Package ‘sensitivity’

August 26, 2014

Version 1.9

Date 2014-08-25

Title Sensitivity Analysis

Author Gilles Pujol, Bertrand Iooss, Alexandre Janon with contributions from Paul Lemaitre, Laurent Gilquin, Loic Le Gratiet, Taieb Touati, Bernardo Ramos, Jana Fruth and Sebastien Da Veiga

Maintainer Bertrand Iooss `<biooss@yahoo.fr>`

Depends R (>= 3.0.0)

Imports boot

Suggests rgl, evd, numbers, DiceKriging, ks, fanovaGraph

Description A collection of functions for factor screening, global sensitivity analysis and reliability sensitivity analysis of model output.

License GPL-2

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-08-26 09:25:25

R topics documented:

`sensitivity-package` *Sensitivity Analysis*

Description

Methods and functions for global sensitivity analysis.

Details

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

- Linear regression coefficients: SRC and SRRC (`src`), PCC and PRCC (`pcc`).
- Bettonvil's sequential bifurcations (Bettonvil and Kleijnen, 1996) (`sb`).
- Morris's "OAT" elementary effects screening method (`morris`).
- Poincare constants for Derivative-based Global Sensitivity Measures (DGSM) (Roustant et al., 2014) (`PoincareConstant`).
- Variance-based sensitivity indices (Sobol' indices):
  - Monte Carlo estimation of Sobol' indices:
    - 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 and total indices with a reduced cost (`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`),
    - Janon-Monod's scheme (Monod et al., 2006; Janon et al., 2013) to compute first order indices with optimal asymptotic variance (`sobolEff`),

    - ∗ Mara's scheme (Mara and Joseph, 2008) to compute first order indices with a cost independent of the dimension, via a unique-matrix permutations (`sobolmara`),
    - ∗ Owen's scheme (Owen, 2013) to compute first order and total indices using improved formulas (via 3 input independent matrices) for small indices (`sobolowen`),
  - Estimation of the Sobol' first and second order indices using replicated orthogonal array-based Latin hypecube sample (Tissot and Prieur, 2012) (`sobolroalhs`),
  - 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`).

Sensitivity Indices based on Csiszar f-divergence (`sensiFdiv`) (particular cases: Borgonovo's indices and mutual-information based indices) and Hilbert-Schmidt Independence Criterion (`sensiHSIC`) of Da Veiga et al., 2014. Reliability sensitivity analysis by the Density Modification Based Reliability Sensitivity Indices (`DMBRSI`) of Lemaitre et al., 2014.

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

Model managing

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

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

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

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

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

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

Author(s)

Gilles Pujol, Bertrand Iooss, Alexandre Janon with contributions from Paul Lemaitre for the `DMBRSI` function, Laurent Gilquin for the `sobolroalhs` function, Loic le Gratiet for the `sobolGP` function, Taieb Touati and Bernardo Ramos fo the `sobolowen` function, Jana Fruth for the `PoincareConstant` function and Sebastien Da veiga for the `sensiFdiv` and `sensiHSIC` functions.

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

References

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

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

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

`decoupling` *Decoupling Simulations and Estimations*

Description

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

Usage

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

Arguments

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

Details

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

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

Value

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

`ask` returns the set of data to simulate.

Author(s)

Gilles Pujol

Examples

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

`DMBRSI` *Density Modification Based Reliability Sensitivity Indices (DMBRSI)*

Description

DMBSRI computes the Density Modification Based Reliability Sensitivity Indices (DMBRSI), which are sensitivity indices related to a probability of exceedence of a model output (i.e. a failure probability), estimated by a Monte Carlo method. See Lemaitre et al. (2014).

Usage

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

Arguments

`failurepoints` a matrix of failure points coordinates, one column per variable.

`failureprobabilityhat`
the estimation of failure probability P through rough Monte Carlo method.

`samplesize` the size of the sample used to estimate P. One must have Pchap=dim(failurepoints)[1]/samplesize

`deltasvector` a vector containing the values of delta for which the indices will be computed.

