model evaluations via ranking (following Gamboa et al. (2020) and Chatterjee (2019)) or nearest neighbour search (Broto et al. (2020) and Azadkia & Chatterjee (2020)). Parallelized computations are possible to accelerate the estimation process. It can be used with categorical inputs (which are transformed with one-hot encoding), dependent inputs and multiple outputs. Sensitivity indices of any group of inputs can be computed, which means that in particular (full) first-order, (independent) total Sobol indices and Shapley effects are accessible. For large sample sizes, the nearest neighbour algorithm can be significantly accelerated by using approximate nearest neighbour search. It is also possible to estimate Shapley effects with the random permutation approach of Castro et al.(2009), where all the terms are obtained with ranking or nearest neighbours.

Usage

shapleysobol_knn(model=NULL, X, method = "knn", n.knn = 2, n.limit = 2000, U = NULL, n.perm = NULL, noise = F, rescale = F, nboot = NULL, boot.level = 0.8, conf=0.95, par1=NULL, ...
## S3 method for class 'shapleysobol_knn'
tell(x, y, ...)
## S3 method for class 'shapleysobol_knn'
extract(x, ...)
## S3 method for class 'shapleysobol_knn'
print(x, ...)
## S3 method for class 'shapleysobol_knn'
plot(x, ylim = c(0,1), ...)
## S3 method for class 'shapleysobol_knn'
ggplot(x, ylim = c(0, 1), ...)
## S3 method for class 'sobol_knn'
print(x, ...)
## S3 method for class 'sobol_knn'

plot(x, ylim = c(0,1), ...)

### Arguments

- **model**: a function defining the model to analyze, taking \( X \) as an argument.
- **X**: a matrix or data frame containing the observed inputs.
- **method**: the algorithm to be used for estimation, either "rank" or "knn", see details. Default is method="knn".
- **n.knn**: the number of nearest neighbours used for estimation.
- **n.limit**: sample size limit above which approximate nearest neighbour search is activated.
- **U**: an integer equal to 0 (total Sobol indices) or 1 (first-order Sobol indices) or a list of vector indices defining the subsets of inputs whose sensitivity indices must be computed or a matrix of 0s and 1s where each row encodes a subset of inputs whose sensitivity indices must be computed (see examples). Default value is NULL, meaning that Shapley values are returned (see details).
- **n.perm**: an integer, indicating the number of random permutations used for the Shapley effects' estimation. Default is n.perm=NULL, indicating that all possible permutations are used.
- **noise**: a logical which is TRUE if the model or the output sample is noisy. See details.
- **rescale**: a logical indicating if continuous inputs must be rescaled before distance computations. If TRUE, continuous inputs are first whitened with the ZCA-cor whitenning procedure (cf. `whiten()` function in package `whitening`). If the inputs are independent, this first step will have a very limited impact. Then, the resulting whitened inputs are individually modified via a copula transform such that each input has the same scale.
- **n.boot**: the number of bootstrap resamples for the bootstrap estimate of confidence intervals. See details.
- **boot.level**: a numeric between 0 and 1 for the proportion of the bootstrap sample size.
- **conf**: the confidence level of the bootstrap confidence intervals.
- **parl**: number of cores on which to parallelize the computation. If NULL, then no parallelization is done.
- **x**: the object returned by `shapleysobol_knn`.
- **y**: a numeric univariate vector containing the observed outputs.
- **ylim**: the y-coordinate limits for plotting.
- **...**: additional arguments to be passed to `model`, or to the methods, such as graphical parameters (see `par`).

### Details

For `method="rank"`, the estimator is defined in Gamboa et al. (2020) following Chatterjee (2019). For first-order indices it is based on an input ranking (same algorithm as in `sobolrank`) while for higher orders, it uses an approximate heuristic solution of the traveling salesman problem applied
to the input sample distances (cf. TSP() function in package TSP). For method="knn", ranking and TSP are replaced by a nearest neighbour search as proposed in Broto et al. (2020) and in Azadkia & Chatterjee (2020) for a similar coefficient.

The computation is done using the subset procedure, defined in Broto, Bachoc and Depecker (2020), that is computing all the Sobol’ closed indices for all possible sub-models first, and then affecting the Shapley weights.

It is the same algorithm as sobolshap_knn with method = "knn" with a slight computational improvement (the search for weight affectations is done on much smaller matrices, stored in a list indexed by their order), and ability to perform parallel computation and bootstrap confidence interval estimates.

Since bootstrap creates ties which are not accounted for in the algorithm, confidence intervals are obtained by sampling without replacement with a proportion of the total sample size boot.level, drawn uniformly.

If the outputs are noisy, the argument noise can be used: it only has an impact on the estimation of one specific sensitivity index, namely $\frac{Var(E(Y|X_1,\ldots,X_p))}{Var(Y)}$. If there is no noise this index is equal to 1, while in the presence of noise it must be estimated.

The distance used for subsets with mixed inputs (continuous and categorical) is the Euclidean distance, thanks to a one-hot encoding of categorical inputs.

If too many cores for the machine are passed on to the par1 argument, the chosen number of cores is defaulted to the available cores minus one.

If argument U is specified, only the estimated first-order or total Sobol’ indices are returned, or the estimated closed Sobol’ indices for the selected subsets. The Shapley effects are not computed, and thus, not returned.

The extract method can be used for extracting first-order and total Sobol’ indices, after the Shapley effects have been computed. It returns a list containing both sensitivity indices.

**Value**

`shapleysobol_knn` returns a list of class "shapleysobol_knn" if U=NULL, containing the following components:

- **call**: the matched call.
- **Shap**: the estimations of the Shapley effect indices.
- **VE**: the estimations of the closed Sobol’ indices for all possible sub-models.
- **indices**: list of all subsets corresponding to the structure of VE.
- **method**: which estimation method has been used.
- **n.perm**: number of random permutations.
- **w**: the Shapley weights.
- **conf_int**: a matrix containing the estimations, biais and confidence intervals by bootstrap (if nboot>0).
- **X**: the observed covariates.
- **y**: the observed outcomes.
- **n.knn**: value of the n.knn argument.
n.limit value of the n.limit argument.
U value of the U argument.
rescale wheter the design matrix has been rescaled.
n.limit maximum number of sample before nearest-neighbor approximation.
boot.level value of the boot.level argument.
noise wheter the Shapley values must sum up to one or not.
boot logical, wheter bootstrap confidence interval estimates have been performed.
nboot value of the nboot argument.
parl value of the parl argument.
conf value of the conf argument.

shapleysobol_knn returns a list of class "sobol_knn" if U, is specified, containing the following components:
call the matched call.
Sobol the estimations of the Sobol' indices.
indices list of all subsets corresponding to the structure of VE.
method which estimation method has been used.
conf_int a matrix containing the estimations, bias and confidence intervals by bootstrap
if nboot>0. X the observed covariates.
y the observed outcomes.
U value of the U argument.
n.knn value of the n.knn argument.
rescale wheter the design matrix has been rescaled.
n.limit value of the n.limit argument.
boot.level value of the boot.level argument.
boot logical, wheter bootstrap confidence interval estimates have been performed.
nboot value of the nboot argument.
parl value of the parl argument.
conf value of the conf argument.

