Package ‘SAFER’

July 20, 2015

Title Sensitivity Analysis For Everybody with R

Version 1.1

Description Range of tools for Global Sensitivity Analysis (GSA)

Depends R (>= 3.0.0), caTools, calibrater

License file LICENSE

URL http://bristol.ac.uk/cabot/resources/safe-toolbox/

LazyData true

NeedsCompilation no

Author Isabella Gollini [aut, cre],
Francesca Pianosi [aut],
Fanny Sarrazin [aut],
Thorsten Wagener [aut]

Maintainer Isabella Gollini <igollini.stats@gmail.com>

R topics documented:

`SAFER-package` *Sensitivity analysis for everybody with R*

Description

Range of tools for Global Sensitivity Analysis (GSA).

Details

It implements several established GSA methods, including method of Morris, regional sensitivity analysis, variance-based sensitivity analysis (Sobol') and FAST. It also includes new approaches and visualization tools to complement these established methods.

`SAFER` is also available in Matlab/Octave version through a Toolbox called `SAFE`. `SAFE` and `SAFER` are open source and freely available from the following website: `http://bristol.ac.uk/cabot/resources/safe-toolbox/`.

BEFORE STARTING

An introduction to the SAFE Toolbox is provided in the paper: Pianosi, F., Sarrazin, F., Wagener, T. (2015), A Matlab toolbox for Global Sensitivity Analysis, Environmental Modelling & Software, 70, 80-85. The paper is freely available at: `http://www.sciencedirect.com/science/article/pii/S1364815215001188`. We recommend reading this (short) paper before getting started.

TO GET STARTED

To get started using SAFER, we suggest opening one of the workflow scripts and running the code step by step. The header of each workflow script gives a short description of the method and case study model, and of the main steps and purposes of that workflow, as well as references for further reading. The name of each workflow is composed as: workflow_<method>_<model>

Implemented models are:

- the hydrological Hymod model (see documentation and help of functions `help("hymod")` and `system.file("docs", "Hymod_structure.pdf", package = "SAFER")`)
- the hydrological HBV model (see documentation and help of functions in `help("hbv")` and `system.file("docs", "HBV_structure.pdf", package = "SAFER")`)

- the Ishigami and Homma test function (see help of functions in `help("ishigami_homma")`
- the Sobol' g-function (see help of functions in `help("sobol_g_function")`

Implemented methods are:

- eet (elementary effects test, or method of Morris)
- fast (Fourier amplitude sensitivity test)
- rsa (regional sensitivity analysis)
- vbsa (variance-based sensitivity analysis, or method of Sobol')

Furthermore, SAFER includes additional workflow scripts:

- visual: how to use visualisation functions for qualitative GSA

There are a 8 workflows in the demo folder of the package:

- `workflow_eet_hbv` This script provides an application example of the Elementary Effects Test to the HBV rainfall-runoff model.
- `workflow_eet_hymod` This script provides a basic application example of the Elementary Effects Test. The application example is the rainfall-runoff Hymod model.
- `workflow_fast_gsobol` This script provides an application example of the Fourier Amplitude Sensitivity Test (FAST). The application example is the Sobol g-function.
- `workflow_fast_hymod` This script provides an application example of the Fourier Amplitude Sensitivity Test (FAST). The application example is the rainfall-runoff Hymod model.
- `workflow_rsa_hymod` This script provides an application example of regional sensitivity analysis. The application example is the rainfall-runoff Hymod model.
- `workflow_vbsa_hymod` This script provides an application example of Variance Based Sensitivity Analysis (VBSA) The application example is the rainfall-runoff Hymod model
- `workflow_vbsa_ishigami_homma` This script applies Variance-Based Sensitivity Analysis to the Ishigami-Homma function.
- `workflow_visual_ishigami_homma` This script provides an application example of how to use several visualization tools (scatter plots, coloured scatter plots, parallel coordinate plots, Andres' plot) to learn about sensitivity. The application example is the Ishigami-Homma function.

The command to find the list of demo in the package is `demo(package = "SAFER")`

The command to start a demo is `demo("workflow_visual_ishigami_homma")`

The command to find the location of the demo folder is `system.file("demo", package = "SAFER")`

`SAFER` depends on the package `calibrater` that is available on Jonty Rougier's webpage `http://www.maths.bris.ac.uk/~mazjcr/#software`.

`AAT_sampling` *Random sampling in the M-dimension space of the input factors of a model.*

Description

This function performs random sampling in the `M`-dimension space of the input factors of a model. To this purpose, the function first perform uniform sampling in the unit hypercube in R^`M`; then, project the sampled points along each direction.

Usage

```
AAT_sampling(samp_strat, M, distr_fun, distr_par, N)
```

Arguments

`samp_strat`	string, sampling strategy. Options: `"rsu"`: random uniform, `"lhs"`: latin hypercube.
`M`	positive integer, number of inputs.
`distr_fun`	string (eg: `"unif"`) if all inputs have the same pdf, or vector of `M` strings (eg: `c("unif", "norm")`) containing the probability distribution function of each input.
`distr_par`	row vector if all input pdfs have the same parameters, list of `M` vectors otherwise containing the parameters of the probability distribution function.
`N`	positive integer, number of samples.

Value

`X` matrix (`N`, `M`) of samples. Each row is a point in the input space. In contrast to `OAT_sampling`, rows do not follow any specific order, and all components (columns) differ from point (row) to point.

See Also

`OAT_sampling`

Examples

```
# Example 1: 2 inputs, both from Unif[0,3]
N <- 1000
M <- 2
distr_fun <- "unif"
distr_par <- c(0, 3)
samp_strat <- "lhs"
X <- AAT_sampling(samp_strat, M, distr_fun, distr_par, N)
# Plot results:
plot(X, xlab = expression(x[1]), ylab = expression(x[2]))
#
# Example 2: 2 inputs, one from Unif[0,3], one from Unif[1,5]
distr_fun <- "unif"
distr_par <- list(c(0, 3), c(1, 5))
X <- AAT_sampling(samp_strat, M, distr_fun, distr_par, N)
```

```
# Plot results:
plot(X, xlab = expression(x[1]), ylab = expression(x[2]))

##################################
# Some more comments about sampling
##################################

# If you want to see the difference between Latin-Hypercube, Sobol and
# Monte Carlo sampling strategy, you can use the following code:

N <- 100
dist_type  <- "unif"
dist_param <- c(0, 1)
X1 <- AAT_sampling("rsu", 2, dist_type, dist_param, N)
X2 <- AAT_sampling("lhs", 2, dist_type, dist_param, N)
par(mfrow = c(1, 2))
plot(X1, xlab = expression(x[1]), ylab = expression(x[2]), main = ("Random Uniform"))
plot(X2, xlab = expression(x[1]), ylab = expression(x[2]), main = ("Latin Hypercube"))
# Example with a Normal distribution on the vertical axis:
dist_type  <- c("unif", "norm")
dist_param <- list(c(0, 1), c(5, 1))
X1 <- AAT_sampling("rsu", 2, dist_type, dist_param, N)
X2 <- AAT_sampling("lhs", 2, dist_type, dist_param, N)
par(mfrow = c(1, 2))
plot(X1, xlab = expression(x[1]), ylab = expression(x[2]), main = ("Random Uniform"))
plot(X2, xlab = expression(x[1]), ylab = expression(x[2]), main = ("Latin Hypercube"))
```

AAT_sampling_extend *Create an expanded sample* X_new *starting from a sample* X

Description

This function create an expanded sample X_new starting from a sample X and using latin hypercube and the maximin criterion.