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

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

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

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

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

- If it is set at TRUE, the mean perturbation will be the same for all the variables.

- If not, the mean perturbation will be new_mean = mean+sigma*delta where mean, sigma are parameters defined in InputDistributions and delta is a value of deltasvector.

Value

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

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

Author(s)

Paul Lemaitre

References

P. Lemaitre, E. Sergienko, A. Arnaud, N. Bousquet, F. Gamboa and B. Iooss. *Density modification based reliability sensitivity analysis*, Journal of Statistical Computation and Simulation, in press, 2014. http://hal.archives-ouvertes.fr/hal-00737978

Examples

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

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

# Monte Carlo sampling to obtain failure points

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

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

# sensitivity indices with perturbation of the mean

v_delta = seq(-3,3,1/20)
Toto = DMBRSI(failurepoints=ptsdef,failureprobabilityhat=pdefchap,samplesize=N,
deltasvector=v_delta,InputDistributions=distributionIshigami,type="MOY",
samedelta=TRUE)
```

```
BIshm = Toto[[1]]
SIshm = Toto[[2]]

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

# sensitivity indices with perturbation of the variance

v_delta = seq(1,5,1/4) # user parameter. (the true variance is 3.29)
Toto = DMBRSI(failurepoints=ptsdef,failureprobabilityhat=pdefchap,samplesize=N,
deltasvector=v_delta,InputDistributions=distributionIshigami,type="VAR",
samedelta=TRUE)
BIshv=Toto[[1]]
SIshv=Toto[[2]]

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

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

fast99 *Extended Fourier Amplitude Sensitivity Test*

Description

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

Usage

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

Arguments

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

Details

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

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

Value

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

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

Author(s)

Gilles Pujol

References

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

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

Examples

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

`morris` *Morris's Elementary Effects Screening Method*

Description

`morris` implements the Morris's elementary effects screening method (Morris 1992). This method, based on design of experiments, allows to identify the few important factors at a cost of $r \times (p+1)$ simulations (where p is the number of factors). This implementation includes some improvements of the original method: space-filling optimization of the design (Campolongo et al. 2007) and simplex-based design (Pujol 2009).

Usage

```
morris(model = NULL, factors, r, design, binf = 0, bsup = 1,
       scale = FALSE, ...)
## 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, ...)
## S3 method for class 'morris'
plot3d(x, alpha = c(0.2, 0), sphere.size = 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.
`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.). 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 1992), 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 2008), 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 before computing the elementary effects so that all factors vary within the range [0,1].
`x`	a list of class `"morris"` storing the state of the screening study (parameters, data, estimates).
`y`	a vector of model responses.
`identify`	logical. If `TRUE`, the user selects with the mouse the factors to label on the (μ^*, σ) graph (see `identify`).
`...`	any other arguments for `model` which are passed unchanged each time it is called.
`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.

Details

`plot2d` draws the (μ^*, σ) graph.

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

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`	a vector of model responses.
`ee`	a $r \times p$ matrix of elementary effects for all the factors.

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

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

Contrary to earlier versions of the function, the scaling of factors isn't forced by default. Although, 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

References

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

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

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

Examples

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

---

pcc *Partial Correlation Coefficients*

Description

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

Usage

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

Arguments

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

Value

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

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

Author(s)

Gilles Pujol

References

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

See Also

`src`

Examples

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

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

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

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

Description

A DGSM is the product between a Poincare Constant (Roustant et al., 2014) and the integral (over the space domain of the input variables) of the squared derivatives of the model output with respect to one input variable. The DGSM is a maximal bound of the total Sobol' index corresponding to the same input (Lamboni et al., 2013).

This DGSM depends on the type of probability distribution of the input variable. In the particular case of log-concave distribution, analytical formulas are available 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 non-truncated distributions (including the non log-concave case), the Poincare constant is computed via a relatively simple optimization process (Lamboni et al., 2013).

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

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

Usage

