Package 'LSDsensitivity'

May 7, 2026

Type Package

Title Sensitivity Analysis Tools for 'LSD' Simulations

Version 1.3.2

Date 2026-4-3

Description Tools for sensitivity analysis of 'LSD' simulation models. Reads object-oriented data produced by 'LSD' simulation models and performs screening and global sensitivity analysis (Sobol decomposition method, Saltelli et al. (2008) ISBN:9780470725177). A Kriging or polynomial meta-model (Kleijnen (2009) <doi:10.1016/j.ejor.2007.10.013>) is estimated using the simulation data to provide the data required by the Sobol decomposition. 'LSD' (Laboratory for Simulation Development) is free software developed by Marco Valente and Marcelo C. Pereira (documentation and downloads available at <https://www.labsimdev.org/>).

Depends R (>= 3.2.0)

Imports LSDinterface (>= 1.2.1), stats, utils, graphics, tseries, kSamples, diptest, lawstat, abind, sensitivity, car, randtoolbox, parallel, rgenoud, DiceKriging, XML

Suggests gplots, rgl, normalp

License GPL-3

Language en-US

Encoding UTF-8

NeedsCompilation no

Author Marcelo C. Pereira [aut, cre] (ORCID: <https://orcid.org/0000-0002-8069-2734>)

Maintainer Marcelo C. Pereira <mcper@unicamp.br>

Repository CRAN

Date/Publication 2026-04-04 06:20:13 UTC

Contents

`LSDsensitivity-package`

Sensitivity Analysis Tools for 'LSD' Simulations

Description

Tools for sensitivity analysis of 'LSD' simulation models. Reads object-oriented data produced by 'LSD' simulation models and performs screening and global sensitivity analysis (Sobol decomposition method, Saltelli et al. (2008) ISBN:9780470725177). A Kriging or polynomial meta-model (Kleijnen (2009) <doi:10.1016/j.ejor.2007.10.013>) is estimated using the simulation data to provide the data required by the Sobol decomposition. 'LSD' (Laboratory for Simulation Development) is free software developed by Marco Valente and Marcelo C. Pereira (documentation and downloads available at <https://www.labsimdev.org/>).

Details

The LSDsensitivity R package provides tools to analyze simulated experiments from **LSD**. LSD offers native tools to sample the configuration (factor) space of a simulation model using different design of experiments (DoE). The produced experimental design data can be transparently imported to R by using the function `read.doe.lsd()`.

The package offers two sensitivity analysis (SA) methods (`elementary.effects.lsd()` and `sobol.decomposition.lsd()`) pre-configured for application on LSD simulations: Morris Elementary Effects (EE) and Sobol Variance Decomposition (SVD).

EE (`elementary.effects.lsd()`) employs a simple one-factor-at-a-time (OAT) SA and is usually applied as an initial screening method while selecting relevant factors to a SVD global SA. EE requires an appropriate set of sample points (the DoE) which can be generated in LSD when "EE Sampling" is selected in the "Data" menu. Please make sure to take note of the DoE parameters used for the sampling, as they will be required for the configuration of the R analysis script.

Due to its high computational cost, `sobol.decomposition.lsd()` (SVD) is performed over a meta-model fitted from the experimental data produced by the LSD original model. The meta-model can be fitted using different sampling strategies offered by LSD, being "NOLH Sampling" (Near Orthogonal Latin Hypercube) usually the most efficient. Additionally to the set of samples used to fit the meta-model, it is recommended to also generate another set for the (external) validation of the meta-model ("MC Range Sampling" is the recommended option).

The package offers two meta-modeling (MM) methods for using with SVD: Kriging and polynomial. Kriging (`kriging.model.lsd()`) is offered under five different variance kernels (Matern 5/2, Matern3/2, Gaussian, exponential and power exponential) and two trend models (constant or first order polynomial) to choose, including auto-selection to the best fitting alternative. Polynomial meta-models of first or second order, with or without interactions, and auto-selection are also offered (`polynomial.model.lsd()`). Kriging is the recommended option in most cases.

Additionally, the package offers tools for the graphical representation of the meta-models response surfaces (2D and 3D) (`response.surface.lsd()`), to predict meta-model response in specific points in the factor space (`model.pred.lsd()`), to identify maximum and minimum responses from a set of factors (`model.limits.lsd()`), and to find optimal parameter settings using the meta-model (`model.optim.lsd()`).

For a complete list of exported functions, use `library(help = "LSDsensitivity")`.

LSD 7.0+ default installation provides *example scripts* for the usage of the LSDsensitivity package. LSD can be downloaded at https://github.com/marcov64/Lsd/. They can also be retrieved from the package itself using the commands:

EE example: `file.show(system.file("examples", "elementary-effects-SA.R", package = "LSDsensitivity"))`

Kriging SVD example: `file.show(system.file("examples", "kriging-sobol-SA.R", package = "LSDsensitivity"))`

Polynomial SVD example: `file.show(system.file("examples", "poly-sobol-SA.R", package = "LSDsensitivity"))`

Optimize MM example: `file.show(system.file("examples", "optimize-MM.R", package = "LSDsensitivity"))`

Note

Below are the minimum required steps to perform SA on a working LSD model using NOLH sampling, Kriging MM and SVD. The changes to perform an EE or to use a polynomial MM are also indicated, as options.

1. Define the parameters/initial values to be explored in the SA, their max/min ranges and the result variables over which the SA is to be done
2. In `LMM` create a no-window (command prompt) version of your model by selecting menu `Model/Create 'No Window' Version`
3. In `LSD Browser` make sure that all parameters/initial values are set with the correct calibration/default values (menu `Data/Initial Values`), the required result variables are being saved (menu `Model/Change Element...`, click on `Save/OK` or simply `Save` in the right mouse button context menu) and the number of MC runs for each SA sample (point) is defined (menu `Run/Simulation Settings`, `Number of simulation runs` field, typically set to 10)
4. Save your setup in a baseline `.lsd` configuration file (menu `File/Save As...`), preferably in a new folder inside your current model configuration folder (you can create a new folder while in the `File/Save As...` dialog box, if you do not, LSD will create a new folder when saving SA configuration and results files, named as your baseline configuration)
5. (Re)load your baseline `.lsd` configuration if it is not already loaded (menu `File/Load...`)

6. Choose the ranges (max/min) for each parameter/initial value in your SA exploration space by using the Sensitivity Analysis button in the menu Model/Change Element... window or the same option in the context menu (mouse right-button click on the parameter/variable name in the Variables & Parameters list box)
7. After choosing all ranges, save your exploration space definition as a .sa sensitivity analysis file using the same base name and folder as your .lsd baseline configuration (menu File/Save Sensitivity...)
8. With both the created .lsd and .sa files loaded (use menu File/Load... and File/Load Sensitivity... if required), select Data/Sensitivity Analysis/NOLH Sampling... and accept the defaults (several new .lsd files will be created in your just created baseline configuration folder (or one created by LSD, if you did not), those are the sample points for the meta-model estimation)
   (a) To perform Elementary Effects (EE) analysis instead of Sobol Variance Decomposition, in the step below select Data/Sensitivity Analysis/EE Sampling... instead (NOLH sampling cannot be used for EE)
   (b) If a polynomial meta-model (MM) is being estimated, sometimes it is preferred to use Data/Sensitivity Analysis/MC Range Sampling... despite not required
9. Immediately after the previous step, select menu Data/Sensitivity Analysis/MC Range Sampling... and accept the defaults (to create the external validation sample, more .lsd files will be created for the additional sampling points)
   (a) EE analysis does not uses external validation, so skip this step for EE
10. Immediately after the previous step select menu Run/Create/Run Parallel Batch, accept using the just created configuration, adjust the number of cores only if going to run in another machine (8 in a modern PC, 20 in a basic server), and decide if you want to start the (time-consuming) processing right now or later (in the current or in another machine)
11. If running later in the same machine, you just have to execute the created script file (.bat or .sh) inside the new folder your baseline .lsd file was created (or the one LSD created if you did not)
12. If running in another machine, you have to copy the entire model folder and sub-folders to the new machine (the remaining LSD folders are not required), recompile LSD for the new platform if required and execute the script file (.bat or .sh) in the same folder as your baseline .lsd file
13. Open R (or RStudio) and check you have the following packages installed and download them if required (if you install LSDsensitivity from CRAN or another online repository, and not from a file, all other dependency packages should be automatically downloaded):
    LSDsensitivity, LSDinterface, abind, tseries, car, minqa, nloptr, Rcpp, RcppEigen, lme4, SparseM, MatrixModels, pbkrtest, quantreg, DiceKriging, kSamples, SuppDists, randtoolbox, rngWELL, rgenoud, sensitivity, xts, TTR, quadprog, zoo, quantmod
14. Open the kriging-sobol-SA.R example script (included in your LSD installation folder) in RStudio or your text editor
   (a) For EE analysis, open elementary-effects-SA.R instead
   (b) For the use of a polynomial MM for the SVD analysis, open poly-sobol-SA.R instead
15. Adjust the vector lsdVars to contain all the LSD saved variables you want to use in your analysis (do not include saved but unused variables, for performance reasons), replacing the dummies varX

