Package ‘sensitivity’

April 19, 2009

Version 1.4-0

Date 2008-07-14

Title Sensitivity Analysis

Author Gilles Pujol

Maintainer Bertrand Iooss <bertrand.iooss@cea.fr>

Depends R (>= 2.7.0), boot

Suggests rgl

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

License CeCILL-2

Repository CRAN

Date/Publication 2008-07-15 15:47:55

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`).
- Morris's "OAT" elementary effects screening method (`morris`).
- Bettonvil's sequential bifurcations (`sb`).
- Monte Carlo estimation of Sobol' indices: Sobol's scheme (1993) to compute the indices given by the variance decomposition up to a specified order (`sobol`), and Saltelli's scheme (2002) to compute first order and total indices with a reduced cost (`sobol2002`).
- Estimation of the Sobol' first order and total indices with Saltelli's so-called "extended-FAST" method (`fast99`).

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

Model managing

The **sensitivity** package works either on R models than on external models (such as executables).

R models must be functions or objects that have a `predict` method, such as `lm` objects. Models defined as functions will be called once with an expression of the form `y <- f(X)` where `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 (we say that such functions are vectorized).

If the model is external to R, for instance a computational code, it must be analyzed with the decoupled approach, see `decoupling`. This approach can also be used on R models that doesn't fit the specifications.

References

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

`decoupling` *Decoupling Simulations and Estimations*

Description

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

Usage

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

Arguments

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

Details

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

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

Value

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

`ask` returns the set of data to simulate.

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").

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.

See Also

`decoupling`

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 2008).

Usage

```
morris(model = NULL, factors, r, design, binf = 0, bsup = 1,
       scale = TRUE, ...)
## S3 method for class 'morris':
tell(x, y = NULL, ...)
## S3 method for class 'morris':
print(x, ...)
## S3 method for class 'morris':
plot(x, identify = FALSE, ...)
plot3d.morris(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 and output data are scaled before computing the elementary effects.
`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)
```

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.

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 (2008), *Simplex-based screening designs for estimating metamodels*, submited to Reliability Engineering and System Safety.

See Also

`decoupling`

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

---

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`).

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)
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...
```

sb *Sequential Bifurcations*

Description

sb implements the Sequential Bifurcations screening method (Bettonvil and Kleijnen 1996). **This is an alpha version that might strongly evolve in the future**.

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.

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)
```

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

Description

`sobol` implements the Monte Carlo estimation of the Sobol' sensitivity indices. 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 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`).

References

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

See Also

`sobol2002`

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 <- 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.

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`).

References

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

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

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)

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).

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.

References

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

Index