```
PoincareConstant(densityfct=dnorm, qfct=qnorm, cdfct,
                 truncated=FALSE, min=0, max=1,
                 logconcave=TRUE, optimize.interval=c(-100, 100), ...)
```

Arguments

`densityfct`	the probability density function of the input variable
`qfct`	the quantile function of the input variable
`cdfct`	the distribution function of the input variable
`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
`logconcave`	logical value: TRUE (default value) for a log-concave distribution
`optimize.interval`	In the non-log concave case, a vector containing the end-points of the interval to be searched for the maximum of the function to be optimized
`...`	additional arguments

Value

PoincareConstant returns the value of the Poincare constant.

Author(s)

Jana Fruth and Bertrand Iooss

References

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

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

Examples

```
# Exponential law (log-concave)
PoincareConstant(dexp,qexp,rate=1)

# Weibull law (non log-concave)
PoincareConstant(dweibull,cdfct=pweibull, logconcave=FALSE,
optimize.interval=c(0, 15), shape=1, scale=1)

## Not run:
# Triangular law (log-concave)
library(triangle)
PoincareConstant(dtriangle, qtriangle, a=49, b=51, c=50)

# Truncated Gumbel law (log-concave)
library(evd)
PoincareConstant(dgumbel, qgumbel, pgumbel, truncated=TRUE,
min=500, max=3000, loc=1013.0, scale=558.0)

## End(Not run)
```

---

sb *Sequential Bifurcations*

Description

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

Usage

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

Arguments

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

Details

The model without interaction is

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

while the model with interactions is

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

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

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

Value

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

`i`	the vector of bifurcations.

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

The groups effects can be displayed with the `print` method.

Author(s)

Gilles Pujol

References

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

Examples

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

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

`sensiFdiv` *Sensitivity Indices based on Csiszar f-divergence*

Description

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

Usage

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

Arguments

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

Details

Some of the Csiszar f-divergences produce sensitivity indices that have already been studied in the context of sensitivity analysis. In particular, "TV" leads to the importance measure proposed by Borgonovo (2007) (up to a constant), "KL" corresponds to the mutual information (Krzykacz-Hausmann 2001) and "Chi2" produces the squared-loss mutual information. See Da Veiga (2014) for details.

Value

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

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

Author(s)

Sebastien Da Veiga, Snecma

References

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

Da Veiga S. (2014), *Global sensitivity analysis with dependence measures*, Journal of Statistical Computation and Simulation, in press. http://hal.archives-ouvertes.fr/hal-00903283

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

See Also

`kde`, `sensiHSIC`

Examples

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

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

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

## End(Not run)
```

---

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

Description

`sensiHSIC` conducts a sensitivity analysis where the impact of an input variable is defined in terms of the distance between the input/output joint probability distribution and the product of their marginals when they are embedded in a Reproducing Kernel Hilbert Space (RKHS). This distance corresponds to the Hilbert-Schmidt Independence Criterion (HSIC) proposed by Gretton et al. (2005) and serves as a dependence measure between random variables, see Da Veiga (2014) for an illustration in the context of sensitivity analysis.