16. Adjust the vector `logVars` to contain all LSD variables (included in `lsdVars`) that require to have the log value used in the analysis (let the vector empty, i.e. `c()`, if no log variable is required)
17. Include in the vector `newVars` any new variable (not included in `lsdVars`) that has to be added to the dataset (let the vector empty, i.e. `c()`, if no new variable is required)
18. Adapt the `eval.vars()` function to compute any new variable included in `newVars` (use the commented example as a reference)
19. Adjust the arguments to the function `read.doe.lsd()` for the relative folder of LSD data files (default is same as R working directory), the data files base name (the file name chosen for the baseline configuration in step 4 without the `.lsd` suffix) and the name of the variable to be used as reference for the sensitivity analysis (you have to run the script multiple times if there is more than one)
20. Save the modified script, renaming if necessary, and run it in R (or click the `Source` button in RStudio), redirecting output to a file first if required

Author(s)

Marcelo C. Pereira [aut, cre] (ORCID: <https://orcid.org/0000-0002-8069-2734>)

Maintainer: Marcelo C. Pereira <mcper@unicamp.br>

References

LSD documentation is available at https://www.labsimdev.org/

The latest LSD binaries and source code can be downloaded at https://github.com/marcov64/Lsd/.

Cioppa T, Lucas T (2007) *Efficient nearly orthogonal and space-filling latin hypercubes*. Technometrics 49(1):45-55

Kleijnen JP (2009) *Kriging metamodeling in simulation: a review*. Eur J Oper Res 192(3):707-716

Morris MD (1991) *Factorial sampling plans for preliminary computational experiments*.Technometrics 33(1):161-174

Rasmussen C, Williams C (2006) *Gaussian processes for machine learning*. MIT Press, Cambridge

Roustant O, Ginsbourger D, Deville Y (2012) *Dicekriging, diceoptim: two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization*. J Stat Softw 51(1):1-55

Saltelli A, Ratto M, Andres T, Campolongo F, Cariboni J, Gatelli D, Saisana M, Tarantola S (2008) *Global sensitivity analysis: the primer*. Wiley, New York

Sekhon JS, Walter RM (1998). *Genetic optimization using derivatives: theory and application to nonlinear models*. Political Analysis 7:187-210

See Also

LSDinterface-package

`elementary.effects.lsd`

Elementary effects sensitivity analysis

Description

This function performs the an elementary effects sensitivity analysis on the sample data produced by a LSD simulation model using the Morris (1991) one-at-a-time sampling method.

Usage

```
elementary.effects.lsd( data, p = 4, jump = 2 )
```

Arguments

`data`	an object created by a previous call to `read.doe.lsd` which contains all the required experimental data for the analysis.
`p`	integer: the number of levels of the DoE as set in LSD when the DoE was configured. The default is 4 (also the LSD default).
`jump`	integer: the size of the jump (increase/decrease) for each point change in the DoE as set in LSD when the DoE was configured. The default is 2 (also the LSD default).

Details

The elementary effects analysis statistics are only meaningful if the DoE was created using the Morris design, as when LSD `EE Sampling...` option is used to produce the DoE.

This function is a wrapper to the function `morris` in `sensitivity-package`.

Value

The function returns an object/list of class `morris.lsd` containing several items, among them:

`table`	the elementary effects sensitivity analysis results data.

The returned object can also be directly printed or plotted using `plot()` or any similar function. See the class `morris` for full details, as this class is equivalent to it.

Note

See the note in LSDsensitivity-package for step-by-step instructions on how to perform the complete sensitivity analysis process using LSD and R.

Author(s)

Marcelo C. Pereira [aut, cre] (ORCID: <https://orcid.org/0000-0002-8069-2734>)

References

Morris MD (1991) *Factorial sampling plans for preliminary computational experiments*.Technometrics 33(1):161-174

See Also

read.doe.lsd,

morris in sensitivity-package

Examples

```
# get the example directory name
path <- system.file( "extdata/ee", package = "LSDsensitivity" )

# Steps to use this function:
# 1. define the variables you want to use in the analysis
# 2. load data from a LSD simulation saved results using read.doe.lsd
# 3. perform the elementary effects analysis applying elementary.effects.lsd

lsdVars <- c( "var1", "var2", "var3" )         # the definition of existing variables

dataSet <- read.doe.lsd( path,                 # data files  folder
                         "Sim2",               # data files base name (same as .lsd file)
                    "var1",            # variable name to perform the sensitivity analysis
                         saveVars = lsdVars )  # LSD variables to keep in dataset

SA <- elementary.effects.lsd( dataSet,         # LSD experimental data set
                          p = 4,          # number of levels of the design (as set in LSD)
                             jump = 2 )       # number of jumps per level (as set in LSD)

print( SA )                                    # show analysis table
plot( SA )                                     # plot analysis chart
```

---

ergod.test.lsd *Stationarity and ergodicity tests*

Description

Perform a set of stationarity and ergodicity tests useful for simulation model data from a Monte Carlo experiment time series. The included tests are: Augmented Dickey-Fuller test (ADF), Phillips-Perron test (PP), Kwiatkowski-Phillips-Schmidt-Shin test (KPSS), Brock-Dechert-Scheinkman test (BDS), Kolmogorov-Smirnov k-sample test (KS), Anderson-Darling k-sample test (AD) and Wald-Wolfowitz k-sample test (WW).

Usage

```
ergod.test.lsd( data, vars = dimnames( data )[[ 2 ]],
                start.period = 0, signif = 0.05, digits = 2,
                ad.method = c( "asymptotic", "simulated", "exact" ) )
```

Arguments

data	a three-dimensional array, as the ones produced by read.3d.lsd, organized as (time steps x variables x Monte Carlo instances).
vars	a vector of the variable names (as strings) contained in data for which the tests will be performed. The default is to test all variables.
start.period	integer: the first time step in data to be considered for the tests. The default value is 0 (all time steps considered).
signif	numeric in [0, 1]: statistical significance to evaluate the tests rejection of the null-hypothesis. The default value is 0.05 (5%).
digits	integer: the number of significant digits to show in results. The default is 2.
ad.method	a string in c("asymptotic", "simulated", "exact") defining the methods to be used by ad.test. The default is "asymptotic".

Details

This function is a wrapper to the functions adf.test, kpss.test and bds.test in tseries package, PP.test and ks.test in stats-package and ad.test in kSamples-package.

Value

The function returns a data frame presenting both the average test statistics and the frequency of test null-hypothesis rejections for all the variables selected in vars. Null hypothesis (H0) for ADF and PP tests is non-stationarity of the time series. Null hypothesis (H0) for KPSS test is stationarity of the time series. Null hypothesis (H0) for BDS test the time series is i.i.d.. Null hypothesis (H0) for KS, AD and WW tests is ergodicity of the time series.

Author(s)

Marcelo C. Pereira [aut, cre] (ORCID: <https://orcid.org/0000-0002-8069-2734>)

See Also

symmet.test.lsd(),

list.files.lsd(), read.3d.lsd() in LSDinterface-package,

adf.test(), bds.test(), kpss.test(),

PP.test(), ks.test() in stats-package(),

ad.test() in kSamples-package

Examples

```
# get the list of file names of example LSD results
library( LSDinterface )
files <- list.files.lsd( system.file( "extdata", package = "LSDsensitivity" ),
                         "Sim1.lsd", recursive = TRUE )

# Steps to use this function:
# 1. load data from a LSD simulation saved results using a read.xxx.lsd
```

```
#    function from LSDinterface package (read.3d.lsd, for instance)
# 2. use ergod.test.lsd to apply the tests on the relevant variables,
#    replacing "var2", "var3" etc. with your data

# read data from Monte Carlo runs
dataSet <- read.3d.lsd( files )

tests <- ergod.test.lsd( dataSet,              # the data set to use
                         c( "var2", "var3" ),  # the variables to test
                         signif = 0.01,        # use 1% significance
                         digits = 4 )          # show results using 4 digits

print( tests )
```

`kriging.model.lsd` *Fit a Kriging meta-model to a LSD model sample data*

Description

This function fits a Kriging meta-model (also known as a Gaussian process), using five alternative variance kernels and two trend model options, to the sampled data from a LSD simulation model.