Author(s)
Marouane Il Idrissi, Sebastien Da Veiga
References


See Also

sobolrank, sobolshap_knn, shapleyPermEx, shapleySubsetMc, lmg, pme_knn

Examples

```r
library(parallel)
library(doParallel)
library(foreach)
library(gtools)
library(boot)
library(RANN)

###########################################################
# Linear Model with Gaussian correlated inputs
###########################################################
library(mvtnorm)
set.seed(1234)
n <- 1000
beta<-c(1,-1,0.5)
sigma<-matrix(c(1,0,0,
0,1,-0.8,
0,-0.8,1),
nrow=3,
ncol=3)
X <-rmvnorm(n, rep(0,3), sigma)
colnames(X)<-c("X1", "X2", "X3")
```
y <- X%*%beta + rnorm(n,0,2)

# Without Bootstrap confidence intervals
x<-shapleysobol_knn(model=NULL, X=X,
  n.knn=3,
  noise=TRUE)
tell(x,y)
print(x)
plot(x)

#Using the extract method to get first-order and total Sobol' indices
extract(x)

# With Bootstrap confidence intervals
x<-shapleysobol_knn(model=NULL, X=X,
  n.boot=10,
  n.knn=3,
  noise=TRUE,
  boot.level=0.7,
  conf=0.95)
tell(x,y)
print(x)
plot(x)

# Extracting Sobol' indices with Bootstrap confidence intervals
nboot <- 10 # put nboot=50 for consistency

#Total Sobol' indices
x<-shapleysobol_knn(model=NULL, X=X,
  n.boot=nboot,
  n.knn=3,
  U=0,
  noise=TRUE,
  boot.level=0.7,
  conf=0.95)
tell(x,y)
print(x)
plot(x)

#First-order Sobol' indices
x<-shapleysobol_knn(model=NULL, X=X,
  n.boot=nboot,
  n.knn=3,
  U=1,
  noise=TRUE,
  boot.level=0.7,
  conf=0.95)
tell(x,y)
print(x)
plot(x)

#Closed Sobol' indices for specific subsets (list)
x<-shapleysobol_knn(model=NULL, X=X,
    nboot=nboot,
    n.knn=3,
    U=list(c(1,2), c(1,2,3), 2),
    noise=TRUE,
    boot.level=0.7,
    conf=0.95)
tell(x,y)
print(x)
plot(x)

 yardım dediğimiz özel kümenin zorunlu olduğu devre
x<-shapleysobol_knn(model=NULL, X=X,
    nboot=nboot,
    n.knn=3,
    U=list(c(1,2), c(1,2,3), 2),
    noise=TRUE,
    boot.level=0.7,
    conf=0.95)
tell(x,y)
print(x)
plot(x)

library(ggplot2) ; ggplot(x)

# We can use the output sample generated for this estimation to compute total indices
# without additional calls to the model
x2 <- shapleysobol_knn(model = NULL, X = X, U = NULL, method = "knn", n.knn = 5)
tell(x2,x$y)
plot(x2)

library(ggplot2) ; ggplot(x2)

library(ggplot2) ; ggplot(x2)

# Test case: the Ishigami function
# Example with given data and the use of approximate nearest neighbour search
n <- 5000
X <- data.frame(matrix(-pi+2*pi*runif(3 * n), nrow = n))
Y <- ishigami.fun(X)
x <- shapleysobol_knn(model = NULL, X = X, U = NULL, method = "knn", n.knn = 5,
    n.limit = 2000)
tell(x,Y)
plot(x)

library(ggplot2) ; ggplot(x)

# Extract first-order and total Sobol indices
x1 <- extract(x) ; print(x1)
library(mvtnorm) # Multivariate Gaussian variables
library(whitening) # For scaling
modlin <- function(X) apply(X,1,sum)
  d <- 3
  n <- 10000
  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)
  X <- Xall(n)
x <- shapleysobol_knn(model = modlin, X = X, U = NULL, method = "knn", n.knn = 5,
                        rescale = TRUE, n.limit = 2000)
print(x)
plot(x)

shapleySubsetMc

Estimation of Shapley effects from data using nearest neighbors method

Description
shapleySubsetMc implements the estimation of the Shapley effects from data using some nearest neighbors method to generate according to the conditional distributions of the inputs. It can be used with categorical inputs.

Usage
shapleySubsetMc(X, Y, Ntot=NULL, Ni=3, cat=NULL, weight=NULL, discrete=NULL, noise=FALSE)
# S3 method for class 'shapleySubsetMc'
plot(x, ylim = c(0, 1), ...)

Arguments
X a matrix or a dataframe of the input sample
Y a vector of the output sample
Ntot an integer of the approximate cost wanted
Ni the number of nearest neighbours taken for each point
cat a vector giving the indices of the input categorical variables
weight a vector with the same length of cat giving the weight of each categorical variable in the product distance
discrete a vector giving the indices of the input variable that are real, and not categorical, but that can take several times the same values

noise logical. If FALSE (the default), the variable Y is a function of X

x a list of class "shapleySubsetMc" storing the state of the sensitivity study (Shapley effects, cost, names of inputs)

ylim y-coordinate plotting limits

... any other arguments for plotting

Details

If weight = NULL, all the categorical variables will have the same weight 1.

If Ntot = NULL, the nearest neighbours will be compute for all the $n(2^p - 2)$ points, where n is the length of the sample. The estimation can be very long with this parameter.

Value

shapleySubsetMc returns a list of class "shapleySubsetMc", containing:

shapley the Shapley effects estimates.

cost the real total cost of these estimates: the total number of points for which the nearest neighbours were computed.

names the labels of the input variables.

Author(s)

Baptiste Broto

References


See Also

shapleyPermEx, shapleyPermRand, shapleyLinearGaussian, sobolrank, shapleysobol_knn

Examples

# First example: the linear Gaussian framework

# we generate a covariance matrice Sigma
p <- 4 #dimension
A <- matrix(rnorm(p^2),nrow=p,ncol=p)
Sigma <- t(A)%*%A # it means t(A)%*%A
C <- chol(Sigma)
n <- 500 #sample size (put n=2000 for more consistency)
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 also allows 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, ...)

```r
Z = matrix(rnorm(p*n), nrow = n, ncol = p)
X = Z %*% C # X is a gaussian vector with zero mean and covariance Sigma
Y = rowSums(X)
Shap = shapleySubsetMc(X = X, Y = Y, Ntot = 5000)
plot(Shap)

# Second example: The Sobol model with heterogeneous inputs

p = 8 # dimension
A = matrix(rnorm(p^2), nrow = p, ncol = p)
Sigma = t(A) %*% A
C = chol(Sigma)
n = 500 # sample size (put n=5000 for more consistency)
Z = matrix(rnorm(p*n), nrow = n, ncol = p)
X = Z

# we create discrete and categorical variables
X[,1] = round(X[,1]/2)
X[,2] = X[,2] > 2
X[,4] = -2 * round(X[,4]) + 4
X[X[,6] > 0 & X[,6] < 1, 6] = 1

cat = c(1, 2) # we choose to take X1 and X2 as categorical variables (with the discrete distance)
discrete = c(4, 6) # we indicate that X4 and X6 can take several times the same value

Y = sobol.fun(X)
Ntot <- 2000 # put Ntot=20000 for more consistency
Shap = shapleySubsetMc(X = X, Y = Y, cat = cat, discrete = discrete, Ntot = Ntot, Ni = 10)
plot(Shap)
```
## 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), ...)
## S3 method for class 'sobol'
plotMultOut(x, ylim = c(0, 1), ...)
## S3 method for class 'sobol'
ggplot(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 output variables with the argument `return.var` (for example, bootstrap outputs `V.boot`, `D.boot` and `S.boot`).
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

```r
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, ...)
```

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)

library(ggplot2)

```r
ggplot(x)
```

---

**Monte Carlo Estimation of Sobol' Indices (scheme by Saltelli 2002)**
## S3 method for class 'sobol2002'
plot(x, ylim = c(0, 1), ...)
## S3 method for class 'sobol2002'
plotMultOut(x, ylim = c(0, 1), ...)
## S3 method for class 'sobol2002'
ggplot(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 output variables with the argument `return.var` (for example, bootstrap outputs `V.boot`, `S.boot` and `T.boot`).
Author(s)

Gilles Pujol

References


See Also

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

Examples

```r
# 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)

library(ggplot2)
ggplot(x)
```

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), ...)