Usage

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

Arguments

`model`	a function, or a model with a `predict` method, defining the model to analyze.
`X`	a matrix or `data.frame` representing the input random sample.
`kernelX`	a string or a list of strings specifying the reproducing kernel to be used for the input variables. If only one kernel is provided, it is used for all input variables. Available choices are "rbf" (Gaussian), "laplace" (exponential), "dcov" (distance covariance, see details), "raquad" (rationale quadratic), "invmultiquad" (inverse multiquadratic), "linear" (Euclidean scalar product), "matern3" (Matern 3/2), "matern5" (Matern 5/2), "ssanova1" (kernel of Sobolev space of order 1) and "ssanova2" (kernel of Sobolev space of order 2).
`paramX`	a scalar or a vector of hyperparameters to be used in the input variable kernels. If only one scalar is provided, it is replicated for all input variables. By default `paramX` is equal to the standard deviation of the input variable for "rbf", "laplace", "raquad", "invmultiquad", "matern3" and "matern5" and to `1` for "dcov". Kernels "linear", "ssanova1" and "ssanova2" do not involve hyperparameters. If `kernelX` is a combination of kernels with and without hyperparameters, `paramX` must have a (dummy) value for the hyperparameter-free kernels, see examples below.
`kernelY`	a string specifying the reproducing kernel to be used for the output variable. Available choices are "rbf" (Gaussian), "laplace" (exponential), "dcov" (distance covariance, see details), "raquad" (rationale quadratic), "invmultiquad" (inverse multiquadratic), "linear" (Euclidean scalar product), "matern3" (Matern 3/2), "matern5" (Matern 5/2), "ssanova1" (kernel of Sobolev space of order 1) and "ssanova2" (kernel of Sobolev space of order 2).
`paramY`	a scalar to be used in the output variable kernel. By default `paramY` is equal to the standard deviation of the output variable for "rbf", "laplace", "raquad", "invmultiquad", "matern3" and "matern5" and to `1` for "dcov". Kernels "linear", "ssanova1" and "ssanova2" do not involve hyperparameters.
`nboot`	the number of bootstrap replicates
`conf`	the confidence level for confidence intervals.
`x`	a list of class `"sensiHSIC"` storing the state of the sensitivity study (parameters, data, estimates).
`y`	a vector of model responses.
`ylim`	y-coordinate plotting limits.

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

Details

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

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

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

Value

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

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

Author(s)

Sebastien Da Veiga, Snecma

References

Da Veiga S. (2014), *Global sensitivity analysis with dependence measures*, Journal of Statistical Computation and Simulation, in press. http://hal.archives-ouvertes.fr/hal-00903283

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

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

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

See Also

`kde`, `sensiFdiv`

Examples

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

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

  # A combination of kernels is given and a dummy value is passed for
  # the first hyperparameter
  x <- sensiHSIC(model = ishigami.fun, X, kernelX = c("ssanova1","matern3","dcov"),
                 paramX = c(1,2,1), kernelY = "ssanova1")
  print(x)

## End(Not run)
```

---

sobol *Monte Carlo Estimation of Sobol' Indices*

Description

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

Usage

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

Arguments

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

Value

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

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

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

Author(s)

Gilles Pujol

References

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

See Also

sobol2002, sobol2007, soboljansen, sobolEff, sobolmara, sobolroalhs, fast99, sobolGP

Examples

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

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

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

---

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

Description

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

Usage

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

Arguments

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

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

Value

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

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

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

Author(s)

Gilles Pujol

References

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

See Also

`sobol`, `sobol2007`, `soboljansen`, `sobolEff`, `sobolmara`, `sobolGP`

Examples

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

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

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

```
# sensitivity analysis

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

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

Description

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

Usage

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

Arguments

`model`	a function, or a model with a `predict` method, defining the model to analyze.
`X1`	the first random sample.
`X2`	the second random sample.
`nboot`	the number of bootstrap replicates.
`conf`	the confidence level for bootstrap confidence intervals.
`x`	a list of class `"sobol"` storing the state of the sensitivity study (parameters, data, estimates).
`y`	a vector of model responses.
`return.var`	a vector of character strings giving further internal variables names to store in the output object `x`.
`ylim`	y-coordinate plotting limits.
`...`	any other arguments for `model` which are passed unchanged each time it is called

Details

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

Value

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

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

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

Author(s)

Bertrand Iooss

References

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

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

See Also

`sobol`, `sobol2002`, `soboljansen`, `sobolEff`, `sobolmara`

Examples

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

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

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

# sensitivity analysis
```

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

`sobolCert` *Monte Carlo Estimation of Sobol' Indices using certified meta-models*

Description

`sobolCert` implements the Monte Carlo estimation of the Sobol' sensitivity indices using certified metamodels using the formulas in Janon et al. (2011).

Usage