Usage

```
kriging.model.lsd( data, ext.wgth = 0.5, trendModel = 0, covModel = 0,
                   digits = 4 )
```

Arguments

`data`	an object created by a previous call to `read.doe.lsd` which contains all the required experimental data for the analysis.
`ext.wgth`	numeric in [0, 1]: the weight given to the fitting metrics calculated over the out-of-sample (external) validation sample in regard to the in-sample metrics. The default value is 0.5.
`trendModel`	a number corresponding to the trend model: 0 = automatic selection (according to fitting metrics, the default); 1 = constant; 2 = first order polynomial.
`covModel`	a number corresponding to the covariance model (or kernel): 0 = automatic selection (according to fitting metrics, the default); 1 = Matern 5/2; 2 = Matern 3/2; 3 = Gaussian; 4 = exponential; 5 = power exponential.
`digits`	integer: the number of significant digits to show in results. The default is 4.

Details

This function fits a universal Kriging meta-model to the experimental data set previously loaded with `read.doe.lsd` using the Gaussian process method (Rasmussen & Williams, 2006).

This function is a wrapper to the function `km` in `DiceKriging-package`.

Value

The function returns an object/list of class `kriging-model` containing several items:

`selected`	an object containing the selected estimated meta-model (standardized).
`comparison`	a print-ready table with all fitting statistics for all fitted meta-model specifications.
`Q2`	the Q2 in-sample fitting statistic for the selected meta-model.
`rmse`	the RMSE out-of-sample fitting statistic for the selected meta-model.
`mae`	the MAE out-of-sample fitting statistic for the selected meta-model.
`rma`	the RMA out-of-sample fitting statistic for the selected meta-model.
`extN`	number of out-of-sample observations.
`estimation`	a print-ready table with the coefficients (hyper-parameters) of the selected estimated meta-model.
`estimation.std`	a print-ready table with the standardized coefficients (hyper-parameters) of the selected estimated meta-model.
`coefficients`	a vector with the coefficients (hyper-parameters) of the selected estimated meta-model.
`coefficients.std`	a vector with the standardized coefficients (hyper-parameters) of the selected estimated meta-model.
`trend`	number of the selected trend model.
`trendNames`	name of the selected trend model.
`cov`	number of the selected covariance model (kernel).
`covNames`	name of the selected covariance model (kernel).

Note

See the note in LSDsensitivity-package for step-by-step instructions on how to perform the complete sensitivity analysis process using LSD and R.

Author(s)

Marcelo C. Pereira [aut, cre] (ORCID: <https://orcid.org/0000-0002-8069-2734>)

References

Kleijnen JP (2009) *Kriging metamodeling in simulation: a review*. Eur J Oper Res 192(3):707-716

Rasmussen C, Williams C (2006) *Gaussian processes for machine learning*. MIT Press, Cambridge

Roustant O, Ginsbourger D, Deville Y (2012) *Dicekriging, diceoptim: two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization*. J Stat Softw 51(1):1-55

See Also

read.doe.lsd()

km in DiceKriging-package

Examples

```
# get the example directory name
path <- system.file( "extdata/sobol", package = "LSDsensitivity" )

# Steps to use this function:
# 1. define the variables you want to use in the analysis
# 2. load data from a LSD simulation saved results using read.doe.lsd,
#    preferrably using two sets of sampled data (DoEs), one for model
#    estimation and the other for out-of-sample (external) validation
# 3. fit a Kriging (or polynomial) meta-model using kriging.model.lsd

lsdVars <- c( "var1", "var2", "var3" )         # the definition of existing variables

dataSet <- read.doe.lsd( path,                 # data files folder
                         "Sim3",               # data files base name (same as .lsd file)
                   "var3",            # variable name to perform the sensitivity analysis
                   does = 2,          # number of experiments (data + external validation)
                         saveVars = lsdVars )  # LSD variables to keep in dataset

model <- kriging.model.lsd( dataSet )          # estimate best Kriging meta-model

print( model$comparison )                      # model comparison table
print( model$estimation.std )                  # model estimation (standardized) table
```

---

model.limits.lsd *Find maximum and minimum meta-model responses*

Description

This function identifies the maximum and minimum meta-model response values when exploring a subset of three meta-model factors (parameters): one at a time and jointly changing the first and the second factors. All the remaining factors are kept at default/calibration values.

Usage

```
model.limits.lsd( data, model, sa = NULL, factor1 = 1, factor2 = 2,
                  factor3 = 3, pop.size = 1000, max.generations = 30,
                  wait.generations = 10, precision = 1e-05, nnodes = 1 )
```

Arguments

data	an object created by a previous call to read.doe.lsd which contains all the required experimental data for the analysis.

`model` an object created by a previous call to `kriging.model.lsd` or `polynomial.model.lsd` which contains the meta-model estimated hyper-parameters.

`sa` an optional object created by a previous call to `sobol.decomposition.lsd` which contains the meta-model factors importance used to select the top 3 most influential ones for the analysis.

`factor1` integer: the index (according to the Sobol index table) to the first factor to be evaluated individually and jointly with the second factor. The default is the first (index order) factor. Not used if a `sa` object is supplied.

`factor2` integer: the index (according to the Sobol index table) to the second factor to be evaluated individually and jointly with the first factor. The default is the second (index order) factor. Not used if a `sa` object is supplied.

`factor3` integer: the index (according to the Sobol index table) to the third factor to be evaluated only individually. The default is the third (index order) factor. Not used if a `sa` object is supplied.

`pop.size` integer: the number of parallel search paths `genoud` uses to solve the optimization problem. The default is 1000.

`max.generations` integer: the maximum number of generations that `genoud` will run when attempting to optimize a function. The default is 30.

`wait.generations` integer: if there is no improvement in the objective function after this number of generations, `genoud` will accept the optimum. The default is 10.

`precision` numeric: the tolerance level used by `genoud`. Numbers within `precision` are considered to be equal. The default is 1e-5.

`nnodes` integer: the maximum number of parallel computing nodes (parallel threads) in the current computer to be used for reading the files. The default, `nnodes = 1`, means single thread processing (no parallel threads). If equal to zero, creates up to one node per CPU (physical) core. Only `Fork` clusters can be used, because `PSOCK` clusters are not working now, so this option is not available in Windows. Please note that each node requires its own memory space, so memory usage increases linearly with the number of nodes.

Details

This function searches for maximum and minimum response surface values by the application of a genetic algorithm (Sekhon & Walter, 1998).

This function is a wrapper to the function `genoud` in `rgenoud` package.

Value

The function returns a data frame containing the found limit values.

Author(s)

Marcelo C. Pereira [aut, cre] (ORCID: <https://orcid.org/0000-0002-8069-2734>)

References

Sekhon JS, Walter RM (1998). *Genetic optimization using derivatives: theory and application to nonlinear models*. Political Analysis 7:187-210

See Also

read.doe.lsd(), kriging.model.lsd(), polynomial.model.lsd()