## S3 method for class 'sobol2007'
plotMultOut(x, ylim = c(0, 1), ...)

## 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
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 output variables with the argument `return.var` (for example, bootstrap outputs `V.boot`, `S.boot` and `T.boot`).

**Author(s)**

Bertrand Iooss

**References**


**See Also**

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

**Examples**

```r
# 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)

library(ggplot2)
ggplot(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), ...)
## S3 method for class 'sobolEff'
ggplot(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


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)

library(ggplot2)
ggplot(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).
### sobolGP

**MCMethod**

A character string specifying the Monte-Carlo procedure used to estimate the Sobol indices. The available 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" obtained 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

References

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

Examples

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
#--------------------------------------#

nsim <- 10 # put nsim <- 100
nboot <- 10 # put nboot <- 100

res <- sobolGP(
  model = m,
  type="UK",
  MCmethod="sobol",
  X1,
  X2,
  nsim = nsim,
```r
conf = 0.95,
nboot = nboot,
sequential = TRUE,
candidate,
sequential.tot=FALSE,
max_iter = 1000
)

res
plot(res)

x <- ask(res)
y <- branin(x)

# The following line doesn't work (uncorrected bug:
#    unused argument in km(), passed by update(), eval(), tell.sobolGP() ??)
#res.new <- tell(res,y,x)
res.new
```

describe the 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 (altogether $2p$ indices), at a total cost of $(p+2) \times n$ model evaluations. These are called the Jansen estimators.

### Usage

```r
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, ...)
   ## S3 method for class 'soboljansen'
plotMultOut(x, ylim = c(0, 1), ...)
   ## S3 method for class 'soboljansen'
.ggplot(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.
soboljansen

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).

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 output 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

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)
library(ggplot2)
ggplot(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)
}
```r
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")
```

---

**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), ...)
## S3 method for class 'sobolmara'
plotMultOut(x, ylim = c(0, 1), ...)
## S3 method for class 'sobolmara'
ggplot(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 rearrangement 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


See Also

sobolroalhs, sobol, sobolMultOut

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)

library(ggplot2)
ggplot(x)
**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**

```r
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, ...)  
## S3 method for class 'sobolmartinez'  
ggplot(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 Xi").
S the estimations of the Sobol’ first-order indices.
T the estimations of the Sobol’ total sensitivity indices.

Users can ask more output 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


See Also

sobol, sobol2002, sobolSalt, sobol2007, soboljansen, soboltouati, 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 <- sobolmartinez(model = sobol.fun, X1, X2, nboot = 0)
print(x)
plot(x)

library(ggplot2)
ggplot(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 via plotMultOut. 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",
ubiquitous = 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 available methods are: "sobol", "sobol2002", "sobol2007", "soboljansen", "sobolmara" and "sobolGP".
ubiquitous if TRUE, 1D functional Sobol indices are computed (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 available and the tell function does not work. Aggregated Sobol’ indices can be plotted with the S3 method plot and ubiquitous Sobol’ indices can be visualized with the S3 method plotMultOut (does not work for the "sobolGP” method).
sobolMultOut

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 ubiquitous=TRUE and plot.fct=TRUE).

- **Tfct**
  - the estimations of the functional Sobol’ total sensitivity indices (if ubiquitous=TRUE and plot.fct=TRUE).

Author(s)

Bertrand Iooss

References


See Also

sobol, sobol2002, sobol2007, soboljansen, sobolmara, sobolGP

Examples

```r
# Tests on the functional toy fct 'Arctangent temporal 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)
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))

sa <- sobolMultOut(model=atantemp.fun, q=100, X1, X2,
                     MCmethod="soboljansen", ubiquitous=TRUE)
print(sa)
plot(sa)
plotMultOut(sa)

library(ggplot2)
ggplot(sa)
ggplot(sa)

---

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), ...)

## S3 method for class 'sobolowen'
ggplot(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 Xi").

S the estimations of the Sobol’ first-order indices.

T the estimations of the Sobol’ total sensitivity indices.

Users can ask more output 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


See Also

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

x <- sobolowen(model = sobol.fun, X1, X2, X3, nboot = 10) # put nboot=100
print(x)
plot(x)
library(ggplot2)
ggplot(x)

sobolrank

First-order sensitivity indices estimation via ranking

Description

sobolrank implements the estimation of all first-order indices using only N model evaluations via ranking following Gamboa et al. (2020) and inspired by Chatterjee (2019).

Usage

sobolrank(model = NULL, X, nboot = 0, conf = 0.95, nsample = round(0.8*nrow(X)), ...)
## S3 method for class 'sobolrank'
tell(x, y = NULL, ...)
## S3 method for class 'sobolrank'
print(x, ...)
## S3 method for class 'sobolrank'
plot(x, ylim = c(0, 1), ...)
## S3 method for class 'sobolrank'
ggplot(x, ylim = c(0, 1), ...)

Arguments

model a function, or a model with a predict method, defining the model to analyze.
X a random sample of the inputs.
sobolrank

nboot the number of bootstrap replicates, see details.
conf the confidence level for confidence intervals, see details.
nsample the size of the bootstrap sample, see details.
x a list of class "sobolrank" 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 sobolrank is defined in Gamboa et al. (2020). It is based on ranking the inputs as was first proposed by Chatterjee (2019) for a Cramer-Von Mises based estimator. All first-order indices can be estimated with a single sample of size N. Since bootstrap creates ties which are not accounted for in the algorithm, confidence intervals are obtained by sampling without replacement with a sample size nsample.

Value

sobolrank returns a list of class "sobolrank", 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)

Sebastien Da Veiga

References


See Also

sobol, sobol2002, sobolSalt, sobol2007, soboljansen, sobolmartinez, sobolSmthSpl, sobolEff, sobolshap_knn
Examples

# Test case: the non-monotonic Sobol g-function
# Example with a call to a numerical model
library(boot)
n <- 1000
X <- data.frame(matrix(runif(8 * n), nrow = n))
x <- sobolrank(model = sobol.fun, X = X, nboot = 100)
print(x)
library(ggplot2)
ggplot(x)
# Test case: the Ishigami function
# Example with given data
n <- 500
X <- data.frame(matrix(-pi+2*pi*runif(3 * n), nrow = n))
Y <- ishigami.fun(X)
x <- sobolrank(model = NULL, X)
tell(x,Y)
print(x)
ggplot(x)

sobolrec  Recursive estimation of Sobol' indices

Description

sobolrec implements a recursive version of the procedure introduced by Tissot & Prieur (2015) using two replicated nested designs. This function estimates either all first-order indices or all closed second-order indices at a total cost of $2 \times N$ model evaluations where $N$ is the size of each replicated nested design.

Usage

sobolrec(model=NULL, factors, layers, order, precision, method=NULL, tail=TRUE, ...)
## S3 method for class 'sobolrec'
ask(x, index, ...)
## S3 method for class 'sobolrec'
tell(x, y = NULL, index, ...)
## S3 method for class 'sobolrec'
print(x, ...)
## S3 method for class 'sobolrec'
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.
layers
If \( \text{order=1} \), a vector specifying the respective sizes of each layer (see "Details"). If \( \text{order=2} \), an integer specifying the size of all layers.

order
an integer specifying which indices to estimate: 1 for first-order indices, 2 for closed second-order indices.

precision
a vector containing:
- the target precision for the stopping criterion.
- the number of steps for the stopping criterion (must be greater than 1).

tail
a boolean specifying the method used to choose the number of levels of the orthogonal array (see "Warning messages").

method
If \( \text{order=2} \), a character specifying the method to construct the orthogonal arrays (see "Details"): 
- "al" for the algebraic method
- "ar" for the accept-reject method
Set to NULL if \( \text{order=1} \).