```
sobolCert(model = NULL, X1=NULL, X2=NULL, nboot = 300, conf = 0.95, lambda0 = 0, h = 0)
## S3 method for class 'sobolCert'
print(x, ...)
```

Arguments

`model`	a function defining the model to analyze. This function must return a list whose components are: • outmetamodel output. • errmetamodel output error bound, satisfying $$\lvert model_{output} - metamodel_{output} \rvert <= err$$
`X1`	the first random sample. If NULL, sobolEff ignores model, X1 and X2, and reuse model outputs from the previous sobolCert call.
`X2`	the second random sample.
`nboot`	the number of bootstrap replicates.
`conf`	the confidence level for confidence intervals.
`lambda0`	if lambda0=0, use the method described in Janon et al. (2011) Section 3.1; else use the method of Section 3.2 with lambda0 as penalty parameter.
`h`	if lambda0=0, this parameter is ignored; else it is used as the h bandwidth parameter.
`x`	a sobolCert object
`...`	currently not used

Value

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

`call`	the matched call.
`S`	the estimations of the Sobol' sensitivity indices.
`penalty`	(only if lambda0>0) value of the smoothing penalty.

Author(s)

Alexandre Janon

References

Janon, A., Nodet M., Prieur C. (2011) Uncertainties assessment in global sensitivity indices estimation from metamodels. To appear in International Journal for Uncertainty Quantification.

See Also

`sobol`, `sobol2002`, `sobol2007`

Examples

```
## Not run:
# Test case

n <- 1000
X1 <- data.frame(matrix(runif(3 * n), nrow = n))
X2 <- data.frame(matrix(runif(3 * n), nrow = n))

# sensitivity analysis
x=sobolCert(model=function(X) { list(out=X[1]+2*X[2]+X[3]+.001*runif(1),err=.01); },
            X1, X2, conf=.99, lambda0=.1, h=.1, nboot=30)
print(x)

x=sobolCert(model=NULL, X1=NULL, X2=NULL, conf=.95)
print(x)

## End(Not run)
```

`sobolEff` *Monte Carlo Estimation of Sobol' Indices*

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). This method allows the estimation of all first-order p indices at a cost of N*(p+1) model calls, where p is the number of inputs and N is the random sample size, or the estimation of all closed second order indices at a cost of N*(p*(p-1)/2+1) model calls.

Usage

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

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

Arguments

`model`	a function, or a model with a `predict` method, defining the model to analyze.
`X1`	the first random sample.
`X2`	the second random sample.
`order`	an integer specifying the order of the indices (1 or 2).
`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

References

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

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

See Also

sobol, sobol2002, sobol2007

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)
print(x)
```

`sobolGP` *Kriging-based sensitivity analysis*

Description

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

Usage

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

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

## S3 method for class 'sobolGP'
```

```
tell(x,y=NULL,xpoint=NULL,newcandidate=NULL, ...)

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

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

Arguments

`model`	an object of class `"km"` specifying the kriging model built from package `"DiceKriging"` (see `km`).
`type`	a character string giving the type of the considered kriging model. `"SK"` refers to simple kriging and `"UK"` refers to universal kriging (see `km`).
`MCmethod`	a character string specifying the Monte-Carlo procedure used to estimate the Sobol indices. The avaible methods are : `"sobol"`, `"sobol2002"`, `"sobol2007"`, `"sobolEff"` and `"soboljansen"`.
`X1`	a matrix representing the first random sample.
`X2`	a matrix representing the second random sample.
`nsim`	an integer giving the number of samples for the conditional Gaussian process. It is used to quantify the uncertainty due to the kriging approximation.
`nboot`	an integer representing the number of bootstrap replicates. It is used to quantify the uncertainty due to the Monte-Carlo integrations. We recommend to set `nboot = 100`.
`conf`	a numeric representing the confidence intervals taking into account the uncertainty due to the bootstrap procedure and the Gaussian process samples.
`sequential`	a boolean. If `sequential=TRUE`, the procedure provides a new point where to perform a simulation. It is the one minimizing the sum of the MAIN effect estimate variances. The variance is taken with respect to the conditional Gaussian process. The new point is selected in the points `candidate`.
`candidate`	a matrix representing the candidate points where the best new point to be simulated is selected. The lines represent the points and the columns represent the dimension.
`sequential.tot`	a boolean. If `sequential.tot=TRUE`, the procedure provides a new point where to perform the simulation. It is the one minimizing the sum of the TOTAL effect estimate. The variance is taken with respect to the conditional Gaussian process. The new point is selected in the points `candidate`.
`max_iter`	a numeric giving the maximal number of iterations for the propagative Gibbs sampler. It is used to simulate the realizations of the Gaussian process.
`x`	an object of class S3 `"sobolGP"` obtaining from the procedure `sobolGP`. It stores the results of the Kriging-based global sensitivity analysis.
`tot`	a boolean. If `tot=TRUE`, the procedure ask provides a point relative to the uncertainty of the total Sobol' indices (instead of first order' ones).
`xpoint`	a matrix representing a new point added to the kriging model.