genoud

Examples

```
# get the example directory name
path <- system.file( "extdata/sobol", package = "LSDsensitivity" )

# Steps to use this function:
# 1. define the variables you want to use in the analysis
# 2. load data from a LSD simulation saved results using read.doe.lsd
# 3. fit a Kriging (or polynomial) meta-model using kriging.model.lsd
# 4. identify the most influential factors applying sobol.decomposition.lsd
# 5. find the maximum and minimum response values for the 3 top-influential
#    factors/parameters using model.limits.lsd
# 6. plot the response surface indicating the limit points found

lsdVars <- c( "var1", "var2", "var3" )          # the definition of existing variables

dataSet <- read.doe.lsd( path,                  # data files folder
                         "Sim3",               # data files base name (same as .lsd file)
                   "var3",            # variable name to perform the sensitivity analysis
                   does = 2,          # number of experiments (data + external validation)
                         saveVars = lsdVars )   # LSD variables to keep in dataset

model <- kriging.model.lsd( dataSet )           # estimate best Kriging meta-model

SA <- sobol.decomposition.lsd( dataSet, model ) # find Sobol indexes

limits <- model.limits.lsd( dataSet,            # LSD experimental data set
                            model,              # estimated meta-model
                            SA )                # use top factors found before

print( limits )                                 # print a table with the limits

resp <- response.surface.lsd( dataSet, model, SA )# prepare surfaces data

# plot the 3D surface (top 2 factors)
theta3d <- 310                                  # horizontal view angle
phi3d <- 30                                     # vertical view angle
grid3d <- 25

zMat <- matrix( resp$calib[[ 2 ]]$mean, grid3d, grid3d, byrow = TRUE )
zlim <- range( zMat, na.rm = TRUE )
```

```
vt <- persp( resp$grid[[ 1 ]], resp$grid[[ 2 ]], zMat, col = "gray90",
             xlab = colnames( dataSet$doe )[ SA$topEffect[ 1 ] ], zlim = zlim,
           ylab = colnames( dataSet$doe )[ SA$topEffect[ 2 ] ], zlab = dataSet$saVarName,
             theta = theta3d, phi = phi3d, ticktype = "detailed" )

# plot the max, min and default points as colored markers
points( trans3d( as.numeric( dataSet$facDef[ SA$topEffect[ 1 ] ] ),
                 as.numeric( dataSet$facDef[ SA$topEffect[ 2 ] ] ),
                 resp$default$mean, vt ), col = "red", pch = 19, cex = 1.0 )
points( trans3d( limits[ SA$topEffect[ 1 ], 7 ],
                 limits[ SA$topEffect[ 2 ], 7 ],
                 limits[ "response", 7 ], vt ), col = "green", pch = 18, cex = 1.0 )
points( trans3d( limits[ SA$topEffect[ 1 ], 8 ],
                 limits[ SA$topEffect[ 2 ], 8 ],
                 limits[ "response", 8 ], vt ), col = "blue", pch = 18, cex = 1.0 )
```

`model.optim.lsd` *Find optimal meta-model factor settings*

Description

This function finds the optimal factor (parameter) settings using the estimated meta-model.

Usage

```
model.optim.lsd( model, data = NULL, lower.domain = NULL, upper.domain = NULL,
                 starting.values = NULL, minimize = TRUE, pop.size = 1000,
                 max.generations = 30, wait.generations = 10,
                 precision = 1e-05, nnodes = 1 )
```

Arguments

`model`	an object created by a previous call to `kriging.model.lsd` or `polynomial.model.lsd` which contains the meta-model estimated hyper-parameters.
`data`	an optional object created by a previous call to `read.doe.lsd` which sets the default values for `lower.domain`, `upper.domain` and `starting.values`.
`lower.domain`	an optional vector or single-line data frame which contains the minimum values to be considered for all the meta-model factors/variables. If `data` is not provided, the default values are the lower limit ranges from the `.sa` file set in LSD.
`upper.domain`	an optional vector or single-line data frame which contains the maximum values to be considered for all the meta-model factors/variables. If `data` is not provided, the default values are the upper limit ranges from the `.sa` file set in LSD.

`starting.values`
: an optional vector or single-line data frame which contains the starting values to be used by genoud for all the meta-model factors/variables. If `data` is provided, the default values are the calibration settings from the baseline configuration `.lsd` file set in LSD.

`minimize`
: logical: set to FALSE to perform maximization. The default is TRUE (minimization).

`pop.size`
: integer: the number of parallel search paths genoud uses to solve the optimization problem. The default is 1000.

`max.generations`
: integer: the maximum number of generations that genoud will run when attempting to optimize a function. The default is 30.

`wait.generations`
: integer: if there is no improvement in the objective function after this number of generations, genoud will accept the optimum. The default is 10.

`precision`
: numeric: the tolerance level used by genoud. Numbers within `precision` are considered to be equal. The default is 1e-5.

`nnodes`
: integer: the maximum number of parallel computing nodes (parallel threads) in the current computer to be used for reading the files. The default, `nnodes = 1`, means single thread processing (no parallel threads). If equal to zero, creates up to one node per CPU (physical) core. Only `Fork` clusters can be used, because `PSOCK` clusters are not working now, so this option is not available in Windows. Please note that each node requires its own memory space, so memory usage increases linearly with the number of nodes.

Details

This function searches for maximum and minimum response surface values by the application of a genetic algorithm (Sekhon & Walter, 1998).

The function can be used to perform any form of optimization by means the user defines the proper objective function to be maximized (or minimized). Any form of objective function can be easily defined as a new variable to the DoE data set when it is created by `read.doe.lsd`.

This function is a wrapper to the function genoud in `rgenoud` package.

Value

The function returns a single-line data frame which contains values (in the rows) for all the meta-model factors/variables (in the columns) or NULL if optimization fails.

Author(s)

Marcelo C. Pereira [aut, cre] (ORCID: <https://orcid.org/0000-0002-8069-2734>)

References

Sekhon JS, Walter RM (1998). *Genetic optimization using derivatives: theory and application to nonlinear models*. Political Analysis 7:187-210

See Also

read.doe.lsd(), kriging.model.lsd(), polynomial.model.lsd()

genoud

Examples

```
# get the example directory name
path <- system.file( "extdata/sobol", package = "LSDsensitivity" )

# Steps to use this function:
# 1. define the variables you want to use in the analysis
# 2. load data from a LSD simulation saved results using read.doe.lsd
# 3. fit a Kriging (or polynomial) meta-model using kriging.model.lsd
# 4. find the factor configuration that produce the minimum (or maximum)
#    value for the analysis variable defined in step 2

lsdVars <- c( "var1", "var2", "var3" )         # the definition of existing variables

dataSet <- read.doe.lsd( path,                 # data files folder
                         "Sim3",               # data files base name (same as .lsd file)
                   "var3",            # variable name to perform the sensitivity analysis
                  does = 2,          # number of experiments (data + external validation)
                        saveVars = lsdVars )  # LSD variables to keep in dataset

model <- kriging.model.lsd( dataSet )          # estimate best Kriging meta-model

config <- model.optim.lsd( model,          # find meta-model configuration for minimum response
                       dataSet )          # use the full range of factors and starting from
                                          # calibration
print( config )
```

---

model.pred.lsd *Predict meta-model response at given point(s)*

Description

This function predicts the meta-model response at a specific point(s) in the factor (parameter) space and provides a confidence interval for the prediction(s) at 95% confidence.

Usage

```
model.pred.lsd( data.point, model )
```

Arguments

data.point	a single or multi line data frame which contains values (in the rows) for all the meta-model factors/variables (in the columns).
model	an object created by a previous call to kriging.model.lsd or polynomial.model.lsd which contains the meta-model estimated hyper-parameters.

Details

This function simply evaluate the meta-model value at the given point. All factor values must be specified. `data.point` can also be specified as an ordered vector or matrix, following the same order for the factors as defined in the meta-model specification.

This function is a wrapper to the functions `predict.km` in `DiceKriging-package` and `predict.lm` in `stats-package`.

Value

The function returns a list containing the prediction(s) and the confidence bounds. If `data.point` is a data frame or matrix with more than one line, the list elements are vectors. The list element names are:

`mean`	the expected response value.
`lower`	the lower confidence bound.
`upper`	the upper confidence bound.

Author(s)

Marcelo C. Pereira [aut, cre] (ORCID: <https://orcid.org/0000-0002-8069-2734>)

See Also

`kriging.model.lsd()`, `polynomial.model.lsd()`

`predict.km` in `DiceKriging-package`, `predict.lm` in `stats-package`

Examples

```
# get the example directory name
path <- system.file( "extdata/sobol", package = "LSDsensitivity" )

# Steps to use this function:
# 1. define the variables you want to use in the analysis
# 2. load data from a LSD simulation saved results using read.doe.lsd
# 3. fit a Kriging (or polynomial) meta-model using kriging.model.lsd
# 4. estimate the meta-model response at any set of points applying
#    model.pred.lsd

lsdVars <- c( "var1", "var2", "var3" )         # the definition of existing variables

dataSet <- read.doe.lsd( path,                 # data files folder
                         "Sim3",               # data files base name (same as .lsd file)
                   "var3",            # variable name to perform the sensitivity analysis
                   does = 2,          # number of experiments (data + external validation)
                         saveVars = lsdVars )  # LSD variables to keep in dataset

model <- kriging.model.lsd( dataSet )          # estimate best Kriging meta-model

# creates a set of four random points in parameter space
points <- data.frame( par1 = rnorm( 4 ), par2 = rnorm( 4 ), par3 = rnorm( 4 ) )
```

```
response <- model.pred.lsd( points, model )    # predict model response at the 3 points

print( points )
print( response )
```

`polynomial.model.lsd` *Fit a polynomial meta-model to a LSD model sample data*

Description

This function fits a Polynomial meta-model of first or second order, with or without interactions, to the sampled data from a LSD simulation model. Polynomial meta-models are usually inadequate to fit nonlinear simulation models, please use the estimated meta-model carefully.

