Package 'mistral'

February 20, 2015

Type Package

Title Methods in Structural Reliability

Version 1.1-1

Date 2014-11-24

Author Nicolas BOUSQUET, Gilles DEFAUX, Bertrand IOOSS, Vincent MOUTOUSSAMY, Clement WALTER

Maintainer Bertrand Iooss <biooss@yahoo.fr>

Depends R (>= 3.0.0)

Imports DiceKriging, e1071, kernlab, Matrix, mvtnorm, rgenoud

Description Various reliability analysis methods for: 1) computing failure probability (probability that the output of a numerical model exceeds a threshold); 2) computing the size of a sample to estimate a quantile with a confidence level.

License CeCILL

NeedsCompilation no

Repository CRAN

Date/Publication 2014-11-25 15:36:31

R topics documented:

`mistral-package` *Methods in structural reliability*

Description

Provide tools for structural reliability analysis (failure probability, quantile).

Details

Package:	mistral	
Type:	Package	
Version:	1.1-0 Date:	2013-12-17
License:	CeCILL	

This package provides tools for structural reliability analysis:

- Calculate failure probability with FORM method and importance sampling, compute the design point, give the importance factors.
- Calculate failure probability with crude Monte Carlo method
- Calculate failure probability with Subset Simulation algorithm
- Calculate failure probability with Monotonic Reliability Methods (MRM)
- Calculate failure probability with metamodel based algorithms : AKMCS, SMART and MetaIS
- Calculate failure probability with a metamodel based Subset Simulation : S2MART
- Compute Wilks formula (minimal size of a sample to estimate quantile with a confidence level).

Author(s)

Gilles Defaux, Bertrand Iooss, Vincent Moutoussamy, Clement Walter with contributions from Nicolas Bousquet and Paul Lemaitre (maintainer: Bertrand Iooss <biooss@yahoo.fr>)

References

S.-K. Au, J. L. Beck. Estimation of small failure probabilities in high dimensions by Subset Simulation. Probabilistic Engineering Mechanics, 2001.

J.-M. Bourinet, F. Deheeger, M. Lemaire. Assessing small failure probabilities by combined Subset Simulation and Support Vector Machines. Structural Safety, 2011.

N. Bousquet. Accelerated monte carlo estimation of exceedance probabilities under monotonicity constraints. Annales de la Faculte des Sciences de Toulouse. XXI(3), 557-592, 2012

H.A. David and H.N. Nagaraja. Order statistics, Wiley, 2003.

F. Deheeger. Couplage mecano-fiabiliste : 2SMART - methodologie d'apprentissage stochastique en fiabilite. PhD. Thesis, Universite Blaise Pascal - Clermont II, 2008

A. Der Kiureghian, T. Dakessian. Multiple design points in first and second-order reliability. Structural Safety, vol.20, 1998.

O. Ditlevsen and H.O. Madsen. Structural reliability methods, Wiley, 1996.

V. Dubourg. Meta-modeles adaptatifs pour l'analyse de fiabilite et l'optimisation sous containte fiabiliste. PhD. Thesis, Universite Blaise Pascal - Clermont II, 2011.

B. Echard, N. Gayton, M. Lemaire. AK-MCS : an Active learning reliability method combining Kriging and Monte Carlo Simulation

M. Lemaire, A. Chateauneuf and J. Mitteau. Structural reliability, Wiley Online Library, 2009.

W.T. Nutt and G.B. Wallis. Evaluation of nuclear safety from the outputs of computer codes in the presence of uncertainties. Reliability Engineering and System Safety, 83:57-77, 2004.

P.-H. Waarts. Structural reliability using finite element methods: an appraisal of DARS, Directional Adaptive Response Surface Sampling. PhD. Thesis, Technical University of Delft, The Netherlands, 2000.

S.S. Wilks. Determination of Sample Sizes for Setting Tolerance Limits. Annals Mathematical Statistics, 12:91-96, 1941.

Examples

```
########## FORM ############

 distribution = list()
 distribution[[1]] = list("gamma",c(2,1))
 distribution[[2]] = list("gamma",c(3,1))

 func <- function(X){
   X[1]/sum(X) - qbeta((1e-5),2,3)
 }

 res.list <- FORM(func, u.dep = c(0,0.1), choice.law = distribution,
     N.calls = 1000, eps = 1e-7, Method = "HLRF", IS = "TRUE",
     q = 0.1, copula = "unif")

########### Wilks ##########

N <- WilksFormula(0.95,0.95,order=1)
print(N)
```

AKMCS *Active learning reliability method combining Kriging and Monte Carlo Simulation*

Description

Estimate a failure probability with the AKMCS method.