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

index
an integer specifying the step of the recursion

y
the model response.

ylim
y-coordinate plotting limits.

... any other arguments for \text{model} which are passed unchanged each time it is called.

Details
For first-order indices, \( \text{layers} \) is a vector:
\[(s_1, \ldots, s_m)\]
specifying the number \( m \) of layers of the nested design whose respective size are given by:
\[
\prod_{i=1}^{k-1} s_i, \quad k = 2, \ldots, m + 1
\]
For closed second-order indices, \( \text{layers} \) directly specifies the size of all layers.
For each Sobol’ index \( S \) the stopping criterion writes:
\[
| S_{l-1} - S_l | < \epsilon
\]
This criterion is tested for the last \( l_0 \) steps (including the current one). \( \epsilon \) and \( l_0 \) are respectively the target precision and the number of steps of the stopping criterion specified in \( \text{precision} \).
\text{sobolrec} uses either an algebraic or an accept-reject method to construct the orthogonal arrays for the estimation of closed second-order indices. The algebraic method is less precise than the accept-reject method but offers more steps when the number of factors is small.
\text{sobolrec} automatically assigns a uniform distribution on \([0,1]\) to each input. Transformations of distributions (between \( \text{U}[0,1] \) and the wanted distribution) have to be performed before the call to \( \text{tell()} \).
sobolrec returns a list of class "sobolrec", 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**: a list of the response used at each step.
- **V**: a list of the model variance estimated at each step.
- **S**: a list of the Sobol' indices estimated at each step.
- **steps**: the number of steps performed.
- **N**: the size of each replicated nested design.

**Warning messages**

"The value entered for layers is not the square of a prime number. It has been replaced by: "
When order=2, the value of layers must be the square of a prime power number. This warning message indicates that it was not the case and the value has been replaced depending on tail. If tail=TRUE (resp. tail=FALSE) the new value of layers is equal to the square of the prime number preceding (resp. following) the square root of layers.

"The value entered for layers is not satisfying the constraint. It has been replaced by: " the value N for layers must satisfied the constraint $N \geq (d-1)^2$ 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) $d - 1$.

**References**


**Examples**

```r
# 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 indices estimation
x <- sobolrec(model = sobol.fun, factors = 8, layers=rep(2,each=15), order=1,
              precision = c(5*10^(-2),2), method=NULL, tail=TRUE)
print(x)

# closed second-order indices estimation
x <- sobolrec(model = sobol.fun, factors = 8, layers=11^2, order=2,
```

sobolrep
generalizes the estimation of the Sobol’ sensitivity indices introduced by Tissot & Prieur (2015) using two replicated orthogonal arrays. This function estimates either

- all first-order and second-order indices at a total cost of $2 \times N$ model evaluations,
- or all first-order, second-order and total-effect indices at a total cost of $N \times (d + 2)$ model evaluations,

where $N = q^2$ and $q \geq d - 1$ is a prime number corresponding to the number of levels of each orthogonal array.

Usage

sobolrep(model = NULL, factors, N, tail=TRUE, conf=0.95, nboot=0, nbrep=1, total=FALSE, ...)
## S3 method for class 'sobolrep'
tell(x, y = NULL, ...)
## S3 method for class 'sobolrep'
print(x, ...)
## S3 method for class 'sobolrep'
plot(x, ylim = c(0,1), choice, ...)
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).
- **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.
- **nbrep**: the number of times the estimation procedure is repeated (see "Details").
- **total**: a boolean specifying whether or not total effect indices are estimated.
- **x**: a list of class "sobolrep" storing the state of the sensitivity study (parameters, data, estimates).
- **y**: the model response.
- **ylim**: y-coordinate plotting limits.
- **choice**: an integer specifying which indices to plot: 1 for first-order indices, 2 for second-order indices, 3 for total-effect indices.
- **...**: any other arguments for model which are passed unchanged each time it is called.

Details

`sobolrep` 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 performed before the call to `tell()` (see "Examples").

`nbrep` specifies the number of times the estimation procedure is repeated. Each repetition makes use of the orthogonal array structure to obtain a new set of Sobol’ indices. It is important to note that no additional model evaluations are performed (the cost of the procedure remains the same).

Value

`sobolrep` returns a list of class "sobolrep", 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 response used.
- **RP**: the matrix of permutations.
- **V**: the model variance.
- **S**: a `data.frame` containing estimations of the first-order Sobol’ indices plus confidence intervals if specified.
S2

a data.frame containing estimations of the second-order Sobol' indices plus confidence intervals if specified.

T

a data.frame containing estimations of the total-effect indices plus confidence intervals if specified.

Warning messages

"The value entered for N is not the square of a prime number. It has been replaced by: " the number of levels q of each 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: " 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) \( d-1 \).

References


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])

x <- sobolrep(model = sobol.fun, factors = 8, N = 1000, nboot=100, nbrep=1, total=FALSE)
print(x)
plot(x,choice=1)
plot(x,choice=2)

# Test case: dealing with non-uniform distributions

x <- sobolrep(model = NULL, factors = 3, N = 1000, nboot=0, nbrep=1, total=FALSE)

# 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
sobolroalhs <- function(x) {rowSums(x)}
y <- toy(x$X)
tell(x, y)
print(x)
plot(x, choice = 1)
plot(x, choice = 2)

---

**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), ...)

## S3 method for class 'sobolroalhs'
ggplot(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).
sobolroalhs

- `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 ≥ (d − 1)^2`. It has been replaced by: " when `order`= 2, the following constraint must be satisfied `N ≥ (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