Usage

```
polynomial.model.lsd( data, ext.wgth = 0.5, ols.sig = 0.2,
                      orderModel = 0, interactModel = 0, digits = 4 )
```

Arguments

`data`	an object created by a previous call to `read.doe.lsd` which contains all the required experimental data for the analysis.
`ext.wgth`	numeric in [0, 1]: the weight given to the fitting metrics calculated over the out-of-sample (external) validation sample in regard to the in-sample metrics. The default value is 0.5.
`ols.sig`	numeric in [0, 1]: the minimum significance considered in the OLS regression.
`orderModel`	a number corresponding to the polynomial model order: 0 = automatic selection (according to fitting metrics, the default); 1 = first order; 2 = second order.
`interactModel`	a number indicating the presence of interaction terms in the model: 0 = automatic selection (according to fitting metrics, the default); 1 = no , 2 = yes.
`digits`	integer: the number of significant digits to show in results. The default is 4.

Details

This function fits a polynomial meta-model to the experimental data set previously loaded with `read.doe.lsd` using the ordinary least-squares (OLS) method.

This function is a wrapper to the function `lm` in `stats-package`.

Value

The function returns an object/list of class `polynomial-model` containing several items:

`selected`	an object containing the selected estimated meta-model.
`comparison`	a print-ready table with all fitting statistics for all fitted meta-model specifications.
`R2`	the adjusted R2 in-sample fitting statistic for the selected meta-model.
`rmse`	the RMSE out-of-sample fitting statistic for the selected meta-model.
`mae`	the MAE out-of-sample fitting statistic for the selected meta-model.
`rma`	the RMA out-of-sample fitting statistic for the selected meta-model.
`extN`	number of out-of-sample observations.
`estimation`	a print-ready table with the coefficients (hyper-parameters) of the selected estimated meta-model.
`estimation.std`	a print-ready table with the standardized coefficients (hyper-parameters) of the selected estimated meta-model.
`coefficients`	a vector with the coefficients (hyper-parameters) of the selected estimated meta-model.
`coefficients.std`	a vector with the standardized coefficients (hyper-parameters) of the selected estimated meta-model.
`order`	order of the selected polynomial model.
`polyNames`	name of the selected polynomial model.
`interact`	number of the selected interaction mode.
`interactNames`	name of the selected interaction mode.

Note

See the note in LSDsensitivity-package for step-by-step instructions on how to perform the complete sensitivity analysis process using LSD and R.

Author(s)

Marcelo C. Pereira [aut, cre] (ORCID: <https://orcid.org/0000-0002-8069-2734>)

See Also

`read.doe.lsd()`

Examples

```
# get the example directory name
path <- system.file( "extdata/sobol", package = "LSDsensitivity" )

# Steps to use this function:
# 1. define the variables you want to use in the analysis
```

```
# 2. load data from a LSD simulation saved results using read.doe.lsd,
#    preferrably using two sets of sampled data (DoEs), one for model
#    estimation and the other for out-of-sample (external) validation
# 3. fit the polynomial meta-model using polynomial.model.lsd

lsdVars <- c( "var1", "var2", "var3" )         # the definition of existing variables

dataSet <- read.doe.lsd( path,                 # data files folder
                         "Sim3",              # data files base name (same as .lsd file)
                    "var3",            # variable name to perform the sensitivity analysis
                    does = 2,          # number of experiments (data + external validation)
                          saveVars = lsdVars )  # LSD variables to keep in dataset

model <- polynomial.model.lsd( dataSet )       # estimate best polynomial meta-model
                                               # using defaults (auto model selection)

print( model$comparison )                      # model comparison table
print( model$estimation.std )                  # model estimation (standardized) table
```

`read.doe.lsd` *Read a set of experimental data from a LSD model*

Description

This function reads the sampling data produced by a LSD model design of experiment (DoE), pre-process it and saves it as a R object that can be used by the other tools provided by the LSDsensitivity package. Optionally, it can be used with a second DoE, on the same simulation model, to allow the out-of-sample (external) validation of the fitted meta-models.

Usage

```
read.doe.lsd( folder, baseName, outVar = "", does = 1, doeFile = NULL,
              respFile = NULL, validFile = NULL, valRespFile = NULL,
              confFile = NULL, limFile = NULL, iniDrop = 0, nKeep = -1,
              saveVars = NULL, addVars = NULL, eval.vars = NULL,
              eval.run = NULL, eval.stat = c( "mean", "median" ),
              pool = TRUE, na.rm = FALSE, rm.temp = TRUE, rm.outl = FALSE,
              lim.outl = 10, nnodes = 1, quietly = TRUE, instance = 1,
              posit = NULL, posit.match = c( "fixed", "glob", "regex" ) )
```

Arguments

`folder`	the *relative* folder path to the LSD DoE data files, using the R working directory as reference (see `getwd()`).
`baseName`	the LSD data files base name, without numbering and extension suffixes (should be the same as the name of the baseline `.lsd` file, without the extension). If `.lsd` extension is included, it is automatically removed.

`outVar` the name of an *existing* variable to be used as the reference to perform the sensitivity analysis. If no name is supplied, the default is to use the first element of `saveVars` or `addVars`.

`does` 1 or 2: number of experiments to be processed, being 2 only when one additional external validation sample (independent from the main sample) is available (see the required files below). The default is 1.

`doeFile` the DoE specification file to be used. For the default (NULL), the `baseName` is used to generate the default LSD generated name.

`respFile` the DoE response file to be used/created. For the default (NULL), the `baseName` is used to generate the name.

`validFile` the external validation DoE specification file to be used. For the default (NULL), the `baseName` is used to generate the default LSD generated name.

`valRespFile` the external validation DoE response file to be used/created. For the default (NULL), the `baseName` is used to generate the name.

`confFile` the LSD baseline `.lsd` configuration file. The default (NULL) is to use `config/baseName.lsd`.

`limFile` the LSD factor limit ranges `.sa` file. The default (NULL) is to use the data contained in the configuration file (`confFile`); if configuration ha no factor limit ranges, tries to load `config/baseName.sa`, if it exists.

`iniDrop` integer: the number of initial time steps to drop from analysis (from `t = 1` till `t = iniDrop`). The default (0) is to remove no time step.

`nKeep` integer: the total number of time steps to keep after `iniDrop`, if simulation length is longer than `nKeep` discard data from the end of the simulation. The default (-1) is to preserve all data.

`saveVars` a vector of existing LSD variable names to be kept in the data set. The default (`NULL`) is to save just `outVar`, if supplied. At least one existing or new variable (defined by `addVars`) must exist. `saveVars` also defines the variables which are available for processing by the function defined by `eval.vars`, the default `saveVars = c()` representing all variables in the results files. For large datasets, this parameter allow reducing the memory required by not loading unnecessary data into memory.

`addVars` a vector of new LSD variable names to be added to the data set. The default (`NULL`) is to add none. At least one existing or new variable must exist.

`eval.vars` a function to recalculate any item of the imported data set, including added variables. The default (NULL) is to have no function (just use selected existing variables as is). If defined, function must take two arguments: the data set for a specific DoE point (time steps in the rows) and the list of variables (columns) in the data set. The function may change any value within the data set but *should not* add or remove rows or columns.

`eval.run` a function to evaluate the DoE response for each experimental sampling point, attributing an optional value to it. The default (NULL) is to have no function. In this case, the function uses the selected variable Monte Carlo mean and standard deviation, or the median and the median absolute deviation if `median = TRUE`, to value the point. If defined, function must take four arguments: (a) the data set (a 3-dimensional array of lists, the numbers of the DoE sampling points in the first dimension, variables in the second, and Monte Carlo runs in the in

the third, being each list element a vector with all time-step values for that sample/variable/MC run), (b) the number of the DoE sampling point to evaluate, (c) the index to the analysis variable in the data set, and (d) the applicable confidence interval. The function must return a R list containing four values for the selected sampling point/variable pair: (a) the average value for the response, (b) the response standard deviation, (c) the the number of observations used, and (d) the number of observations discarded.

`eval.stat` character: define the statistics to be used to evaluate the DoE response when `eval.run = NULL`. Options are `"mean"` to use the mean and the standard deviation (default), or `"median"`, to use the median and the median absolute deviation (MAD). Names can be abbreviated. If an evaluation function is provided in `eval.run`, this option is ignored.