Usage

```
AAT_sampling_extend(X, distr_fun, distr_par, N_new, nrep = 10)
```

Arguments

X	matrix (N,M) of initial samples.
distr_fun	string (eg: "unif") if all inputs have the same pdf, or vector of M strings (eg: c("unif", "norm")) containing the probability distribution function of each input.
distr_par	row vector if all input pdfs have the same parameters, list of M vectors otherwise containing the parameters of the probability distribution function.
N_new	positive integer, new dimension of the sample (must be > N)
nrep	scalar, number of replicate to select the maximin hypercube (default value: 10)

Value

X_new matrix (N_new, M) of expanded sample.

See Also

AAT_sampling

Examples

```
# Example 1: 2 inputs, both from Unif[0,3]
N <- 1000
M <- 2
distr_fun <- "unif"
distr_par <- c(0, 3)
samp_strat <- "lhs"
X <- AAT_sampling(samp_strat, M, distr_fun, distr_par, N)
# Plot results:
plot(X, xlab = expression(x[1]), ylab = expression(x[2]))
#
# Adding up new samples
N2 <- 500 # increase of base sample size
# (that means: N2 * (M+2) new samples that will need to be evaluated)
Xext <- AAT_sampling_extend(X, distr_fun, distr_par, 2 * (N + N2)) # extended sample
# (it includes the already evaluated samples X and the new ones)
Xnew <- Xext[-(1:(2 * N)),] # extract the new input samples that need to be evaluated
# Plot results:
par(mfrow = c(1, 2))
plot(X, xlab = expression(x[1]), ylab = expression(x[2]), main = "X")
plot(Xnew, xlab = expression(x[1]), ylab = expression(x[2]), main = "X new")
```

Andres_plots *Andres plots*

Description

This function creates the validation plots first proposed by Andres (1993) (see also applications in Tang et al. (2007))

Usage

```
Andres_plots(X, Y, X_ref, idx, fun_test, ...)
```

Arguments

X	matrix NxM set of input samples
Y	vector N set of output samples
X_ref	vector M reference values for the inputs
idx	scalar. Index of input to be analyzed
fun_test	character. Name of the function implementing the model: $Y = f(X)$
...	other parameters needed in output_def

References

Andres, T.H. (1997). Sampling methods and sensitivity analysis for large parameter sets. Journal of Statistical Computation and Simulation, 57(1-4), 77-110.

Tang, Y., Reed, P., Van Werkhoven, K. and Wagener, T. (2007). Advancing the identification and evaluation of distributed rainfall-runoff models using global sensitivity analysis. Water Resources Research.

Examples

```
# Step 1 (setup the model)

fun_test  <- "ishigami_homma_function"
M <- 3
distr_fun <- "unif"
distrpar <-  c(-pi, pi)

# Step 2 (sampling and model evaluation)

N <- 3000
X <- AAT_sampling("lhs", M, distr_fun, distrpar, N)
Y <- model_evaluation(fun_test, X)

# Step 3 (Andres' visualization test)

Xref <- c(2,2,2)
idx <- 1
Andres_plots(X, Y, Xref, idx, fun_test)
```

---

boxplot1 *Boxplot of vectors Boxplots when the mean, lower values and upper values are specified.*

Description

Boxplot of vectors Boxplots when the mean, lower values and upper values are specified.

Usage

```
boxplot1(a, al = NULL, au = NULL, labels = NULL, col = 2:(length(a) +
  1), ylim = NULL, ...)
```

Arguments

`a`	vector (`M`) of mean values to be plotted
`al`	vector (`M`) of lower values to be plotted. Default `al = NULL`
`au`	vector (`M`) of upper values to be plotted. Default `au = NULL`
`labels`	labels for the x-axis of the boxplot
`col`	colors for the plot. Default `col = 2:(length(a) + 1)`
`ylim`	limits of the y-axis. Default `ylim = NULL`
`...`	others arguments to be passed in `plot`

See Also

boxplot2 boxplot plot

Examples

```
# Setup the model and define input ranges
myfun  <- "ishigami_homma_function"
M <- 3
DistrFun <- "unif"
DistrPar <-  c(-pi, pi)
# Sample parameter space using the resampling strategy proposed by
# (Saltelli, 2008; for reference and more details, see help of functions
# vbsa_resampling and vbsa_indices)
SampStrategy <- "lhs"
N <- 3000 # Base sample size.
# Comment: the base sample size N is not the actual number of input
# samples that will be evaluated. In fact, because of the resampling
# strategy, the total number of model evaluations to compute the two
# variance-based indices is equal to N*(M+2)
X <- AAT_sampling(SampStrategy, M, DistrFun, DistrPar, 2 * N)
XABC <- vbsa_resampling(X)
# Run the model and compute selected model output at sampled parameter
# sets:
YA <- model_evaluation(myfun, XABC$XA) # size (N,1)
YB <- model_evaluation(myfun, XABC$XB) # size (N,1)
YC <- model_evaluation(myfun, XABC$XC) # size (N*M,1)
# Compute main (first-order) and total effects:
ind <- vbsa_indices(YA, YB, YC)
Si <- ind[1,]
STi <- ind[2,]
# Plot results:
# plot main and total separately
par(mfrow = c(1, 2))
boxplot1(Si, main = "Si")
boxplot1(STi, main = "STi")
```

---

boxplot2 — *Boxplot of vectors with two groups Boxplots when the mean, lower values and upper values are specified for two groups a and b.*

Description

Boxplot of vectors with two groups Boxplots when the mean, lower values and upper values are specified for two groups a and b.

Usage

```
boxplot2(a, b, al = NULL, au = NULL, bl = NULL, bu = NULL,
  labels = NULL, leg = NULL, col = 2:3, ylim = NULL, ...)
```

Arguments

`a`	vector (`M`) of mean values to be plotted
`b`	vector (`M`) of mean values to be plotted
`al`	vector (`M`) of lower values to be plotted. Default `al = NULL`
`au`	vector (`M`) of upper values to be plotted. Default `au = NULL`
`bl`	vector (`M`) of lower values to be plotted. Default `bl = NULL`
`bu`	vector (`M`) of upper values to be plotted. Default `bu = NULL`
`labels`	labels for the x-axis of the boxplot
`leg`	names for the legend. Default `leg = NULL`
`col`	colors for the plot. Default `col = 2:3`
`ylim`	limits of the y-axis. Default `ylim = NULL`
`...`	others arguments to be passed in `plot`

See Also

`boxplot1` `boxplot` `plot`

Examples

```
# Setup the model and define input ranges
myfun  <- "ishigami_homma_function"
M <- 3
DistrFun <- "unif"
DistrPar <-  c(-pi, pi)
# Sample parameter space using the resampling strategy proposed by
# (Saltelli, 2008; for reference and more details, see help of functions
# vbsa_resampling and vbsa_indices)
SampStrategy <- "lhs"
N <- 3000 # Base sample size.
# Comment: the base sample size N is not the actual number of input
# samples that will be evaluated. In fact, because of the resampling
# strategy, the total number of model evaluations to compute the two
# variance-based indices is equal to N*(M+2)
X <- AAT_sampling(SampStrategy, M, DistrFun, DistrPar, 2 * N)
XABC <- vbsa_resampling(X)
# Run the model and compute selected model output at sampled parameter
# sets:
YA <- model_evaluation(myfun, XABC$XA) # size (N,1)
YB <- model_evaluation(myfun, XABC$XB) # size (N,1)
YC <- model_evaluation(myfun, XABC$XC) # size (N*M,1)
# Compute main (first-order) and total effects:
ind <- vbsa_indices(YA, YB, YC)
Si <- ind[1,]
STi <- ind[2,]
# Plot results:
boxplot2(Si, STi, leg = c("main effects", "total effects"))
```

EET_convergence *Mean and standard deviation of Elementary Effects*

Description

Compute mean and standard deviation of Elementary Effects (EEs) using an increasing number of samples.