Usage

```
AKMCS(dimension,
      limit_state_function,
      N                 = 500000,
      N1                = 10*dimension,
      Nmax              = 200,
      learn_db          = NULL,
      lsf_value         = NULL,
      failure           = 0.0,
      precision         = 0.05,
      meta_model        = NULL,
      kernel            = "matern5_2",
      learn_each_train  = FALSE,
      crit_min          = 2,
      limit_fun_MH      = NULL,
      sampling_strategy = "MH",
      first_DOE         = "Gaussian",
      seeds             = NULL,
      seeds_eval        = NULL,
      burnin            = 30,
      thinning          = 4,
      plot              = FALSE,
      limited_plot      = FALSE,
      add               = FALSE,
      output_dir        = NULL,
      z_MH              = NULL,
      z_lsf             = NULL,
      verbose        = 0)
```

Arguments

`dimension`	an integer giving the dimension of the input space.
`limit_state_function`	the failure function.
`N`	an integer defining the Monte-Carlo population size for probability estimation.
`N1`	an integer defining the size of the first Design Of Experiment got by clustering of the `N` standard gaussian samples.
`Nmax`	an integer defining the maximum number of calls to the limit state function during refinement steps. This means total number of call will be `N1` + `Nmax`.
`learn_db`	optional. A matrix of already known points, with dim : `dimension` x number_of_vector.
`lsf_value`	values of the `limit_state_function` on the vectors given in `learn_db`.
`failure`	the value defining the failure domain F = { x | `limit_state_function`(x) < `failure` }.
`precision`	the maximum value of the coefficient of variation for `proba`. If the first run with `N` gives a too large `cov`, then approximate necessary `N` is derived from `cov` and Monte-Carlo estimate is run again.

`meta_model` optional. If a kriging based metamodel has already been fitted to the data (from **DiceKriging** package) it can be given as an input to keep the same parameters.

`kernel` a specified kernel to be used for the metamodel. See **DiceKriging** for available options.

`learn_each_train` specify if hyperparameters of the model should be evaluated each time points are added to the learning database ("TRUE") or only the first time ("FALSE").

`crit_min` the minimum value of the criterion to be used for refinement step.

`limit_fun_MH` optional. If the working space is to be reduced to some subset defining by a function, eg. in case of use in a Subset Simulation algorithm. As for the `limit_state_function`, failure domain is defined by points whom values of `limit_fun_MH` are negative.

`sampling_strategy` either "AR" or "MH", to specify which sampling strategy is to be used when generating Monte-Carlo population in a case of subset simulation : "AR" stands for 'accept-reject' while "MH" stands for Metropolis-Hastings.

`first_DOE` Either "Gaussian" or "Uniform", to specify the population on which clustering if done

`seeds` optional. If `sampling_strategy`=="MH", seeds from which MH algorithm starts. This should be a matrix with 'nrow' = `dimension` and 'ncol' = number of vector.

`seeds_eval` optional. The value of the `limit_fun_MH` on the `seeds`.

`burnin` a burnin parameter for Metropolis-Hastings algorithm.

`thinning` a thinning parameter for Metropolis-Hastings algorithm. `thinning` = 0 means no thinning.

`plot` a boolean parameter specifying if function and samples should be plotted. The plot is refreshed at each iteration with the new data. Note that this option is only to be used when working on 'light' limit state functions as it requires the calculus of this function on a grid of size 161x161 (plot are done a -8:8 x -8:8 grid with 161 meshes.

`limited_plot` only a final plot with `limit_state_function`, final DOE and metamodel. Should be used with `plot`==FALSE.

`add` optional. "TRUE" if plots are to be added to the current active device.

`output_dir` optional. If plots are to be saved in .jpeg in a given directory. This variable will be pasted with "_AKMCS.jpeg" to get the full output directory.

`z_MH` optional. For plots, if metamodel has already been evaluated on the grid then `z_MH` (from outer function) can be provided to avoid extra computational time.

`z_lsf` optional. For plots, if LSF has already been evaluated on the grid then `z_lsf` (from outer function) can be provided to avoid extra computational time.

`verbose` Eiher 0 for an almost no output message, or 1 for medium size or 2 for full size

Details

AKMCS strategy is based on a original Monte-Carlo population which is classified with a kriging-based metamodel. This means that no sampling is done during refinements steps. Indeed, it tries to classify this Monte-Carlo population with a confidence greater than a given value, for instance 'distance' to the failure should be greater than `crit_min` standard deviation.

Thus, while this criterion is not verified, the point minimizing it is added to the learning database and then evaluated.