See Also

dobolmara, sobolroauc, sobolMultOut

Examples

```r
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)

library(ggplot2)
ggplot(x)

# closed second-order sensitivity indices
x <- sobolroalhs(model = sobol.fun, factors = 8, N = 1000, order = 2, nboot=100)
print(x)
ggplot(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)
ggplot(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)
ggplot(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), ...)
## S3 method for class 'sobolroauc'
ggplot(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.
**sobolroauc**

- **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.
sobolroauc

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 ≥ (d − 1)^2. It has been replaced by: " when order = 2, the following constraint must be satisfied N ≥ (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


See Also

sobolroalhs, sobolmar

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)

library(ggplot2)
ggplot(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)
ggplot(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 (altogether 2\(p\) indices) at a total cost of \(n \times (p + 2)\) model evaluations; or first-order, second-order and total indices at the same time (altogether \(2p + p \times (p - 1)/2\) 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, ...)
## S3 method for class 'sobolSalt'
ggplot(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

A. Saltelli, 2002, Making best use of model evaluations to compute sensitivity indices, Computer

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)

library(ggplot2)
ggplot(x, choice=1)

---

sobolshap_knn  Flexible sensitivity analysis via ranking / nearest neighbours

Description

WARNING: DEPRECATED function: use shapleysobol_knn instead. sobolshap_knn implements the estimation of several sensitivity indices using only N model evaluations via ranking (following Gamboa et al. (2020) and Chatterjee (2019)) or nearest neighbour search (Broto et al. (2020) and Azadkia & Chatterjee (2020)). It can be used with categorical inputs (which are transformed with one-hot encoding), dependent inputs and multiple outputs. Sensitivity indices of any group of inputs can be computed, which means that in particular first-order/total Sobol indices and Shapley effects are accessible. For large sample sizes, the nearest neighbour algorithm can be significantly accelerated by using approximate nearest neighbour search. It is also possible to estimate Shapley effects with the random permutation approach of Castro et al.(2009), where all the terms are obtained with ranking or nearest neighbours.

Usage

sobolshap_knn(model = NULL, X, id.cat = NULL, U = NULL, method = "knn", n.knn = 2, 
  return.shap = FALSE, randperm = FALSE, n.perm = 1e4, 
  rescale = FALSE, n.limit = 2000, noise = FALSE, ...)

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

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

## S3 method for class 'sobolshap_knn'
extract(x, ...)

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

## S3 method for class 'sobolshap_knn'
plot(x, ylim = c(0, 1), type.multout = "lines", ...)

## S3 method for class 'sobolshap_knn'
ggplot(x, ylim = c(0, 1), type.multout = "lines", ...)

Arguments

model a function, or a model with a predict method, defining the model to analyze.
X a random sample of the inputs.
id.cat a vector with the indices of the categorical inputs.
an integer equal to 0 (total Sobol indices) or 1 (first-order Sobol indices) or
a list of vector indices defining the subsets of inputs whose sensitivity indices
must be computed or a matrix of 0s and 1s where each row encodes a subset of
inputs whose sensitivity indices must be computed (see examples) or NULL (all
possible subsets).

method
the algorithm to be used for estimation, either "rank" or "knn", see details.
n.knn
the number of nearest neighbours used for estimation if method="knn".
return.shap
a logical indicating if Shapley effects must be estimated, can only be TRUE if
U=NULL.
randperm
a logical indicating if random permutations are used to estimate Shapley effects,
only if U=NULL and return.shap=TRUE.
n.perm
the number of random permutations used for estimation if randperm=TRUE.
rescale
a logical indicating if continuous inputs must be rescaled before distance compu-
tations. If TRUE, continuous inputs are first whitened with the ZCA-cor whiten-
ing procedure (cf. whiten() function in package whitening). If the inputs are
independent, this first step will have a very limited impact. Then, the resulting
whitened inputs are individually modified via a copula transform such that each
input has the same scale.
n.limit
the sample size limit above which approximate nearest neighbour search is acti-
vated, only used if method="knn".
noise
a logical which is TRUE if the model or the output sample is noisy, see details.
x
a list of class "sobolshap_knn" storing the state of the sensitivity study (pa-
rameters, data, estimates).
y
a vector of model responses.
ylim
y-coordinate plotting limits.
type.multout
the plotting method in the case of multiple outputs, either "points" or "lines",
see examples.
...
any other arguments for model which are passed unchanged each time it is
called.

Details
For method="rank", the estimator is defined in Gamboa et al. (2020) following Chatterjee (2019).
For first-order indices it is based on an input ranking (same algorithm as in sobolrank) while for
higher orders, it uses an approximate heuristic solution of the traveling salesman problem applied
to the input sample distances (cf. TSP() function in package TSP). For method="knn", ranking and
TSP are replaced by a nearest neighbour search as proposed in Broto et al. (2020) and in Azadkia &
Chatterjee (2020) for a similar coefficient. The algorithm is the same as in shapleySubsetMc but
with an optimized implementation. In particular, the distance used for subsets with mixed inputs
(continuous and categorical) are the same but here the additional one-hot encoding of categorical
variables makes it possible to work only with Euclidean distances. Furthermore, a fast approximate
nearest neighbour search is also available, which is strongly recommended for large sample sizes.
The main difference with shapleySubsetMc is that here we use the entire N sample to compute all
indices, while in shapleySubsetMc the user can specify a total cost Ntot which performs a specific
allocation of sample sizes to the estimation of each index. In addition, the weights option is not
available here yet. If the outputs are noisy, the argument `noise` can be used: it only has an impact on the estimation of one specific sensitivity index, namely $\text{Var}(E(Y|X_1,\ldots,X_p))/\text{Var}(Y)$. If there is no noise this index is equal to 1, while in the presence of noise it must be estimated.

When `randperm=TRUE`, Shapley effects are no longer estimated by computing all the possible subsets of variables but only on subsets obtained with random permutations as proposed in Castro et al.(2009). This is useful for problems with a large number of inputs, since the number of subsets increases exponentially with dimension.

The `extract` method is useful if in a first step the Shapley effects have been computed and thus sensitivity indices for all possible subsets are available. The resulting `sobolshap_knn` object can be post-treated by `extract` to get first-order and total Sobol indices very easily.

Value

`sobolshap_knn` returns a list of class "sobolshap_knn", 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.
- `U` the subsets of inputs for which sensitivity indices have been computed.
- `S` the estimations of the Sobol sensitivity indices (see details).
- `Shap` the estimations of Shapley effects, if `return.shap` was set to TRUE.
- `order` 0 (total indices), 1 (first-order indices) or NULL. Used for plotting defaults.

Author(s)

Sebastien Da Veiga

References


See Also

`sobolrank`, `shapleysobol_knn`, `shapleySubsetMc`
Examples

# Test case: the non-monotonic Sobol g-function
# Example with a call to a numerical model
# First compute first-order indices with ranking
n <- 1000
X <- data.frame(matrix(runif(8 * n), nrow = n))
x <- sobolshap_knn(model = sobol.fun, X = X, U = 1, method = "rank")
print(x)
library(ggplot2)
ggplot(x)

# We can use the output sample generated for this estimation to compute total indices
# without additional calls to the model
x2 <- sobolshap_knn(model = NULL, X = X, U = 0, method = "knn", n.knn = 5)
tell(x2,x$y)
ggplot(x2)

# Test case: the Ishigami function
# Example with given data and the use of approximate nearest neighbour search
library(RANN)
n <- 5000
X <- data.frame(matrix(-pi+2*pi*runif(3 * n), nrow = n))
Y <- ishigami.fun(X)
x <- sobolshap_knn(model = NULL, X = X, U = NULL, method = "knn", n.knn = 5,
return.shap = TRUE, n.limit = 2000)
tell(x,Y)
library(ggplot2)
ggplot(x)

# We can also extract first-order and total Sobol indices
x1 <- extract(x)
print(x1)

# Test case : Linear model (3 Gaussian inputs including 2 dependent) with scaling
# See Iooss and Prieur (2019)
library(mvtnorm) # Multivariate Gaussian variables
library(whitening) # For scaling
modlin <- function(X) apply(X,1,sum)
d <- 3
n <- 10000
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)
X <- Xall(n)
x <- sobolshap_knn(model = modlin, X = X, U = NULL, method = "knn", n.knn = 5,
return.shap = TRUE, rescale = TRUE, n.limit = 2000)
print(x)

# Test case: functional toy fct 'Arctangent temporal function'
n <- 3000
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.

Details

WARNING: This function can give bad results for reasons that have not been yet investigated.

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


See Also

sobol, sobolEff, sobolGP

Examples

X = matrix(runif(5000), ncol = 10)
Y = sobol.fun(X)
sa = sobolSmthSpl(Y, X)
plot(sa)

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 \((1 + N^2 + (N, 2)) \times n\) 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'
ggplot(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


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)

library(ggplot2)
ggplot(x)

library(igraph)
plotFG(x)
```

**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'
ggplot(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


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)

library(ggplot2)
ggplot(x)

library(igraph)
plotFG(x)

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), ...)
## S3 method for class 'soboltouati'
ggplot(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


squaredIntEstim

See Also

sobol, sobol2002, sobolSalt, sobol2007, soboljansen, 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)

library(ggplot2)
ggplot(x)


<table>
<thead>
<tr>
<th>squaredIntEstim</th>
<th>Squared integral estimate</th>
</tr>
</thead>
</table>

Description

This function provides two estimators of a squared expectation. The first one, naive, is the square of the sample mean. It is positively biased. The second one is a U-statistics, and unbiased. The two are equivalent for large sample sizes.