`y`	a numeric giving the response of the function at `xpoint`.
`newcandidate`	a matrix representing the new candidate points where the best point to be simulated is selected. If `newcandidate=NULL`, these points correspond to `candidate` without the new point `xpoint`.
`...`	any other arguments to be passed

Details

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

Value

An object of class S3 `sobolGP`.

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

- mean : a matrix giving the mean of the Sobol index estimates.
- var : a matrix giving the variance of the Sobol index estimates.
- ci : a matrix giving the confidence intervals of the Sobol index estimates according to `conf`.
- varPG : a matrix giving the variance of the Sobol index estimates due to the Gaussian process approximation.
- varMC : a matrix giving the variance of the Sobol index estimates due to the Monte-Carlo integrations.
- xnew : if `sequential.tot=TRUE`, a matrix giving the point in `candidate` which is the best to simulate.
- xnewi : if `sequential.tot=TRUE`, an integer giving the index of the point in `candidate` which is the best to simulate.

Author(s)

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

References

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

See Also

`sobol`, `sobol2002`, `sobol2007`, `sobolEff`, `soboljansen`, `km`

Examples

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

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

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

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

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

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

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

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

res <- sobolGP(
model = m,
type="UK",
MCmethod="sobol",
X1,
X2,
nsim = 100,
conf = 0.95,
nboot=100,
sequential = TRUE,
candidate,
sequential.tot=FALSE,
max_iter = 1000
)

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

## End(Not run)
```

---

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

Description

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

Usage

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

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 large first-order indices, and (large and small) total indices.

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`	the response used
`V`	the estimations of Variances of the Conditional Expectations (VCE) with respect to each factor and also with respect to the complementary set of each factor ("all but X_i").
`S`	the estimations of the Sobol' first-order indices.
`T`	the estimations of the Sobol' total sensitivity indices.

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

Author(s)

Bertrand Iooss

References

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

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

See Also

`sobol`, `sobol2002`, `sobol2007`, `sobolEff`, `sobolmara`

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)
```

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

Description

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

Usage

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

Arguments

`model`	a function, or a model with a `predict` method, defining the model to analyze.
`X1`	the random sample.
`x`	a list of class `"sobolEff"` storing the state of the sensitivity study (parameters, data, estimates).

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

Details

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

Value

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

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

Author(s)

Bertrand Iooss

References

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

See Also

`sobol`, `sobol2002`, `sobol2007`, `soboljansen`, `sobolEff`

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)
```

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

Description

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

Usage

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

Arguments

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

Value

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

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

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

Author(s)

Taieb Touati and Bernardo Ramos

References

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

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

See Also

sobol, sobol2002, sobol2007, soboljansen, sobolEff, sobolmara, sobolGP

Examples

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

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

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

# sensitivity analysis

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

```
print(x)
plot(x)

## End(Not run)
```

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

Description

`sobolroalhs` implements the estimation of the Sobol' sensitivity indices introduced by Tissot & Prieur (2012) using two Orthogonal Array-based Latin Hypercubes. This function allows the estimation of all first-order indices at a total cost of $2 \times N$ or all second-order subset indices (containing the sum of the second-order effect between two inputs and the individual effects of each input) at a cost of $2 \times q^2$ where q is a prime number. Second-order effects require the package `numbers`.

Usage