`na.rm` logical: if TRUE NA values are stripped before the computation proceeds.

`rm.temp` logical: if `TRUE` (default), remove response and temporary speed up files. If `FALSE`, do not remove response and temporary speed up files, allowing fast execution of subsequent call to this function. Please note that when set to FALSE, the user *must* temporary files (extension `.Rdata`) manually whenever LSD data files are updated, so they are re-read. This can be done by simply turning `rm.temp` to TRUE once.

`rm.outl` logical: if `TRUE`, remove outliers from data set. Default is `FALSE`, no outliers removal.

`lim.outl` numeric: if `rm.outl = TRUE`, defines the limit for non-outliers deviation in number of standard deviations.

`nnodes` integer: the maximum number of parallel computing nodes (parallel threads) in the current computer to be used for reading the files. The default, `nnodes = 1`, means single thread processing (no parallel threads). If equal to zero, creates up to one node per CPU core. Only `PSOCK` clusters are used, to ensure compatibility with any platform. Please note that each node requires its own memory space, so memory usage increases linearly with the number of nodes.

`quietly` logical: if `TRUE`, no message confirming file reading/information is printed. The default (`FALSE`) is to show details about each results file read.

`pool` logical: if `TRUE` (default), create a simpler data array, pooling together only a single instance for each variable. The instance to be considered is provided by `instance`. If `FALSE`, use all variable instances, which are treated as a single one (indeed, pooling all values...). More control over which instances are used can be obtained by simultaneously using `posit`.

`instance` integer: the instance of the variable to be read, for variables that exist in more than one object. This number is based on the relative position (column) of the variable in the results file. The default (1) is to read the first instance. Only a single existing instance at a time can be read for analysis.

`posit` a string, a vector of strings or an integer vector describing the LSD object position of the variable(s) to select. If an integer vector, it should define the position of a SINGLE LSD object. If a string or vector of strings, each element should define one or more different LSD objects, so the returning matrix may contain variables from more than one object. By setting `posit.match`, globbing (wildcard), and regular expressions can be used to select multiple objects at once;

in this case, all matching objects are selected, but only the instance defined by `instance` is used.

`posit.match` a string defining how the `posit` argument, if provided, should be matched against the LSD object positions. If equal to `"fixed"`, the default, only exact matching is done. `"glob"` allows using simple wildcard characters ('*' and '?') in `posit` for matching. If `posit.match="regex"` interpret `posit` as POSIX 1003.2 extended regular expression(s). See `regular expressions` for details of the different types of regular expressions. Options can be abbreviated.

Details

The function reuses any existing response file(s) (for the main and the optional external validation DoEs) or try to create it (them) if not existing. The response files can be created in relation to any existing, modified or new variable from any simulated time step, including complex combinations of those. New and modified variables (w.r.t. the ones available from LSD) can be easily created by the definition of a `eval.vars(data, varList)` function, as shown in the example below. The response values for each sampling point in the DoE(s) can be evaluated using any math/statistical technique over the entire data for each sampled point in every Monte Carlo run by the definition of a `eval.run(data, mc.run, var.idx, ci)`, as in the example below.

Each call to the function can process a single variable. If sensitivity analysis is being performed on multiple variables, the function must be called several times. However, if `rm.tmp = FALSE` the processing time from the second variable is significantly shortened.

This function requires that the complete set of LSD DoE data files be stored in a single folder/directory. The list of required files is the following (`XX`, `YY` and `ZZ` are sequential control numbers produced by LSD, `i = 0, 1, ...`):

```
folder/baseName.lsd              : LSD baseline configuration (1 file)
folder/baseName.sa               : factor ranges (1 file)
folder/baseName_XX_YY.csv        : DoE specification (1 file)
folder/baseName_XX+i.res[.gz]]   : DoE data (YY-XX+1 files)
folder/baseName_YY+1_ZZ.csv      : validation specification (optional - 1 file)
folder/baseName_YY+1+i.res[.gz]  : validation data (optional - ZZ-YY+1 files)
```

The function generates the required response files for the selected variable of analysis and produces the following files in the same folder/directory (`WWW` is the name of the selected analysis variable):

```
folder/baseName_XX_YY_WWW.csv    : DoE response for the selected variable (1 file)
folder/baseName_YY+1_ZZ_WWW.csv  : validation response for variable (optional - 1 file)
```

When `posit` is supplied together with `col.names` or `instance`, the variable selection process is done in two steps. Firstly, the column names set by `saveVars` and `instance` are selected. Secondly, the instances defined by `posit` are selected from the first selection set. See `select.colnames.lsd` and `select.colattrs.lsd` for examples on how to apply advanced selection options.

Value

The function returns an object/list of class `lsd-doe` containing all the experimental data and the corresponding results regarding the selected reference variable `outVar`, including the data for the out-of-sample (external) validation of the produced meta-models, if available, as well the DoE(s) details

required by the package meta-modelling tools (elementary.effects.lsd, kriging.model.lsd, and polynomial.model.lsd).

List components:

doe	the DoE data. Can be a tabular data frame if pool = TRUE or a four-dimensional array otherwise.
resp	the DoE response data table.
valid	the external validation DoE data. Can be a tabular data frame if pool = TRUE or a four-dimensional array otherwise.
valResp	the external validation DoE response data table.
facLim	the factors limit ranges table.
facLimLo	the factors minimum values.
facLimUp	the factors maximum values.
facDef	the factors default/calibration values.
saVarName	the sensitivity analysis reference variable name, as defined by outVar.

Note

See the note in LSDsensitivity-package for step-by-step instructions on how to perform the complete sensitivity analysis process using LSD and R.

Please refer to LSD documentation about the details on the files produced by its sensitivity analysis tools, in particular when using NOLH, Elementary Effects and MC Range Sensitivity Analysis sampling:

LSD documentation is available at https://www.labsimdev.org/ and the latest binaries and source code can be downloaded at https://github.com/marcov64/Lsd/.

Author(s)

Marcelo C. Pereira [aut, cre] (ORCID: <https://orcid.org/0000-0002-8069-2734>)

See Also

elementary.effects.lsd(), kriging.model.lsd(), polynomial.model.lsd() list.files.lsd()