Usage

squaredIntEstim(x, method = "unbiased")

Arguments

x           A vector of observations supposed to be drawn independently from a square integrable random variable
method      If "unbiased", computes the U-statistics, otherwise the square of the sample mean is computed
Details

Let $X_1, \ldots, X_n$ be i.i.d. random variables. The aim is to estimate $t = E(X_i)^2$. The naive estimator is the square of the sample mean: $T_1 = [(X_1 + \ldots + X_n)/n]^2$. It is positively biased, and the bias is equal to $s^2/n$, where $s^2 = \text{var}(X_1)$. The U-statistics estimator is the average of $X_i \cdot X_j$ over all unordered pairs $(i,j)$. Equivalently, it is equal to $T_1$ minus the (unbiased) sample variance divided by $n$.

Value

A real number, corresponding to the estimated value of the squared integral.

Author(s)

O. Roustant

References


Examples

```r
n <- 100  # sample size
nsim <- 100  # number of simulations
mu <- 0

T <- Tunb <- rep(NA, nsim)
theta <- mu^2  # E(X)^2, with X following N(mu, 1)

for (i in 1:nsim){
  x <- rnorm(n, mean = mu, sd = 1)
  T[i] <- squaredIntEstim(x, method = "biased")
  Tunb[i] <- squaredIntEstim(x, method = "unbiased")
}

par(mfrow = c(1, 1))
boxplot(cbind(T, Tunb))
abline(h = theta, col = "red")
abline(h = c(mean(T), mean(Tunb)), col = c("blue", "cyan"), lty = "dotted")
# look at the difference between median and mean
```

---

**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, logistic = FALSE, nboot = 0, conf = 0.95)
## S3 method for class 'src'
print(x, ...)
## S3 method for class 'src'
plot(x, ylim = c(-1,1), ...)
## S3 method for class 'src'
ggplot(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.
logistic logical. If TRUE, the analysis is done via a logistic regression (binomial GLM).
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).

Details

Logistic regression model (logistic = TRUE) and rank-based indices (rank = TRUE) are incompatible.

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 and Bertrand Iooss
References

B. Iooss, V. Chabridon and V. Thouvenot, Variance-based importance measures for machine learning model interpretability, Congres lambda-mu23, Saclay, France, 10-13 octobre 2022 https://hal.archives-ouvertes.fr/hal-03741384


See Also

pcc, lmg, pmvd

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)

library(ggplot2)
ggplot(x)

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, ...)

support

Support index functions: Measuring the effect of input variables over their support
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 \text{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 \text{plot}, as well as the underlying diagonal scatterplots of derivatives (S3 method \text{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.

... see 'arguments' section.

Author(s)

O. Roustant
References


See Also

S3 methods `plot` and `scatterplot`: `plot.support`

Examples

```r
# 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.
**template.replace**

**Usage**

\[
\text{template.replace}(\text{text}, \text{replacement}, \text{eval} = \text{FALSE}, \\
\quad \text{key.pattern} = \text{NULL}, \text{code.pattern} = \text{NULL})
\]

**Arguments**

- `text`: vector of character strings, the template text.
- `replacement`: the list values to replace in `text`.
- `eval`: boolean, \text{TRUE} if the code within `code.pattern` has to be evaluated, \text{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 $(\text{KEY})$.

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

Users can define custom patterns. These patterns must be perl-compatible regular expressions (see \texttt{regexpr}). The default ones are:

\[
\text{key.pattern} = "\\\$$\text{\(\text{KEY}\)}\\" \\
\text{code.pattern} = "\@\\{\texttt{\text{CODE}}\}\\"
\]

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

**Author(s)**

Gilles Pujol

**Examples**

\[
\text{txt} \leftarrow \text{c}(\"Hello $(name)!\", \"$(a) + $(b) = @\{($a)+($b)$}\", \\
\quad \text{"pi = @\{format(pi,digits=5)\}"})
\]

replacement \leftarrow \text{list}(\text{name} = \"world\", \text{a = 1, b = 2})

# 1. without code evaluation:
\text{txt.rpl1} \leftarrow \text{template.replace}(\text{txt}, \text{replacement})
\text{print(\text{txt.rpl1})}

# 2. with code evaluation:
\text{txt.rpl2} \leftarrow \text{template.replace}(\text{txt}, \text{replacement}, \text{eval} = \text{TRUE})
\text{print(\text{txt.rpl2})}
Test Models for Sensitivity Analysis

Description

These functions are standard testcase for sensitivity analysis benchmarks. For a scalar output (see Saltelli et al. 2000 and https://www.sfu.ca/~ssurjano/):

- the g-function of Sobol’ with 8 inputs, \( X \sim U[0,1] \);
- the function of Ishigami with 3 inputs, \( X \sim U[-\pi,\pi] \);
- the function of Morris with 20 inputs, \( X \sim U[0,1] \);
- the Linkletter decreasing coefficients function, \( X \sim U[0,1] \) (Linkletter et al. (2006));
- the heterdisc function with 4 inputs, \( X \sim U[0,20] \);
- the Friedman function with 5 inputs, \( X \sim U[0,1] \) (Friedman, 1991).

For functional output cases:

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

Usage

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

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 and Bertrand Iooss
References


Examples