```
sobolroalhs(model = NULL, factors, levels, order, choice="A", conf=0.95, ...)
## 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), type="standard", ...)
```

Arguments

`model`	a function, or a model with a `predict` method, defining the model to analyze.
`factors`	an integer specifying the number of factors, or a vector of character strings giving their names.
`levels`	an integer specifying the number of levels of the design.
`order`	an integer specifying the order of the indices (1 or 2).
`choice`	a character (`A` or `B`) specifying the method to generate the orthogonal array-based design (`A` for the Bose method and `B` for the Hadamard generalized matrix based method).
`conf`	the confidence level for confidence intervals.
`x`	a list of class `"sobolroalhs"` storing the state of the sensitivity study (parameters, data, estimates).
`y`	a vector of model responses.
`ylim`	coordinate plotting limits for the indices values.
`type`	a character specifying the type of estimator to plot (`standard` for the basic estimator or `monod` for the Janon-Monod estimator.)
`...`	any other arguments for `model` which are passed unchanged each time it is called.

Details

The method used by sobolroalhs only considers models whose inputs follow uniform distributions on [0,1]. The transformations of each input (between its initial distribution and a U[0,1] distribution) have therefore to be realized before the call to sobolroalhs().

Bootstrap confidence intervals are not available with this method ; the given confidence intervals come from asymptotical formula.

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 `matrix` containing the design of experiments.
`y`	a vector of model responses.
`V`	a data.frame containing the estimations of the variance (`Vs` for the standard variance and `Veff` for the Janon-Monod variance).
`S`	a data.frame containing the estimations of the Sobol' sensitivity indices (`S` for the standard estimator and `Seff` for the Janon-Monod estimator).

Warning messages

"the number of levels recquired was not a prime number. the number was replaced by : " when order=2, the number of levels must be a prime number. This warning message indicates that the number of levels specified was note a prime number and was automatically replaced by the prime number following the square root of `factors`.

"the number of levels recquired was not satisfying the constraint, the number was replaced by : " when order=2, the number of levels must satisfied the constrain $N \geq (d-1)^2$ where d is the number of factors.

Author(s)

Laurent Gilquin

References

Tissot, J. Y. and Prieur, C. (2012), *Estimating Sobol's indices combining Monte Carlo integration and Latin hypercube sampling*.

A.S. Hedayat, N.J.A. Sloane, John Stufken (1999), *Orthogonal Arrays: Theory and Applications*.

See Also

`sobol`, `sobolmara`

Examples

```
# Test case : the non-monotonic Sobol g-function
library(numbers)

# 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, levels = 1000, order = 1)
print(x)
plot(x)

# global second-order sensitivity indices
x <- sobolroalhs(model = sobol.fun, factors = 8, levels = 1000, order = 2)
print(x)
plot(x)

# Test case : the Ishigami function

# New function because sobolroalhs() works with U[0,1] inputs
ishigami1.fun=function(x) ishigami.fun(x*2*pi-pi)

x <- sobolroalhs(model = ishigami1.fun, factors = 3, levels = 100000, order = 1)
print(x)
plot(x)

x <- sobolroalhs(model = ishigami1.fun, factors = 3, levels = 100000, order = 2)
print(x)
plot(x)
```

---

src *Standardized Regression Coefficients*

Description

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

Usage

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

Arguments

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

Value

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

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

Author(s)

Gilles Pujol

References

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

See Also

`pcc`

Examples

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

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

```
# linear model : Y = X1 + X2 + X3

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

# sensitivity analysis

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

template.replace *Replace Values in a Template Text*

Description

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

Usage

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

Arguments

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

Details

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

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

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

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

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

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

Author(s)

Gilles Pujol

Examples

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

`testmodels` *Test Models for Sensitivity Analysis*

Description

These functions are standard testcase for sensitivity analysis benchmarks: The g-function of Sobol', the function of Ishigami and the function of Morris (see Saltelli et al. 2000, section 2.9)

Usage

```
sobol.fun(X)
ishigami.fun(X)
morris.fun(X)
```

Arguments

`X`	a matrix (or `data.frame`) containing the input sample.

Value

A vector of function responses.

Author(s)

Gilles Pujol

References

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

Index