Examples

```
# get the example directory name
path <- system.file( "extdata/sobol", package = "LSDsensitivity" )

# Steps to use this function:
# 1. define the variables you want to use in the analysis
# 2. optionally, define special handling functions (see examples below)
# 3. load data from a LSD simulation saved results using read.doe.lsd
# 4. perform the elementary effects analysis applying elementary.effects.lsd

# the definition of existing, to take log and to be added variables
lsdVars <- c( "var1", "var2", "var3" )
```

```
logVars <- c( "var1", "var3" )
newVars <- c( "var4" )

# load data from a LSD simulation baseline configuration named "Sim1.lsd" to
# perform sensitivity analysis on the variable named "var1"
# there are two groups of sampled data (DoEs) created by LSD being read
# just use no handling functions for now, see possible examples below
dataSet <- read.doe.lsd( path,                 # data files folder
                         "Sim3",              # data files base name (same as .lsd file)
                        "var3",             # variable name to perform the sens. analysis
                       does = 2,            # # of experiments (data + external validation)
                          iniDrop = 0,          # initial time steps to drop (0=none)
                          nKeep = -1,           # number of time steps to keep (-1=all)
                          saveVars = lsdVars,   # LSD variables to keep in dataset
                          addVars = newVars,    # new variables to add to the LSD dataset
                          eval.stat = "median", # use median to evaluate runs
                          rm.temp = FALSE,      # reuse temporary speedup files
                          rm.outl = FALSE,      # remove outliers from dataset
                      lim.outl = 10,       # limit non-outliers deviation (# of std. devs.)
                          quietly = FALSE )     # show information during processing
print( dataSet$doe )             # the design of the experiment sample done in LSD
print( dataSet$valid )           # the extenal validation sample
print( dataSet$saVarName )       # the variable for which the response was analyzed
print( dataSet$resp )            # analysis of the response of the selected variable

#### OPTIONAL HANDLING FUNCTION EXAMPLES ####

# eval.vars( ) EXAMPLE 1
# the definition of a function to take the log of the required variables () and
# compute the new ones (for use on pool = TRUE databases)

eval.vars <- function( dataSet, allVars ) {
  tsteps <- nrow( dataSet )        # number of time steps in simulated data set
  nvars <- ncol( dataSet )         # number of variables in data set (including new ones)

  # ---- Recompute values for existing variables ----
  for( var in allVars ) {
    if( var %in% logVars ) {     # take the log values of selected variables
      try( dataSet[ , var ] <- log( dataSet[ , var ] ), silent = TRUE )  # <= 0 as NaN
    }
  }

  # ---- Calculate values of new variables (added to LSD data set) ----
  dataSet[ , "var4" ] <- dataSet[ , "var1" ] + dataSet[ , "var2" ]   # example of new var

  return( dataSet )
}

# load data again, now using new variable v4 for analysis
dataSet <- read.doe.lsd( path,                 # data files folder
                         "Sim3",              # data files base name (same as .lsd file)
                        "var4",             # variable name to perform the sens. analysis
                       does = 2,            # # of experiments (data + external validation)
```

```
                          iniDrop = 0,          # initial time steps to drop (0=none)
                          nKeep = -1,           # number of time steps to keep (-1=all)
                          saveVars = lsdVars,   # LSD variables to keep in dataset
                          addVars = newVars,    # new variables to add to the LSD dataset
                    eval.vars = eval.vars,# function to evaluate/adjust/expand the dataset
                          rm.temp = TRUE,       # remove temporary speedup files
                          rm.outl = FALSE,      # remove outliers from dataset
                     lim.outl = 10 )      # limit non-outliers deviation (# of std. devs.)
print( dataSet$doe )            # the design of the experiment sample done in LSD
print( dataSet$valid )          # the external validation sample
print( dataSet$saVarName )      # the variable for which the response was analyzed
print( dataSet$resp )           # analysis of the response of the selected variable

# eval.vars( ) EXAMPLE 2
# the definition of a function to compute the new variables
# (for use on pool = FALSE databases)

# ---- 4D data frame version (when pool = FALSE) ----

eval.vars <- function( data, vars ) {
  tsteps <- length( data [ , 1, 1, 1 ] )
  nvars <- length( data [ 1, , 1, 1 ] )
  insts <- length( data [ 1, 1, , 1 ] )
  samples <- length( data [ 1, 1, 1, ] )

  # ---- Compute values for new variables, preventing infinite values ----
  for( m in 1 : samples )               # for all MC samples (files)
    for( j in 1 : insts )               # all instances
      for( i in 1 : tsteps )            # all time steps
        for( var in vars ) {            # and all variables

          if( var == "var4" ) {
            # Normalization of key variables using the period average size
            mean <- mean( data[ i, "var2", , m ], na.rm = TRUE )
            if( is.finite ( mean ) && mean != 0 )
              data[ i, var, j, m ] <- data[ i,"var2", j, m ] / mean
            else
              data[ i, var, j, m ] <- NA
          }
        }
  return( data )
}

# load data again, now using new variable var2 for analysis
dataSet <- read.doe.lsd( path,                 # data files folder
                         "Sim3",              # data files base name (same as .lsd file)
                       "var2",             # variable name to perform the sens. analysis
                       does = 2,           # # of experiments (data + external validation)
                          iniDrop = 0,          # initial time steps to drop (0=none)
                          nKeep = -1,           # number of time steps to keep (-1=all)
                          pool = FALSE,         # don't pool MC runs
                          saveVars = lsdVars,   # LSD variables to keep in dataset
                          addVars = newVars,    # new variables to add to the LSD dataset
```

```
                    eval.vars = eval.vars,# function to evaluate/adjust/expand the dataset
                         rm.temp = TRUE,       # remove temporary speedup files
                         rm.outl = FALSE,      # remove outliers from dataset
                     lim.outl = 10 )      # limit non-outliers deviation (# of std. devs.)
print( dataSet$doe )            # the design of the experiment sample done in LSD
print( dataSet$valid )          # the external validation sample
print( dataSet$saVarName )      # the variable for which the response was analyzed
print( dataSet$resp )           # analysis of the response of the selected variable

# eval.run( ) EXAMPLE
# the definition of a function to evaluate a point in the DoE, associating a result
# with it (in terms of average result and dispersion/S.D.)
# the example evaluates the fat-taildness of the distribution of the selected
# variable, using the Subbotin distribution b parameter as metric (response)

library( normalp )

eval.run <- function( data, run, varIdx, conf ) {

  obs <- discards <- 0

  # ------ Compute Subbotin fits for each run ------
  bSubbo <- rep( NA, dim( data )[ 3 ] )
  for( i in 1 : dim( data )[ 3 ] ) {
    x <- data[[ run, varIdx, i ]]
    sf <- paramp( x )
    sf$p <- estimatep( x, mu = sf$mean, p = sf$p, method = "inverse" )
    if( sf$p >= 1 ) {
      bSubbo[ i ] <- sf$p
      obs <- obs + 1
    } else {
      bSubbo[ i ] <- NA
      discards <- discards + 1
    }
  }

  return( list( mean( bSubbo, na.rm = TRUE ),
                var( bSubbo, na.rm = TRUE ), obs, discards ) )
}

# load data again, now using the defined evaluation function
dataSet <- read.doe.lsd( path,                 # data files folder
                         "Sim3",              # data files base name (same as .lsd file)
                       "var2",             # variable name to perform the sens. analysis
                      does = 2,           # # of experiments (data + external validation)
                         iniDrop = 0,          # initial time steps to drop (0=none)
                         nKeep = -1,           # number of time steps to keep (-1=all)
                         saveVars = lsdVars,   # LSD variables to keep in dataset
                         addVars = newVars,    # new variables to add to the LSD dataset
                      eval.run = eval.run,  # function to evaluate the DoE point response
                         rm.temp = TRUE,       # remove temporary speedup files
                         rm.outl = FALSE,      # remove outliers from dataset
                     lim.outl = 10 )      # limit non-outliers deviation (# of std. devs.)
```

```
print( dataSet$doe )            # the design of the experiment sample done in LSD
print( dataSet$valid )          # the external validation sample
print( dataSet$saVarName )      # the variable for which the response was analyzed
print( dataSet$resp )           # analysis of the response of the selected variable
```

`response.surface.lsd` *Generate the meta-model 3D response surface data*

Description

This function produces a data object for the three-dimensional graphical representations of the meta-model response surfaces for a set of factors (parameters), including the confidence interval for the surfaces.

Usage

```
response.surface.lsd( data, model, sa, gridSz = 25, defPos = 2,
                      factor1 = 0, factor2 = 0, factor3 = 0 )
```

Arguments

`data`	an object created by a previous call to `read.doe.lsd` which contains all the required experimental data for the analysis.
`model`	an object created by a previous call to `kriging.model.lsd` or `polynomial.model.lsd` which contains the meta-model estimated hyper-parameters.
`sa`	an object created by a previous call to `sobol.decomposition.lsd` which contains the estimated total and conditional variances for all the meta-model factors.
`gridSz`	integer: the number of divisions in the 3D wire frame grid. The default is 25.
`defPos`	1, 2, 3: the position of the default/calibration configuration on the 3 plot sequence. The default is 2 (center position).
`factor1`	integer: the index of the first most-important factor: 0 = automatic selection (according to the Sobol index, the default); any other number = the selected factor index, according to DoE factor order.
`factor2`	integer: the index of the second most-important factor: 0 = automatic selection (according to the Sobol index, the default); any other number = the selected factor index, according to DoE factor order.
`factor3`	integer: the index of the third most-important factor: 0 = automatic selection (according to the Sobol index, the default); any other number = the selected factor index, according to DoE factor order.

Details

This function produces data for three different wire frame 3D plots. In the 3 plots, the x-y plan is defined by the 2 most-important factors (calculated or set by the user in `sobol.decomposition.lsd`) and the z axis represents the response variable chosen. The three different plots shows the response surface for three values of the third most-important factor: the minimum, the default/calibration and the maximum. The order the three response surfaces are shown is defined by `defPos`.

The automatically set most-important factors can be overridden by any factors chosen by the user by the usage of the arguments `factor1`, `factor2` and `factor3`. This way, the response surfaces can be represented for a combination of any 3 factors (parameters) in the model.

Value

The function returns an object/list of class `response` containing three similar objects, one for each 3D plot, each of them comprised of:

`calib`	the predicted meta-model response values on each point of the 3D grid.
`factor`	the predicted values for each individual factor.
`default`	the predicted values for the default/calibration configuration.

Note

See the note in LSDsensitivity-package for step-by-step instructions on how to perform the complete sensitivity analysis process using LSD and R.