Usage

```
EET_convergence(EE, rr, Nboot = 0, alfa = 0.05)
```

Arguments

EE	matrix (r, M) of r elementary effects
rr	vector (R), sample sizes at which indices will be estimated (must be a vector of integer positive values not exceeding r)
Nboot	scalar, number of resamples used for boostrapping (default: 0)
alfa	scalar, significance level for the confidence intervals estimated by bootstrapping (default: 0.05)

Value

List containing:

- m_r mean of EEs at different sampling size
- s_r standard deviation of EEs at different sampling size

If Nboot > 1 it also contains

- m_lb_r lower bound of EEs mean at different sampling size
- m_ub_r upper bound of EEs mean at different sampling size
- s_lb_r lower bound of EEs std.dev. at different sampling size
- s_ub_r upper bound of EEs std.dev. at different sampling size
- m_sd_r standard deviation of EEs mean at different sampling size
- s_sd_r standard deviation of EEs std.dev. at different sampling size.

All output arguments are matrices of size (R,M)

See Also

EET_indices EET_plot

Examples

```
# See the demo
# demo("workflow_eet_hymod")
# or
# demo("workflow_eet_hbv")
```

EET_indices *Mean and standard deviation of Elementary Effects*

Description

This function computes mean and standard deviation of Elementary Effects (EEs)

Usage

```
EET_indices(r, xrange, X, Y, design_type, Nboot = 0, alfa = 0.05)
```

Arguments

`r`	scalar, number of sampling point
`xrange`	list of `M` vectors of length 2 containing the input ranges
`X`	matrix (`r * (M + 1)`, `M`) matrix of sampling datapoints where EE must be computed
`Y`	vector (`r * (M + 1)`) vector of output values
`design_type`	string, design type (string) Options: `"trajectory"`, `"radial"`
`Nboot`	scalar, number of resamples used for boostrapping (default:0) - scalar
`alfa`	scalar, significance level for the confidence intervals estimated by bootstrapping (default: 0.05)

Value

List containing:

- `mi` vector (`M`) mean of the elementary effects
- `sigma` vector (`M`) standard deviation of the elementary effects
- `EE` matrix (`r`, `M`) of elementary effects

If `Nboot > 1` it also contains

- `mi_sd` vector (`M`) standard deviation of `mi` across `Nboot` estimations
- `sigma_sd` vector (`M`) standard deviation of `sigma` across `Nboot` estimations
- `mi_lb` vector (`M`) lower bound of `mi` (at level `alfa`) across `Nboot` estimations
- `sigma_lb` vector (`M`) lower bound of `sigma` across `Nboot` estimations
- `mi_ub` vector (`M`) upper bound of `mi` (at level `alfa`) across `Nboot` estimations
- `sigma_ub` vector (`M`) upper bound of `sigma` across `Nboot` estimations.

See Also

`EET_convergence` `EET_plot`

Examples

```
# See the demo
# demo("workflow_eet_hymod")
# or
# demo("workflow_eet_hbv")
```

`EET_plot` *EET plot Plot the mean and the standard deviation of the elementary effects.*

Description

EET plot Plot the mean and the standard deviation of the elementary effects.

Usage

```
EET_plot(m, s, ml = NULL, mu = NULL, sl = NULL, su = NULL,
  labels = NULL, xlim = NULL, ylim = NULL, xlab = "Mean of EEs",
  ylab = "Sd of EEs", col = 1:length(m), pch = 15:19, ...)
```

Arguments

`m`	vector (`M`) mean of the elementary effects
`s`	vector (`M`) standard deviation of the elementary effects
`ml`	vector (`M`) lower bound of `mi` (at level `alfa`) across `Nboot` estimations. Default `ml = NULL`
`mu`	vector (`M`) upper bound of `mi` (at level `alfa`) across `Nboot` estimations. Default `mu = NULL`
`sl`	vector (`M`) lower bound of `sigma` across `Nboot` estimations. Default `sl = NULL`
`su`	vector (`M`) upper bound of `sigma` across `Nboot` estimations. Default `su = NULL`
`labels`	labels for legend. Default `labels = NULL`
`xlim`	limits of the x-axis. Default `xlim = NULL`
`ylim`	limits of the y-axis. Default `ylim = NULL`
`xlab`	label for the x-axis. Default `xlab = 'Mean of EEs'`
`ylab`	label for the x-axis. Default `ylab = 'Sd of EEs'`
`col`	colors for the plot. Default `col = 1:length(m)`
`pch`	symbols to use when plotting points. Default `pch = 15:19`
`...`	others arguments to be passed in `plot`

See Also

`EET_indices` `EET_convergence` `plot`

Examples

```
# See the demo
# demo("workflow_eet_hymod")
# or
# demo("workflow_eet_hbv")
```

`FAST_indices`	*Main effect (first-order) sensitivity index for the Fourier Amplitude Sensitivity Test (FAST)*

Description

Computes main effect (first-order) sensitivity index according to the Fourier Amplitude Sensitivity Test (FAST; Cukier et al., 1978).

Usage

```
FAST_indices(Y, M, Nharm = 4, omega = generate_FAST_frequency(M))
```

Arguments

`Y`	vector (`N`), set of model output samples
`M`	scalar, number of inputs
`Nharm`	scalar, interference factor, i.e.the number of higher harmonics to be considered (default is 4)
`omega`	vector (`M`), angular frequencies associated to inputs (default values computed by function `generate_FAST_frequency`)

Value

List containing:

- `Si` (vector of length `M`) of main effect (first-order) sensitivity indices
- `V` (scalar) total output variance
- `A` (vector `N`) Fourier coefficients
- `B` (vector `N`) Fourier coefficients
- `Vi` (vector of length `M`) output variances from each input

References

Cukier, R.I., Levine, H.B., and Shuler, K.E. (1978), Nonlinear Sensitivity Analyis of Multiparameter Model SYstems, Journal of Computational Physics, 16, 1-42.

Saltelli, A., Tarantola, S. and Chan, K.P.S. (1999), A Quantitative Model-Independent Method ofr Global Sensitivty Analysis of Model Output, Technometrics, 41(1), 39-56.

See Also

`FAST_sampling` `generate_FAST_frequency`

Examples

```
fun_test  <- 'sobol_g_function'
# Define input distribution and ranges:
M <- 5 # may range from 5 to 11
distr_fun <- 'unif'
distrpar <-  c(0, 1)
a <- 9

## Step 1: Choose sampling size #

Nfast <- 2^5 + 1 # this value just for a quick example
# it would be preferred: Nfast<- 2^13 + 1
## Step 2: Approximate first-order variance-based indices by FAST

# FAST
Fsamp <- FAST_sampling(distr_fun, distrpar, M, N = Nfast)
X <- Fsamp$X
s <- Fsamp$s
# Run the model and compute selected model output at sampled parameter
# sets:
Y <- model_evaluation(fun_test, X, a = a)
# Estimate indices:
Si_fast <- FAST_indices(Y, M)
```

`FAST_sampling` *Sampling for the Fourier Amplitude Sensitivity Test (FAST)*

Description

Implements sampling for the Fourier Amplitude Sensitivity Test (FAST; Cukier et al., 1978) and returns a matrix `X` of `N` input samples.