```r
# 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)
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)
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)
}
```

truncateddistrib  Truncated distributions

Description

dnorm.trunc, pnorm.trunc, qnorm.trunc and rnorm.trunc are functions for the Truncated Normal Distribution. dgumbel.trunc, pgumbel.trunc, agumbel.trunc and rgumbel.trunc are functions for the Truncated Gumbel Distribution.
Usage

dnorm.trunc(x, mean = 0, sd = 1, min = -1e6, max = 1e6)
pnorm.trunc(q, mean = 0, sd = 1, min = -1e6, max = 1e6)
qnorm.trunc(p, mean = 0, sd = 1, min = -1e6, max = 1e6)
rnorm.trunc(n, mean = 0, sd = 1, min = -1e6, max = 1e6)
dgumbel.trunc(x, loc = 0, scale = 1, min = -1e6, max = 1e6)
pgumbel.trunc(q, loc = 0, scale = 1, min = -1e6, max = 1e6)
qgumbel.trunc(p, loc = 0, scale = 1, min = -1e6, max = 1e6)
rgumbel.trunc(n, loc = 0, scale = 1, min = -1e6, max = 1e6)

Arguments

x, q vector of quantiles
p vector of probabilities
n number of observations
mean, sd means and standard deviation parameters
loc, scale location and scale parameters
min vector of minimal bound values
max vector of maximal bound values

Details

See dnorm for details on the Normal distribution. The Gumbel distribution comes from the evd package. See dgumbel for details on the Gumbel distribution.

Value

dnorm.trunc and dgumbel.trunc give the density, pnorm and pgumbel.trunc give the distribution function, qnorm and qgumbel.trunc give the quantile function, rnorm and rgumbel.trunc generate random deviates.

Author(s)

Gilles Pujol and Bertrand Iooss

Weight-function to transform an output variable in order to perform Target Sensitivity Analysis (TSA)

Description

Transformation function of one variable (vector sample)

Usage

weightTSA(Y, c, upper = TRUE, type="indicTh", param=1)
Arguments

- **Y**: The output vector
- **c**: The threshold
- **upper**: TRUE for upper threshold and FALSE for lower threshold
- **type**: The weight function type ("indicTh", "zeroTh", "logistic", "exp1side"):
  - indicTh: indicator-thresholding
  - zeroTh: zero-thresholding (keeps the variable value above (upper=TRUE case) or below the threshold)
  - logistic: logistic transformation at the threshold
  - exp1side: exponential transformation above (upper=TRUE case) or below the threshold (see Raguet and Marrel)
- **param**: The parameter value for "logistic" and "exp1side" types

Details

The weight functions depend on a threshold \( c \) and/or a smooth relaxation. These functions are defined as follows:

- if type = "indicTh": \( w = 1_{Y>c} \) (upper threshold) and \( w = 1_{Y<c} \) (lower threshold),
- if type = "zeroTh": \( w = Y1_{Y>c} \) (upper threshold) and \( w = Y1_{Y<c} \) (lower threshold),
- if type = "logistic":
  \[
  w = \left[ 1 + \exp\left( -\frac{\text{param} \cdot (Y - c)}{|c|} \right) \right]^{-1}
  \]
  (upper threshold) and
  \[
  w = \left[ 1 + \exp\left( -\frac{\text{param} \cdot (c - Y)}{|c|} \right) \right]^{-1}
  \]
  (lower threshold),
- if type = "exp1side":
  \[
  w = \left[ 1 + \exp\left( -\frac{\max(c - Y, 0)}{\text{param} \cdot \sigma(Y)} \right) \right]
  \]
  (upper threshold) and
  \[
  w = \left[ 1 + \exp\left( -\frac{\max(Y - c, 0)}{\text{param} \cdot \sigma(Y)} \right) \right]
  \]
  (lower threshold), where \( \sigma(Y) \) is an estimation of the standard deviation of \( Y \) and \( \text{param} = 1 \) is a parameter tuning the smoothness.

Value

The vector sample of the transformed variable

Author(s)

B. Iooss
References

H. Raguet and A. Marrel, *Target and conditional sensitivity analysis with emphasis on dependence measures*, Preprint, https://hal.archives-ouvertes.fr/hal-01694129


Examples