Author(s)

Marcelo C. Pereira [aut, cre] (ORCID: <https://orcid.org/0000-0002-8069-2734>)

See Also

`read.doe.lsd()`, `kriging.model.lsd()`, `polynomial.model.lsd()`, `sobol.decomposition.lsd()`

Examples

```
# get the example directory name
path <- system.file( "extdata/sobol", package = "LSDsensitivity" )

# Steps to use this function:
# 1. define the variables you want to use in the analysis
# 2. load data from a LSD simulation saved results using read.doe.lsd
# 3. fit a Kriging (or polynomial) meta-model using kriging.model.lsd
# 4. identify the most influential factors applying sobol.decomposition.lsd
# 5. calculate the response surface for the selected factors using model.limits.lsd
# 6. plot the response surface

lsdVars <- c( "var1", "var2", "var3" )          # the definition of existing variables

dataSet <- read.doe.lsd( path,                  # data files folder
                         "Sim3",               # data files base name (same as .lsd file)
                    "var3",            # variable name to perform the sensitivity analysis
```

```
                    does = 2,           # number of experiments (data + external validation)
                          saveVars = lsdVars )   # LSD variables to keep in dataset

model <- kriging.model.lsd( dataSet )           # estimate best Kriging meta-model

SA <- sobol.decomposition.lsd( dataSet, model ) # find Sobol indexes

resp <- response.surface.lsd( dataSet,          # LSD experimental data set
                              model,            # estimated meta-model
                              SA )              # Sobol sensitivity analysis results

theta3d <- 310                                  # horizontal view angle
phi3d <- 30                                     # vertical view angle
grid3d <- 25

for( i in 1 : 3 ) {                             # do for each top factor
                                                # plot 3D grid charts
  zMat <- matrix( resp$calib[[ i ]]$mean, grid3d, grid3d, byrow = TRUE )
  zlim <- range( zMat, na.rm = TRUE )

  vt <- persp( resp$grid[[ 1 ]], resp$grid[[ 2 ]], zMat, col = "gray90",
              xlab = colnames( dataSet$doe )[ SA$topEffect[ 1 ] ], zlim = zlim,
            ylab = colnames( dataSet$doe )[ SA$topEffect[ 2 ] ], zlab = dataSet$saVarName,
              theta = theta3d, phi = phi3d, ticktype = "detailed" )
}
```

`sobol.decomposition.lsd`

Sobol variance decomposition sensitivity analysis

Description

This function performs the global sensitivity analysis of a previously fitted meta-model using the Sobol variance decomposition method (Saltelli et al., 2008). If no model is supplied, uses a B-spline smoothing interpolation model.

Usage

```
sobol.decomposition.lsd( data, model = NULL, krig.sa = FALSE, sa.samp = 1000 )
```

Arguments

`data`	an object created by a previous call to `read.doe.lsd` which contains all the required experimental data for the analysis.
`model`	an object created by a previous call to `kriging.model.lsd` or `polynomial.model.lsd` which contains the meta-model estimated hyper-parameters. If no model is supplied (the default), performs the decomposition directly over the experimental data assuming a B-spline smoothing interpolation model.

krig.sa	logical: use alternative Kriging-specific algorithm if TRUE (see sobolGP). Default is FALSE. Applicable only to Kriging meta-models.
sa.samp	integer: number of samples to use in sensitivity analysis. The default is 1000.

Details

This function performs the global sensitivity analysis on a meta-model, previously estimated with kriging.model.lsd or polynomial.model.lsd, using the Sobol variance decomposition method (Saltelli et al., 2008).

This function is a wrapper to the functions fast99 and sobolGP in sensitivity-package.

Value

The function returns an object/list of class kriging-sa or polynomial-sa, according to the input meta-model, containing several items:

metamodel	an object/list of class fast99 containing the estimated total and conditional variances for all the meta-model factors.
sa	a print-ready data frame with the Sobol indexes for each factor.
topEffect	a vector containing the indexes to the three most influential factors, automatically calculated (if factorX = 0) or according to the order pre-selected by the user.

If no model is supplied and a B-spline smoothing interpolation model cannot be fitted, returns NULL.

Note

See the note in LSDsensitivity-package for step-by-step instructions on how to perform the complete sensitivity analysis process using LSD and R.

Author(s)

Marcelo C. Pereira [aut, cre] (ORCID: <https://orcid.org/0000-0002-8069-2734>)

References

Saltelli A, Ratto M, Andres T, Campolongo F, Cariboni J, Gatelli D, Saisana M, Tarantola S (2008) *Global sensitivity analysis: the primer*. Wiley, New York

See Also

read.doe.lsd(), kriging.model.lsd(), polynomial.model.lsd()

fast99(), sobolGP() in sensitivity-package

Examples

```
# get the example directory name
path <- system.file( "extdata/sobol", package = "LSDsensitivity" )

# Steps to use this function:
# 1. define the variables you want to use in the analysis
# 2. load data from a LSD simulation saved results using read.doe.lsd
# 3. fit a Kriging (or polynomial) meta-model using kriging.model.lsd
# 4. perform the sensitivity analysis applying sobol.decomposition.lsd

lsdVars <- c( "var1", "var2", "var3" )         # the definition of existing variables

dataSet <- read.doe.lsd( path,                 # data files folder
                         "Sim3",               # data files base name (same as .lsd file)
                   "var3",            # variable name to perform the sensitivity analysis
                  does = 2,          # number of experiments (data + external validation)
                        saveVars = lsdVars )  # LSD variables to keep in dataset

model <- kriging.model.lsd( dataSet )          # estimate best Kriging meta-model

SA <- sobol.decomposition.lsd( dataSet,        # LSD experimental data set
                               model )         # estimated meta-model

print( SA$topEffect )                          # indexes to the top 3 factors
print( SA$sa )                                 # Sobol indexes table
barplot( t( SA$sa ) )                          # plot Sobol indexes chart
```

`symmet.test.lsd` *Unimodality and symmetry tests*

Description

Perform a set of unimodality and symmetry tests useful for simulation model data from a Monte Carlo experiment distributions. The included tests are: Hartigans dip test for unimodality (Hdip), and the Cabilio and Masaro (CM), the Mira (M), and the Miao, Gel and Gastwirth tests for symmetry.

Usage

```
symmet.test.lsd( data, vars = dimnames( data )[[ 2 ]], start.period = 0,
                 signif = 0.05, digits = 2, sym.boot = FALSE )
```

Arguments

`data`	a three-dimensional array, as the ones produced by `read.3d.lsd`, organized as (time steps x variables x Monte Carlo instances).
`vars`	a vector of the variable names (as strings) contained in `data` for which the tests will be performed. The default is to test all variables.

start.period	integer: the first time step in data to be considered for the tests. The default value is 0 (all time steps considered).
signif	numeric in [0, 1]: statistical significance to evaluate the tests rejection of the null-hypothesis. The default value is 0.05 (5%).
digits	integer: the number of significant digits to show in results. The default is 2.
sym.boot	logical: set to TRUE to use bootstrap to obtain critical values. The default (FALSE) is to use asymptotic distribution of the statistics.

Details

This function is a wrapper to the functions dip.test in diptest package, and symmetry.test in lawstat package.

Value

The function returns a data frame presenting both the average test statistics and the frequency of test null-hypothesis rejections for all the variables selected in vars. Null hypothesis (H0) for Hdip test is an unimodal distribution for the Monte Carlo distribution. Null hypothesis (H0) for CM, M and MGG tests is a symmetric distribution for the Monte Carlo distribution.

Author(s)

Marcelo C. Pereira [aut, cre] (ORCID: <https://orcid.org/0000-0002-8069-2734>)

See Also

ergod.test.lsd(),

list.files.lsd(), read.3d.lsd() in LSDinterface-package,

dip.test(), symmetry.test()

Examples

```
# get the list of file names of example LSD results
library( LSDinterface )
files <- list.files.lsd( system.file( "extdata", package = "LSDsensitivity" ),
                         "Sim1.lsd", recursive = TRUE )

# Steps to use this function:
# 1. load data from a LSD simulation saved results using a read.xxx.lsd
#    function from LSDinterface package (read.3d.lsd, for instance)
# 2. use symmet.test.lsd to apply the tests on the relevant variables,
#    replacing "var1", "var2" etc. with your data

# read data from Monte Carlo runs
dataSet <- read.3d.lsd( files )

# apply tests
tests <- symmet.test.lsd( dataSet,              # the data set to use
                          c( "var2", "var3" ),  # the variables to test
                          signif = 0.01,        # use 1% significance
```

```
                          digits = 4,           # show results using 4 digits
                          sym.boot = FALSE )    # use bootstrap for precision
print( tests )
```

Index