Usage

```
FAST_sampling(distr_fun, distr_par, M, Nharm = 4,
  omega = generate_FAST_frequency(M), N = 2 * Nharm * max(omega) + 1)
```

Arguments

`distr_fun`	string (eg: `'unif'`) if all inputs have the same pdf, or vector of `M` strings (eg: `c('unif','norm')`) containing the probability distribution function of each input.
`distr_par`	row vector if all input pdfs have the same parameters, list of `M` vectors otherwise containing the parameters of the probability distribution function.
`M`	scalar, number of inputs
`Nharm`	scalar, interference factor, i.e.the number of higher harmonics to be considered (default is 4, taken from Saltelli et al. (1999; page 42))
`omega`	vector (`M`), angular frequencies associated to inputs (default values computed by function `generate_FAST_frequency`)
`N`	odd scalar, number of samples (default is `2 * Nharm * max(omega) + 1` which is the minimum sampling size according to Cukier et al. (1978))

Value

`X` matrix of `N` input samples.

References

Cukier, R.I., Levine, H.B., and Shuler, K.E. (1978), Nonlinear Sensitivity Analyis of Multiparameter Model SYstems, Journal of Computational Physics, 16, 1-42.

Saltelli, A., Tarantola, S. and Chan, K.P.S. (1999), A Quantitative Model-Independent Method ofr Global Sensitivty Analysis of Model Output, Technometrics, 41(1), 39-56.

See Also

`FAST_sampling_unif` `generate_FAST_frequency`

Examples

```
fun_test  <- 'sobol_g_function'
# Define input distribution and ranges:
M <- 5 # may range from 5 to 11
distr_fun <- 'unif'
distrpar <-  c(0, 1)
a <- 9

## Step 1: Choose sampling size

# suggested values of Nfast

Nfast <- 2^13 + 1
## Step 2: Approximate first-order variance-based indices by FAST

# FAST
Fsamp <- FAST_sampling(distr_fun, distrpar, M, N = Nfast)
X <- Fsamp$X
s <- Fsamp$s
```

`FAST_sampling_unif` *Sampling for the Fourier Amplitude Sensitivity Test (FAST)*

Description

Implements sampling for the Fourier Amplitude Sensitivity Test (FAST; Cukier et al., 1978) and returns a matrix `X` of `N` input samples. Inputs are assumed to be uniformly distributed in the unit hypercube [0,1]^`M`. Samples are taken along the search curve defined by transformations

$$x_i(s) = G_i(\sin(\omega_i * s)) \;\; i = 1, ..., M$$

where s is a scalar variable that varies in $(-\frac{1}{\pi}, \frac{1}{\pi})$ Notes: Here we use the curve proposed by Saltelli et al. (1999):

$$x_i(s) = \frac{1}{2} + \frac{1}{\pi} * \arcsin(\sin(\omega_i * s))$$

Usage

```
FAST_sampling_unif(M, Nharm = 4, omega = generate_FAST_frequency(M), N = 2
  * Nharm * max(omega) + 1)
```

Arguments

`M`	scalar, number of inputs
`Nharm`	scalar, interference factor, i.e.the number of higher harmonics to be considered (default is 4, taken from Saltelli et al. (1999; page 42))
`omega`	vector (`M`), angular frequencies associated to inputs (default values computed by function `generate_FAST_frequency`)
`N`	odd scalar, number of samples (default is `2 * Nharm * max(omega) + 1` which is the minimum sampling size according to Cukier et al. (1978))

References

Cukier, R.I., Levine, H.B., and Shuler, K.E. (1978), Nonlinear Sensitivity Analyis of Multiparameter Model SYstems, Journal of Computational Physics, 16, 1-42.

Saltelli, A., Tarantola, S. and Chan, K.P.S. (1999), A Quantitative Model-Independent Method ofr Global Sensitivty Analysis of Model Output, Technometrics, 41(1), 39-56.

See Also

`FAST_sampling` `generate_FAST_frequency`

`generate_FAST_frequency` *Generate FAST frequency*

Description

Generate frequency set free of interferences through (at least) 4th order (vector (`M`))

Usage

```
generate_FAST_frequency(M)
```

Arguments

`M`	integer scalar between 4 and 50, number of inputs

Details

For `M > 4`, frequencies are computed based on the recursive algorithm by Cukier et al. (1975) which is free of interferences through the 4th order. For `M <= 4`, we use values from the literature that guarantee higher order interferences free: if `M = 2` use values from Sec. 3.1 in Xu, C. and G. Gertner (2007) (free of interference through 10th order), if `M = 4` use values from Table III in Cukier et al. (1975) (free of interferences through 6th order).

References

Cukier et al. (1975) Study of the sensitivity of coupled reaction systems to uncertainties in rate coefficients. III. Analysis of the approximations, J. Chem. Phys. 63, 1140

Xu, C. and G. Gertner (2007), Extending a global sensitivity analysis technique to models with correlated parameters, Computational Statistics and Data Analysis, 51, 5579-5590.

Examples

```
M <- 5
generate_FAST_frequency(M)
```

`hbvdata` *hbvdata*

Description

The case study area is is the Nezinscot River at Turner center, Maine, USA (USGS 01055500, see `http://waterdata.usgs.gov/nwis/nwismap`)

Format

Data frame with 20454 rows and 7 columns

Details

- `date` vector `T` of dates of the time series
- `prec` vector `T` time series of precipitation
- `ept` vector `T` time series of evapotranspiration
- `flow` vector `T` time series of observed flow
- `temp` vector `T` time series of temperature, it is the mean of `t1` and `t2`
- `t1` vector `T` time series of temperature `t1`
- `t2` vector `T` time series of temperature `t2`

`hbv_snow_objfun` *hbv snow objfun*

Description

This function simulates the snow accumulation/melting process (via the internal function `snow_routine` and the rainfall-runoff process (via the HBV model by Seibert (1997)) and returns 6 objective functions (see Kollat et al, 2002).

Usage

```
hbv_snow_objfun(param, dat, warmup, Case)
```

Arguments

`param`	vector (13) of model parameters. Snow routine parameters: • `Ts` = threshold temperature [C] • `CFMAX` = degree day factor [mm/C] • `CFR` = refreezing factor [-] • `CWH` = Water holding capacity of snow [-] • `temp` = temperature [C] HBV parameters: • `BETA` = Exponential parameter in soil routine [-] • `LP` = evapotranspiration limit [-] • `FC` = field capacity [mm] • `PERC` = maximum flux from Upper to Lower Zone [mm/Dt] • `K0` = Near surface flow coefficient (ratio) [1/Dt] • `K1` = Upper Zone outflow coefficient (ratio) [1/Dt] • `K2` = Lower Zone outflow coefficient (ratio) [1/Dt] • `UZL` = Near surface flow threshold [mm]
`dat`	dataset containing time series of precipitation (`prec`), evapotranspiration (`ept`), observed flow (`flow`) and temperature (`temp`)
`warmup`	scalar, warmup time
`Case`	scalar, 1 or 2, indicates the preferred path. `Case = 1`: runoff from the upper zone, `Case = 2`: percolation.