```r
n <- 100  # sample size
a <- 1.5
Y <- rnorm(n)
Yt <- weightTSA(Y, a)
```
Index

* IO
  template.replace, 160
* design
delsa, 9
discrepancyCriteria_cplus, 11
fast99, 13
morris, 19
sb, 72
shapleyPermEx, 87
shapleyPermRand, 91
sobol, 103
sobol2002, 105
sobol2007, 107
sobolEff, 110
soboljansen, 116
sobolmara, 119
sobolmartinez, 121
sobolMultOut, 124
sobolowen, 126
sobolroalhs, 136
sobolroauc, 139
sobolSalt, 142
sobolTIIlo, 149
sobolTIIpf, 151
soboltouati, 153
* methods
decoupling, 8
* misc
testmodels, 162
discrepancyCriteria_cplus, 11
fast99, 4
src, 156
* proportional values
  pme_knn, 50
* regression
  pcc, 27
  src, 156
* shapley
  lmg, 15
  pmvd, 55
  shapleysobol_knn, 94
* utilities
  parameterSets, 26
  addelman_const, 5, 6, 7
  ask (decoupling), 8
  ask.sb (sb), 72
  ask.sobolGP (sobolGP), 112
  ask.sobolrec (sobolrec), 130
  atantemp.fun (testmodels), 162
  campbell1D.fun (testmodels), 161
decoupling, 6, 8
delsa, 3, 6, 9, 26
dgumbel.trunc (truncateddistrib), 163
discrepancyCriteria_cplus, 5, 6, 11, 19
dnorm.trunc (truncateddistrib), 163
extract (decoupling), 8
extract.shapleysobol_knn
  (shapleysobol_knn), 94
extract.sobolshap_knn (sobolshap_knn), 144
fast99, 4, 13, 105
friedman.fun (testmodels), 162
ggplot.pcc (pcc), 27
ggplot.pme_knn (pme_knn), 50
ggplot.qosa (qosa), 69
ggplot.sensifdiv (sensifdiv), 74
ggplot.sensiHSIC (sensiHSIC), 76
ggplot.shapleyPermEx (shapleyPermEx), 87
ggplot.shapleyPermRand
  (shapleyPermRand), 91
  shapleysobol_knn (shapleysobol_knn), 94
  sobol, 103
ggplot.sobol2002 (sobol2002), 105
INDEX

ggplot.sobol2007 (sobol2007), 107

plot.pcc (pcc), 27

plot.pme_knn (pme_knn), 50

plot.pmvd (pmvd), 55

plot.qosa (qosa), 69

plot.sb (sb), 72

plot.sensifdiv (sensifdiv), 74

plot.sensiHSIC (sensiHSIC), 76

plot.shapleyPermEx (shapleyPermEx), 87

plot.shapleyPermRand (shapleyPermRand), 91

plot.shapleysobol_knn (shapleysobol_knn), 94

plot.shapleySubsetMc (shapleySubsetMc), 101

plot.sobol (sobol), 103

plot.sobol2002 (sobol2002), 105

plot.sobol2007 (sobol2007), 107

plot.sobol_knn (shapleysobol_knn), 94

plot.sobolEff (sobolEff), 110

plot.sobolGP (sobolGP), 112

plot.soboljansen (soboljansen), 116

plot.soboljansen (soboljansen), 119

plot.sobolmartinez (sobolmartinez), 121

plot.sobolMultOut (sobolMultOut), 124

plot.sobolowen (sobolowen), 126

plot.sobolrank (sobolrank), 128

plot.sobolroalhs (sobolroalhs), 136

plot.sobolroauc (sobolroauc), 139

plot.sobolsalt (sobolsalt), 142

plot.sobolshap_knn (sobolshap_knn), 144

plot.sobolTIIlo (sobolTIIlo), 147

plot.sobolTIIpf (sobolTIIpf), 151

plot.soboltouati (soboltouati), 153

plot.src (src), 156

heterdisc.fun (testmodels), 162

identify, 21

ishigami.fun (testmodels), 162

kde, 76, 80

knn, 112, 115

linkletter.fun (testmodels), 162

lmg, 3, 6, 15, 28, 53, 57, 89, 98, 158

maximin.cplus, 5, 6, 12, 18

morris, 3, 8, 19, 25

morris.fun (testmodels), 162

morrisMultiOut, 5, 6, 23, 24

par, 10

parameterSets, 6, 9, 10, 26

pcc, 3, 17, 27, 57, 158

pgumbel.trunc (truncateddistr), 163

PLI, 5, 6, 29, 35, 39, 43, 47

PLIquantile, 5, 6, 30, 33, 39, 43, 47

PLIquantile_multivar, 5, 6, 30, 35, 37, 47

PLIsuperquantile, 5, 6, 30, 35, 39, 41, 47

PLIsuperquantile_multivar, 5, 6, 30, 35, 39, 43, 45

plot (plot.support), 49

plot.delsa (delsa), 9

plot.fast99 (fast99), 13

plot.lmg (lmg), 15

plot.morris (morris), 19

plotMultOut.sobol2007 (sobol2007), 107

plotMultOut.sobolEff (sobolEff), 110

plotMultOut.soboljansen (soboljansen), 116

plotMultOut.sobolmartinez (sobolmartinez), 121

plotMultOut.sobolMultOut (sobolMultOut), 124

plotMultOut.sobolowen (sobolowen), 126

plotMultOut.sobolrank (sobolrank), 128

plotMultOut.sobolroalhs (sobolroalhs), 136

plotMultOut.sobolroauc (sobolroauc), 139

plotMultOut.sobolsalt (sobolsalt), 142

plotMultOut.sobolshap_knn (sobolshap_knn), 144

plotMultOut.sobolTIIlo (sobolTIIlo), 147

plotMultOut.sobolTIIpf (sobolTIIpf), 151

plotMultOut.soboltouati (soboltouati), 153

plotMultOut.sobolrep (sobolrep), 133

plotMultOut.sobolTIIlo (sobolTIIlo), 147

plotMultOut.sobolTIIpf (sobolTIIpf), 151

plotMultOut.soboljansen (soboljansen), 116

plotMultOut.sobolmartinez (sobolmartinez), 121

plotMultOut.sobolMultOut (sobolMultOut), 124

plotMultOut.sobolowen (sobolowen), 126

plotMultOut.sobolrank (sobolrank), 128

plotMultOut.sobolrep (sobolrep), 133

plotMultOut.sobolroalhs (sobolroalhs), 136

plotMultOut.sobolroauc (sobolroauc), 139

plotMultOut.sobolsalt (sobolsalt), 142

plotMultOut.sobolshap_knn (sobolshap_knn), 144

plotMultOut.sobolTIIlo (sobolTIIlo), 147

plotMultOut.sobolTIIpf (sobolTIIpf), 151
INDEX

pmdv, 3, 6, 17, 28, 53, 55, 158
pnorm.trunc (truncateddistribution), 163
PoincareChaosSqrCoef, 3, 6, 59, 68
PoincareConstant, 3, 6, 63, 68
print.delsa (delsa), 9
print.fast99 (fast99), 13
print.lmg (lmg), 15
print.morris (morris), 19
print.pcc (pcc), 27
print.pme_knn (pme_knn), 50
print.pmvd (pmvd), 55
print.qosa (qosa), 69
print.qs (qs), 72
print.sensiFdiv (sensiFdiv), 74
print.sensiHSIC (sensiHSIC), 76
print.shapleyPermEx (shapleyPermEx), 87
print.shapleyPermRand (shapleyPermRand), 91
print.shapleysobol_knn (shapleysobol_knn), 94
print.sobol (sobol), 103
print.sobol2002 (sobol2002), 105
print.sobol2007 (sobol2007), 107
print.sobol_knn (shapleysobol_knn), 94
print.sobolEff (sobolEff), 110
print.sobolGP (sobolGP), 112
print.soboljansen (soboljansen), 116
print.sobolmara (sobolmara), 119
print.sobolmartinez (sobolmartinez), 121
print.sobolMultOut (sobolMultOut), 124
print.sobolowen (sobolowen), 126
print.sobolrank (sobolrank), 128
print.sobolrec (sobolrec), 130
print.sobolrep (sobolrep), 133
print.sobolroalhs (sobolroalhs), 136
print.sobolroauc (sobolroauc), 139
print.sobolSalt (sobolSalt), 142
print.sobolshap_knn (sobolshap_knn), 144
print.sobolTI1lo (sobolTI1lo), 149
print.sobolTI1pf (sobolTI1pf), 151
print.soboltouati (soboltouati), 153
print.src (src), 156
qgumbel.trunc (truncateddistribution), 163
qnorm.trunc (truncateddistribution), 163
qosa, 5, 69
regexpr, 161
rgumbel.trunc (truncateddistribution), 163
rnorm.trunc (truncateddistribution), 163
sb, 3, 8, 72
scatterplot (plot.support), 49
sensiFdiv, 5, 74, 80
sensiHSIC, 5, 6, 76, 76
sensitivity, 10
sensitivity (sensitivity-package), 3
sensitivity-package, 3
shapleyBlockEstimation, 4, 6, 82, 87
shapleyBlockEstimation$ (shapleyBlockEstimation), 82
shapleyBlockEstimationX (shapleyBlockEstimation), 82
shapleyLinearGaussian, 4, 6, 84, 86, 89, 93, 102
shapleyPermEx, 4, 6, 17, 53, 84, 87, 87, 93, 98, 102
shapleyPermRand, 5, 6, 84, 87, 89, 91, 102
shapleysobol_knn, 5, 6, 17, 53, 89, 93, 94, 102, 146
shapleySubsetMc, 5, 6, 53, 84, 87, 89, 93, 98, 101, 146
sobol, 4, 103, 107, 109, 111, 115, 118, 120, 123, 125, 129, 143, 149, 155
sobol.fun (testmodels), 162
sobol2002, 4, 105, 105, 109, 111, 115, 118, 123, 125, 129, 155
sobol2007, 4, 105, 107, 107, 111, 115, 118, 123, 125, 129, 143, 155
sobolEff, 4, 105, 107, 109, 110, 115, 118, 127, 129, 143, 149
sobolGP, 4, 6, 105, 107, 112, 125, 149
soboljansen, 4, 105, 107, 109, 111, 115, 116, 123, 125, 129, 143, 155
sobolmara, 4, 105, 107, 109, 118, 125, 138, 141
sobolmartinez, 4, 105, 107, 109, 111, 115, 118, 121, 127, 129, 143, 155
sobolMultOut, 5, 105, 107, 109, 115, 118, 120, 123, 124, 138
sobolowen, 4, 6, 126
sobolrank, 4, 53, 98, 102, 128, 146
sobolrec, 4, 6, 130
sobolrep, 4, 6, 133
sobolroalhs, 4, 6, 105, 120, 136, 141
sobolroauc, 4, 6, 138, 139
sobolSalt, 4, 6, 105, 107, 109, 111, 118, 123, weightTSA, 5, 80, 164
sobolshap_knn, 5, 98, 129, 144
sobolSmthSpl, 3, 6, 105, 111, 129, 148
sobolTIIlo, 4, 6, 149, 152
sobolTIIpf, 4, 6, 150, 151
soboltouati, 4, 6, 123, 153
squaredIntEstim, 5, 6, 59, 155
src, 3, 8, 17, 28, 57, 156
support, 5, 6, 50, 158
tell (decoupling), 8
tell.delsa (delsa), 9
tell.fast99 (fast99), 13
tell.morris (morris), 19
tell.morrisMultOut (morrisMultOut), 24
tell.pme_knn (pme_knn), 50
tell.qosa (qosa), 69
tell.sb (sb), 72
tell.sensiFdiv (sensiFdiv), 74
tell.sensiHSIC (sensiHSIC), 76
tell.shapleyPermEx (shapleyPermEx), 87
tell.shapleyPermRand (shapleyPermRand), 91
tell.shapleysobol_knn (shapleysobol_knn), 94
tell.sobol (sobol), 103
tell.sobol2002 (sobol2002), 105
tell.sobol2007 (sobol2007), 107
tell.sobolEff (sobolEff), 110
tell.sobolGP (sobolGP), 112
tell.soboljansen (soboljansen), 116
tell.sobolmara (sobolmara), 119
tell.sobolmartinez (sobolmartinez), 121
tell.sobolowen (sobolowen), 126
tell.sobolrank (sobolrank), 128
tell.sobolrec (sobolrec), 130
tell.sobolrep (sobolrep), 133
tell.sobolroalhs (sobolroalhs), 136
tell.sobolroauc (sobolroauc), 139
tell.sobolsalt (sobolsalt), 142
tell.sobolshap_knn (sobolshap_knn), 144
tell.sobolTIIlo (sobolTIIlo), 149
tell.sobolTIIpf (sobolTIIpf), 151
tell.soboltouati (soboltouati), 153
template.replace, 5, 160
testmodels, 5, 162
truncateddistrib, 5, 163