Value

List containing:

- `f` = vector (6) of objective functions 1: AME, 2: NSE, 3: BIAS, 4: TRMSE, 5: SFDCE, 6: RMSE
- `Q_sim` = vector (`N`) time series of simulated flow
- `STATES` = matrix `(N, 5)` time series of simulated storages (all in mm) 1: water content of snowpack (snow component) 2: water content of snowpack (liquid component) 3: water content of soil (soil moisture) 4: water content of upper reservoir of flow routing routine 5: water content of lower reservoir of flow routing routine
- `FLUXES` = matrix `(N, 8)` time series of simulated fluxes (all in mm/Dt) 1: refreezing 2: snowmelt 3: actual evapotranspiration 4: recharge (water flux from soil moisture accounting module to flow routing module) 5: percolation (water flux from upper to lower reservoir of the flow routing module) 6: runoff from upper reservoir 7: runoff from lower reservoir

References

Seibert,J.(1997)."Estimation of Parameter Uncertainty in the HBV Model". Nordic Hydrology.28(4/5).247-262.

Kollat,J.B.,Reed,P.M.,Wagener,T.(2002)."When are multiobjective calibration trade-offs in hydrologic models meaningful?". Water resources research, VOL.48, W03520.

`hymod_MulObj` *Hymod RMSE and BIAS*

Description

This function runs the rainfall-runoff Hymod model and returns 2 metrics of model performance: RMSE and BIAS

Usage

```
hymod_MulObj(x, dat)
```

Arguments

`x`	vector 5 of model parameters `(Smax, beta, alfa, Rs, Rf)`
`dat`	dataset containing `rain` vector `T` time series of rainfall, `evap` vector `T` time series of potential evaporation, `flow` vector `T` time series of observed flow.

Value

`Y` vector `(2)` of objective functions `(RMSE, BIAS)`

See Also

`hymod_sim`

`hymod_nse` *Hymod Nash-Sutcliffe Efficiency*

Description

This function runs the rainfall-runoff Hymod model and returns the associated Nash-Sutcliffe Efficiency

Usage

```
hymod_nse(x, dat)
```

Arguments

`x`	vector 5 of model parameters `(Smax, beta, alfa, Rs, Rf)`
`dat`	dataset containing `rain` vector `T` time series of rainfall, `evap` vector `T` time series of potential evaporation, `flow` vector `T` time series of observed flow.

Value

y Nash-Sutcliffe Efficiency

See Also

`hymod_sim`

`hymod_sim` *Hymod rainfall-runoff model*

Description

This function simulates the Hymod rainfall-runoff model [Boyle, 2001; Wagener et al., 2002]

Usage

```
hymod_sim(rain, evap, param)
```

Arguments

`rain`	vector (`T`) time series of rainfall
`evap`	vector (`T`) time series of potential evaporation
`param`	5 elements vector of model parameters (`Smax, beta, alfa, Rs, Rf`)

References

Boyle, D. (2001). Multicriteria calibration of hydrological models. PhD thesis, Dep. of Hydrol. and Water Resour., Univ. of Ariz., Tucson.

Wagener, T., Boyle, D., Lees, M., Wheater, H., Gupta, H., and Sorooshian, S. (2001). A framework for development and application of hydrological models. Hydrol. Earth Syst. Sci., 5, 13-26.

`ishigami_homma_function` *Ishigami Homma function*

Description

This function does Ishigami Homma

Usage

```
ishigami_homma_function(x)
```

Arguments

`x`	vector (`M`) of inputs

Value

`Y` matrix (`N, P`) of associated model ouputs (`P` being the number of scalar model outputs associated to each sampled input combination), `comp_time` total computing time for model evaluation (sec)

Examples

```
# See the demo
# demo("workflow_visual_ishigami_homma")
# or
# demo("workflow_vbsa_ishigami_homma")
```

`LeafCatch` *Leaf Catch*

Description

Leaf Catch daily rainfall, evaporation and flow.

Format

Data frame with 14610 rows and 3 columns

Details

- rain vector `(T)` time series of rainfall (mm/h).
- evaporation vector `(T)` time series of potential evaporation (mm/h).
- flow vector `T` time series of observed flow (m3/s).

Examples

```
data(LeafCatch)
```

`model_evaluation` *Model evaluation*

Description

This function does model evaluation

Usage

```
model_evaluation(fun_test, X, ...)
```

Arguments

`fun_test`	name of the function implementing the model (string)
`X`	matrix `(N, M)` of `N` sampled input factors
`...`	other parameters to be passed in `fun_test`

Value

`Y` matrix `(N, P)` of associated model ouputs (`P` being the number of scalar model outputs associated to each sampled input combination), `comp_time` total computing time for model evaluation (sec)

Morris_orientation_matrix *Morris orientation matrix*

Description

This function builds a Morris orientation matrix

Usage

```
Morris_orientation_matrix(k, p)
```

Arguments

k	integer positive scalar, number of inputs
p	integer positive even scalar, number of levels

Value

Bstar matrix of (k + 1) datapoints in the k-dimensional input space (to be used for computing one Elementary Effect for each of the k inputs)

Examples

```
# Example in two-dimensional space (k = 2):
#
p <- 4
plot(1, xlim = c(0,1), ylim = c(0, 1), type ="n", xlab = "", ylab = "")
Bstar <- Morris_orientation_matrix(2,p)
lines(Bstar[,1], Bstar[,2], col = "red")
points(Bstar[,1],Bstar[,2], col = "red", pch = 20)
# if you want to generate more datapoints:
Bstar <- Morris_orientation_matrix(2,p)
lines(Bstar[,1], Bstar[,2], col = "red")
points(Bstar[,1],Bstar[,2], col = "red", pch = 20)
Bstar <- Morris_orientation_matrix(2,p)
lines(Bstar[,1], Bstar[,2], col = "red")
points(Bstar[,1],Bstar[,2], col = "red", pch = 20)
Bstar <- Morris_orientation_matrix(2,p)
lines(Bstar[,1], Bstar[,2], col = "red")
points(Bstar[,1],Bstar[,2], col = "red", pch = 20)
```

Morris_sampling *Morris Sampling*

Description

This function builds a matrix X of input samples to be used for the Elementary Effects Test, using a One-At-the-Time sampling strategy as described in Campolongo et al. (2011).

Usage

```
Morris_sampling(r, xmin, xmax, L)
```

Arguments

`r`	positive integer number, number of elementary effects.
`xmin`	vector `(M)`. Lower bounds of input ranges.
`xmax`	vector `(M)`. Upper bounds of input ranges.
`L`	positive, even number. Number of levels in the sampling grid

Value

`X` matrix of sampling datapoints where `EE` must be computed. This is a matrix with `r * (M + 1)` rows and `M` columns. Each row is a point in the input space. Rows are sorted in `r` blocks, each including `M + 1` rows. Within each block, points (rows) differ in one component at the time. Thus, each block can be used to compute one Elementary Effect `(EE_i)` for each model input `(i = 1, ...,M)`.

References

Morris, M.D. (1991), Factorial sampling plans for preliminary computational experiments, Technometrics, 33(2).

`OAT_sampling` *One-At-the-Time sampling strategy*

Description

This function builds a matrix `X` of input samples to be used for the Elementary Effects Test, using a One-At-the-Time sampling strategy as described in Campolongo et al. (2011).

Usage

```
OAT_sampling(r, M, distr_fun, distr_par, samp_strat, des_type)
```

Arguments

`r`	positive integer number , number of elementary effects
`M`	positive integer number , number of inputs
`distr_fun`	probability distribution function of each input. list (eg: `"unif"`) if all inputs have the same pdf or a list of length `M` strings (eg: `list("unif", "norm")`) otherwise. See help of `AAT_sampling` to check supported PDF types.
`distr_par`	parameters of the probability distribution function - row vector if all input pdfs have the same parameters - list of `M` vectors otherwise
`samp_strat`	sampling strategy - string Options: `"rsu"`: random uniform, `"lhs"`: latin hypercube.
`des_type`	design type - string. Options: `"trajectory"`, `"radial"`

Value

X matrix of sampling datapoints where EE must be computed. This is a matrix with r * (M + 1) rows and M columns. Each row is a point in the input space. Rows are sorted in r blocks, each including M + 1 rows. Within each block, points (rows) differ in one component at the time. Thus, each block can be used to compute one Elementary Effect (EE_i) for each model input (i = 1, ...,M).

References

Campolongo F., Saltelli, A. and J. Cariboni (2011), From screening to quantitative sensitivity analysis. A unified approach, Computer Physics Communications, 182(4), 978-988.

See Also

AAT_sampling

Examples

```
# Example 1: 2 inputs, both from Unif[0,3]
r <-  10
M <-  2
distr_fun <-  "unif"
distr_par <-  c(0, 3)
samp_strat <-  "lhs"
des_type <-  "trajectory"
X <-  OAT_sampling(r, M, distr_fun, distr_par, samp_strat, des_type)
# Plot results:
plot(X[,1], X[,2], col = rep(rainbow(r), each = M + 1), pch = 19,
xlab = expression(x[1]), ylab = expression(x[2]))
for(k in 0:(r-1)){
 segments(X[c(1,3) + (M+1) * k, 1], X[c(1,3) + (M+1) * k, 2],
X[2 + (M+1) * k, 1], X[2 + (M+1) * k, 2], lty = 2, col = "gray")
}
# Example 2: 2 inputs, one from Unif[0,3], one from Unif[1,5]
distr_fun <-  "unif"
distr_par <- list(c(0, 3), c(1, 5))
X <-  OAT_sampling(r, M, distr_fun, distr_par, samp_strat, des_type)
plot(X[,1], X[,2], col = rep(rainbow(r), each = M + 1), pch = 19,
xlab = expression(x[1]), ylab = expression(x[2]))
for(k in 0:(r-1)){
 segments(X[c(1,3) + (M+1) * k, 1], X[c(1,3) + (M+1) * k, 2],
X[2 + (M+1) * k, 1], X[2 + (M+1) * k, 2], lty = 2, col = "gray")
}
```

---

plot_convergence *Plot convergence Draws the plot to analise the convergence*

Description

Plot convergence Draws the plot to analise the convergence

Usage

```
plot_convergence(n, est, estl = NULL, estu = NULL, ex = NULL,
  labels = NULL, ylim = NULL, col = 1:M, lty = 1:(M - 1), ...)
```

Arguments

`n`	vector `P` of number of model evaluations
`est`	matrix `(P, M)` with the mean of the indices.
`estl`	matrix `(P, M)` with the lower bound of the estimates of the indices. Default `estl = NULL`
`estu`	matrix `(P, M)` with the lower bound of the estimates of the indices. Default `estu = NULL`
`ex`	exact values to be drawn as a line. Default `ex = NULL`
`labels`	labels for legend. Default `labels = NULL`
`ylim`	limits of the y-axis. Default `ylim = NULL`
`col`	colors for the plot. Default `col = 1:M`
`lty`	line type. Default `lty = 1:(M-1)`
`...`	others arguments to be passed in `plot`

Examples

```
# Setup the model and define input ranges
fun_test  <- "ishigami_homma_function"
M <- 3
distr_fun <- "unif"
distrpar <-  c(-pi, pi)
# Compute the exact values of the output variance (V) and of
# the first-order (Si_ex) and total-order (STi_ex)
# variance-based sensitivity indices (this is possible in this
# very specific case because V, Si_ex and STi_ex can be computed
# analytically)
ihfun <- ishigami_homma_function(runif(M))
Si_ex <- attributes(ihfun)$Si_ex
STi_ex <- attributes(ihfun)$STi_ex
# Sample parameter space:
SampStrategy <- "lhs"
N <- 3000
X <- AAT_sampling(SampStrategy, M, distr_fun, distrpar, 2 * N)
# Apply resampling strategy for the efficient approximation of the indices:
XABC <- vbsa_resampling(X)
# Run the model and compute selected model output at sampled parameter
# sets:
YA <- model_evaluation(fun_test, XABC$XA) # size (N,1)
YB <- model_evaluation(fun_test, XABC$XB) # size (N,1)
YC <- model_evaluation(fun_test, XABC$XC) # size (N*M,1)
# Analyze convergence of sensitivity indices:
NN <- seq(N / 5, N, by = N/5)
conv <- vbsa_convergence(c(YA, YB, YC), M, NN)
Si <- conv$Si
STi <- conv$STi
par(mfrow = c(1, 2))
plot_convergence(NN * (M + 2), Si, ex = Si_ex,
xlab = "model evals", ylab = "main effect")
plot_convergence(NN * (M + 2), STi, ex = STi_ex,
xlab = "model evals", ylab = "total effect")
# With bootstrap
Nboot <- 100
conv100 <- vbsa_convergence(c(YA, YB, YC), M, NN, Nboot)
Si100 <- conv100$Si
STi100 <- conv100$STi
Sil100 <- conv100$Si_lb
Siu100 <- conv100$Si_ub
STil100 <- conv100$STi_lb
STiu100 <- conv100$STi_ub
par(mfrow = c(1, 2))
plot_convergence(NN * (M + 2), Si100, Sil100, Siu100, ex = Si_ex,
xlab = "model evals", ylab = "main effect")
plot_convergence(NN * (M + 2), STi100, STil100, STiu100, ex = STi_ex,
xlab = "model evals", ylab = "total effect")
```

RSA_convergence_thres *Regional Sensitivity Analysis (with threshold)*

Description

This function computes and plots the sensitivity indices obtained by RSA_indices_thres using an increasing number of output samples.

Usage

```
RSA_convergence_thres(X, Y, NN, threshold = NULL, flag = 1, Nboot = 0,
  alfa = 0.05)
```

Arguments

X	matrix (N, M) set of inputs samples
Y	matrix (N, P) set of output samples
NN	vector (R) of subsample sizes at which indices will be estimated (max(NN) must not exceed N)
threshold	vector (P) threshold for output values. Default value: threshold = median(Y)
flag	scalar specify the statistic to assess the distance between CDFs flag = 1: stat maximum vertical distance (see mvd below), flag = 2: stat area between curves (see spread below) flag = 3: stat input range reduction (see irr below) Default value: flag = 1
Nboot	scalar, number of resamples used for boostrapping. Default Nboot = 0, i.e. no bootstrapping.
alfa	scalar significance level for the confidence intervals estimated by bootstrapping. Default: 0.05

Value

List containing:

- `stat` distance measure between CDFs for each input vector `(M)`.

If `Nboot > 1` it also contains

- `stat_lb` lower bound of vector `(stat)` from bootstrapping vector `(M)`
- `stat_ub` upper bound of `stat` from bootstrapping vector `(M)`

See Also

`RSA_plot_thres` `RSA_indices_thres`

Examples

```
# See the demo
# demo("workflow_rsa_hymod")
```

`RSA_indices_groups` *Regional Sensitivity Analysis (with grouping)*

Description

Computation function for Regional Sensitivity Analysis (with grouping). It splits the samples in a dataset X into `ngroups` sub-sets corresponding to `ngroup` equally spaced values of `Y` (as first proposed by Wagener et al., 2001). Then assess the distance between the CDFs of `X` in the different datasets by a statistic (maximum or median) of the maximum vertical distance between the CDFs, i.e.

Usage

```
RSA_indices_groups(X, Y, ngroup = 10, flag = 1, Nboot = 0, alfa = 0.05)
```

Arguments

`X`	matrix `(N, M)` set of inputs samples
`Y`	matrix `(N, P)` set of output samples
`ngroup`	number of groups considered (default: 10)
`flag`	scalar, 1 or 2. Statistic for the definition of the RSA index. `flag = 1`: median (default), `flag = 2`: maximum (see 'spread' below)
`Nboot`	scalar, number of resamples used for boostrapping. Default `Nboot = 0`, i.e. no bootstrapping.
`alfa`	scalar significance level for the confidence intervals estimated by bootstrapping. Default: `0.05`

Details

$stat = max(max_x(|Fi(x) - Fj(x)|))$

or

$stat = median(max_x(|Fi(x) - Fj(x)|))$

where $Fi()$ is the CDF of `X` in i-th dataset and $Fj()$ is the CDF in the j-th dataset.

Value

List containing:

- `stat` vector `(M)` distance measure between CDFs for each input
- `idxb` vector `N` respective group of the samples
- `Yk` vector `ngroup + 1` range of `Y` in each group

You can easily derive the n_groups datasets `Xi` as: `Xi = X[idx == i]`

If `Nboot > 1` it also contains

- `stat_lb` vector `(M)` lower bound of vector `(stat)` from bootstrapping
- `stat_ub` vector `(M)` upper bound of `stat` from bootstrapping

References

Wagener, T., Boyle, D. P., Lees, M. J., Wheater, H. S., Gupta, H. V., and Sorooshian, S. (2001): A framework for development and application of hydrological models, Hydrol. Earth Syst. Sci., 5, 13-26.

See Also

`RSA_plot_groups` `RSA_indices_thres`

Examples

```
# See the demo
# demo("workflow_rsa_hymod")
```

`RSA_indices_thres` *Regional Sensitivity Analysis (with threshold)*

Description

Computation function for Regional Sensitivity Analysis (with threshold). It splits the samples in a dataset `X` into two datasets (`Xb` and `Xnb`) depending on whether the associated sample in Y satisfies the condition:$Y(i,j) < threshold(j)$ for $j = 1, ..., P$. Then assess the distance between the CDFs of `Xb` and `Xnb` by a suitable measure (maximum vertical distance, area between curves, etc.). Use the function `RSA_plot_thres` to visualize results.

Usage

```
RSA_indices_thres(X, Y, threshold = NULL, flag = 1, Nboot = 0,
  alfa = 0.05)
```

Arguments

`X`	matrix `(N, M)` set of inputs samples
`Y`	matrix `(N, P)` set of output samples
`threshold`	vector `(P)` threshold for output values. Default value: `threshold = median(Y)`
`flag`	scalar specify the statistic to assess the distance between CDFs `flag = 1`: `stat` maximum vertical distance (see `mvd` below), `flag = 2`: `stat` area between curves (see `spread` below) `flag = 3`: `stat` input range reduction (see `irr` below) Default value: `flag = 1`
`Nboot`	scalar, number of resamples used for boostrapping. Default `Nboot = 0`, i.e. no bootstrapping.
`alfa`	scalar significance level for the confidence intervals estimated by bootstrapping. Default: `0.05`

Value

List containing:

- `stat` vector `(M)` distance measure between CDFs for each input
- `idxb` vector `N` indices of samples statisfying the condition.

You can easily derive the two datasets `Xb` and `Xnb` as: `Xb = X[idxb, ]`, `Xnb = X[!idxb, ]`. If `Nboot > 1` it also contains

- `stat_lb` vector `(M)` lower bound of vector `(stat)` from bootstrapping
- `stat_ub` vector `(M)` upper bound of `stat` from bootstrapping vector

See Also

`RSA_plot_thres` `RSA_convergence_thres`

Examples

```
# See the demo
# demo("workflow_rsa_hymod")
```

`RSA_plot_groups` *Plotting function for Regional Sensitivity Analysis with grouping*

Description

Plotting function for Regional Sensitivity Analysis with grouping. Plot `Ng` CDFs of the samples in `X` with different colours.

Usage

```
RSA_plot_groups(X, idx, Yk, n_col = 5, labels = NULL, col = rainbow(1:Ng),
  ...)
```

Arguments

`X`	matrix `(N, M)` set of input samples
`idx`	vector `(N)` index of group to which input samples belong
`Yk`	vector `(Ng + 1)` range of `Y` in each group
`n_col`	scalar number of panels per row in the plot (default: `min(5, M)`)
`labels`	vector `(M)` labels for the horizontal axis (default: `c("X1", "X2",...)`)
`col`	color of the lines in the plot (default `rainbow(Ng)`)
`...`	other

See Also

`RSA_indices_groups` `RSA_plot_thres`

Examples

```
# See the demo
# demo("workflow_rsa_hymod")
```

`RSA_plot_thres` *Plotting function for Regional Sensitivity Analysis*

Description

This function plots Regional Sensitivity Analysis

Usage

```
RSA_plot_thres(X, idxb, n_col = 5, labels = NULL, str_legend = NULL)
```

Arguments

`X`	matrix `(N, M)` set of input samples
`idxb`	vector `(N)` indices of samples statisfying the condition
`n_col`	scalar number of panels per row in the plot (default: `min(5,M)`)
`labels`	vector `(M)` labels for the horizontal axis (default: `c("#1", "#2",...)`)
`str_legend`	vector (2) text for legend (default: no legend)

See Also

`RSA_indices_thres` `RSA_convergence_thres`

Examples

```
# See the demo
# demo("workflow_rsa_hymod")
```

scatter_plots *Scatter plots of* y *against* X

Description

This function produces scatter plots of the model ouput y against model inputs $x(1), x(2), ..., x(M)$

Usage

```
scatter_plots(X, Y, ...)
```

Arguments

X	matrix (N, M) of N inputs samples
Y	vector N of associated ouput samples
...	parameters to be passed in the plot see plot

Examples

```
##############################
# Step 1 (setup the model)
#############################
fun_test  <- "ishigami_homma_function"
M <- 3
distr_fun <- "unif"
distrpar <-  c(-pi, pi)
# ############################
# Step 2 (sampling and model evaluation)
# ############################
N <- 3000
X <- AAT_sampling("lhs", M, distr_fun, distrpar, N)
Y <- model_evaluation(fun_test, X)
# ############################
## Step 3 (Scatter plots)
# ############################
# Use scatter plots of inputs againts output to visually assess
# direct effects:
scatter_plots(X,Y)
```

scatter_plots_interaction *scatter plots of x(i) against all other x(j)*

Description

This function produces scatter plots of input $x(i)$ against all other inputs $x(j) (j = 1, ..., M; j! = i)$, where the color of the marker is proportional to the value of the model output y

Usage

```
scatter_plots_interaction(X, Y, col = NULL, ...)
```

Arguments

`X`	matrix `(N, M)` of `N` inputs samples
`Y`	vector `(N)` of associated ouput samples
`col`	vector of length two of column interactions. Default `col = NULL`, all the possible combinations are plotted.
`...`	parameters to be passed in the plot see `plot`

Examples

```
##############################
# Step 1 (setup the model)
#############################
fun_test  <- "ishigami_homma_function"
M <- 3
distr_fun <- "unif"
distrpar <-  c(-pi, pi)
# #############################
# Step 2 (sampling and model evaluation)
# #############################
N <- 3000
X <- AAT_sampling("lhs", M, distr_fun, distrpar, N)
Y <- model_evaluation(fun_test, X)
# #############################
# Step 3 (Scatter plots)
# #############################
# Use coloured scatter plots of one input against another on to assess
# interactions:
# plot x(i1) against x(i3)
dev.new()
scatter_plots_interaction(X, Y, col = c(1, 3))
# Put all possible combinations into one figure:
# Customize titles:
dev.new()
colnames(X) <- c("x(1)", "x(2)", "x(3)")
scatter_plots_interaction(X, Y)
```

`sobol_g_function` *Sobol' g-function*

Description

Implements the Sobol' g-function, a standard benchmark function in the Sensitivity Analysis literature (see for instance Sec. 3.6 in Saltelli et al. (2008)).

Usage

```
sobol_g_function(x, a)
```

Arguments

`x`	vector (`M`) of inputs $x(1), x(2), ...x(M)$. $x(i)$ ~ $Unif(0, 1)$ for all i
`a`	vector (`M`) or scalar

Value

y scalar output. Two attributes `V` scalar output variance (exact value computed analytically), `Si_ex` vector (3), first-order sensitivity indices (exact value computed analytically)

References

Saltelli et al. (2008) Global Sensitivity Analysis, The Primer, Wiley.

Examples

```
a <- 9 # options for the (fixed) parameters
M <- 5 # options for the number of inputs
y <- sobol_g_function(runif(M), a)
Si_ex <- attributes(y)$Si_ex
```

`vbsa_convergence` *Variance-based first-order and total effects indices*

Description

This function computes the variance-based first-order indices (or main effects) and total effects indices (Homma and Saltelli, 1996) using an increasing number of output samples.

Usage

```
vbsa_convergence(Y, M, NN, Nboot = 0, alfa = 0.05)
```

Arguments

`Y`	vector of `N * (M + 2)` model output samples `Y = c(YA, YB, YC)`
`M`	scalar number of inputs
`NN`	vector `(R)`, sample sizes at which indices will be estimated (must be a vector of integer positive values not exceeding `N`)
`Nboot`	scalar, number of resamples used for boostrapping (default: 0)
`alfa`	scalar, significance level for the confidence intervals estimated by bootstrapping (default: 0.05)

Value

List containing:

- `Si` estimates of the main effects at different sampling size
- `STi`estimates of the total effects at different sampling size.

If `Nboot > 1` it also contains

- `Si_sd` standard deviation of main effects at different sampling size
- `STi_sd` standard deviation of total effects at different sampling size
- `Si_lb` lower bound of main effects at different sampling size
- `STi_lb` lower bound of total effects at different sampling size

- `Si_ub` upper bound of main effects at different sampling size
- `STi_ub` upper bound of total effects at different sampling size.

All output arguments are matrices of size `(R, M)`

See Also

`vbsa_indices` `vbsa_resampling`

Examples

```
fun_test  <- "ishigami_homma_function"
M <- 3
distrfun <- "unif"
distrpar <- c(-pi, pi)
N <- 1000
sampstrat <- "lhs"
X <- AAT_sampling(sampstrat, M, distrfun, distrpar, 2 * N)
XABC <- vbsa_resampling(X)
YA <- model_evaluation(fun_test, XABC$XA)
YB <- model_evaluation(fun_test, XABC$XB)
YC <- model_evaluation(fun_test, XABC$XC)
Y <- c(YA, YB, YC)
NN <- seq(N/10, N, by = N / 10)
SiSTi <- vbsa_convergence(Y, M, NN)
```

`vbsa_indices` *Variance-based first-order indices and total effects indices*

Description

This function computes the variance-based first-order indices (or main effects) and total effects indices (Homma and Saltelli, 1996). The indices are approximated by the estimator suggested e.g. in Saltelli et al. (2008) and (2010).

Usage

```
vbsa_indices(YA, YB, YC, Nboot = 0, alfa = 0.05)
```

Arguments

`YA`	vector `(N)`, set of output samples
`YB`	vector `(N)`, set of output samples (independent from `YA`)
`YC`	vector `(N*M)`, set of output samples from resampling (*)
`Nboot`	scalar, number of resamples used for boostrapping (default: 0)
`alfa`	scalar, significance level for the confidence intervals estimated by bootstrapping (default: 0.05)

Details

NOTES:

(*) By default, here we use the estimators described by Saltelli et al. (2008) and Saltelli et al. (2010) (see comments in the code for specific references to the equations implemented here!). These are obtained from 3 sets of output samples (`YA`, `YB` and `YC`), which are obtained by model evaluation against three input matrices `XA`, `XB` and `XC` generated by the `vbsa_resampling` function: - see example below about how to use `vbsa_resampling` - see help of `vbsa_resampling` to learn more about `XA`, `XB` and `XC`.

(**) If bootstrapping is used, `Si` and `STi` are the average of the respective `Nboot` estimates obtained at each bootstrap resampling.

Value

List containing:

- `Si` estimates of the main effects at different sampling size
- `STi` estimates of the total effects at different sampling size

If `Nboot > 1` it also contains

- `Si_sd` standard deviation of main effects at different sampling size
- `STi_sd` standard deviation of total effects at different sampling size
- `Si_lb` lower bound of main effects at different sampling size
- `STi_lb` lower bound of total effects at different sampling size
- `Si_ub` upper bound of main effects at different sampling size
- `STi_ub` upper bound of total effects at different sampling size.

All output arguments are matrices of size `(R, M)`

References

Homma, T. and A., Saltelli (1996). Importance measures in global sensitivity analysis of nonlinear models. Reliability Engineering & System Safety, 52(1), 1-17.

Saltelli et al. (2008), Global Sensitivity Analysis, The Primer, Wiley.

Saltelli et al. (2010), Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index, Computer Physics Communications, 181, 259-270.

See Also

`vbsa_resampling`

Examples

```
fun_test  <- "ishigami_homma_function"
M <- 3
distrfun <- "unif"
distrpar  <- c(-pi, pi)
N <- 1000
sampstrat <- "lhs"
X <- AAT_sampling(sampstrat, M, distrfun, distrpar, 2 * N)
XABC <- vbsa_resampling(X)
YA <- model_evaluation(fun_test, XABC$XA)
```

```
YB <- model_evaluation(fun_test, XABC$XB)
YC <- model_evaluation(fun_test, XABC$XC)
SiSTi <- vbsa_indices(YA,YB,YC)
```

`vbsa_resampling` *Resampling strategy needed to build the approximators of the first-order and total order sensitivity indices*

Description

This function implements the resampling strategy needed to build the approximators of the first-order (main effects) and total order sensitivity indices (e.g. Saltelli et al. 2008; 2010). This function is meant to be used in combination with `vbsa_indices`.

Usage

```
vbsa_resampling(X)
```

Arguments

`X`	matrix `(NX, M)` of `NX` input samples

Value

List containing:

- `XA` matrix `(N, M)` first `N = NX / 2` rows of `X`
- `XB` matrix `(N, M)` last `N = NX / 2` rows of `X`
- `XC` matrix `(N*M, M)` Block matrix of `M` recombinations of `XA` and `XB`, each block `XCi` is a `(N, M)` matrix whose columns are all taken from `XB` exception made for `i`-th, which is taken from `XA`.

References

Saltelli et al. (2008), Global Sensitivity Analysis, The Primer, Wiley.

Saltelli et al. (2010), Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index, Computer Physics Communications, 181, 259-270.

See Also

`vbsa_indices`

Examples

```
fun_test  <- "ishigami_homma_function"
M <- 3
distrfun <- "unif"
distrpar  <- c(-pi, pi)
N <- 1000
sampstrat <- "lhs"
X <- AAT_sampling(sampstrat, M, distrfun, distrpar, 2 * N)
XABC <- vbsa_resampling(X)
```

Index