Finally, once all points are classified or when the maximum number of calls has been reached, crude Monte-Carlo is performed. A final test controlling the size of this population regarding the targeted coefficient of variation is done; if it is too small then a new population of sufficient size (considering ordre of magnitude of found probability) is generated, and algorithm run again.

Value

An object of class `list` containing the failure probability and some more outputs as described below:

`proba`	The estimated failure probability.
`cov`	The coefficient of variation of the Monte-Carlo probability estimate.
`Ncall`	The total number of calls to the `limit_state_function`.
`learn_db`	The final learning database, ie. all points where `limit_state_function` has been calculated.
`lsf_value`	The value of the `limit_state_function` on the learning database.
`meta_fun`	The metamodel approximation of the `limit_state_function`. A call output is a list containing the value and the standard deviation.
`meta_model`	The final metamodel. An S4 object from **DiceKriging**.
`points`	Points in the failure domain according to the metamodel.
`meta_eval`	Evaluation of the metamodel on these points.
`z_meta`	If `plot`==TRUE, the evaluation of the metamodel on the plot grid.

Note

Problem is supposed to be defined in the standard space. If not, use `UtoX` to do so. Furthermore, each time a set of vector is defined as a matrix, 'nrow' = `dimension` and 'ncol' = number of vector.

Author(s)

Clement Walter
<clement.walter@cea.fr>

References

- B. Echard, N. Gayton, M. Lemaire:
AK-MCS : an Active learning reliability method combining Kriging and Monte Carlo Simulation
Structural Safety, Elsevier, 2011.

- B. Echard, N. Gayton, M. Lemaire and N. Relun:
A combined Importance Sampling and Kriging reliability method for small failure probabilities with time-demanding numerical models
Reliability Engineering \& System Safety,2012

- B. Echard, N. Gayton and A. Bignonnet:
A reliability analysis method for fatigue design
International Journal of Fatigue, 2014

See Also

SubsetSimulation MonteCarlo km (in package **DiceKriging**)

Examples

```
#Limit state function defined by Kiureghian & Dakessian :
kiureghian = function(x, b=5, kappa=0.5, e=0.1) {
x = as.matrix(x)
b - x[2,] - kappa*(x[1,]-e)^2
}

## Not run: res = AKMCS(dimension=2,limit_state_function=kiureghian,plot=TRUE)

#Compare with crude Monte-Carlo reference value
N = 500000
U = matrix(rnorm(dimension*N),dimension,N)
G = apply(U,2,kiureghian)
P = mean(G<0)
cov = sqrt((1-P)/(N*P))

## End(Not run)

#See impact of kernel choice with Waarts function :
waarts = function(x) {
x = as.matrix(x)
apply(x, 2, function(u) {min(
(3+(u[1]-u[2])^2/10 - (u[1]+u[2])/sqrt(2)),
(3+(u[1]-u[2])^2/10 + (u[1]+u[2])/sqrt(2)),
u[1]-u[2]+7/sqrt(2),
u[2]-u[1]+7/sqrt(2))})
}

## Not run:
res = list()
res$matern5_2 = AKMCS(2, waarts, plot=TRUE)
res$matern3_2 = AKMCS(2, waarts, kernel="matern3_2", plot=TRUE)
res$gaussian  = AKMCS(2, waarts, kernel="gauss", plot=TRUE)
res$exp       = AKMCS(2, waarts, kernel="exp", plot=TRUE)

#Compare with crude Monte-Carlo reference value
N = 500000
```

```
U = matrix(rnorm(dimension*N),dimension,N)
G = apply(U,2,waarts)
P = mean(G<0)
cov = sqrt((1-P)/(N*P))

## End(Not run)
```

ComputeDistributionParameter *Compute internal parameters and moments for univariate distribution functions*

Description

Compute the internal parameters needed in the definition of several distribution functions when unknown

Usage

```
ComputeDistributionParameter(margin)
```

Arguments

margin	A list containing the definition of the marginal distribution function

Value

margin	The updated list

Author(s)

gilles DEFAUX, <gilles.defaux@cea.fr>

Examples

```
distX1 <- list(type='Lnorm', MEAN=120.0, STD=12.0, P1=NULL, P2=NULL, NAME='X1')
distX1 <- ComputeDistributionParameter(distX1)
print(distX1)
```

FORM *FORM method*

Description

Calculate failure probability by FORM method and important sampling.

Usage

```
FORM(f, u.dep, choice.law, N.calls = 1e4, eps = 1e-7,
     Method = "HLRF", IS = FALSE, q = 0.5, copula = "unif")
```

Arguments

f	the failure fonction
u.dep	a vector, starting point to the research of the design point
choice.law	a list which contains the name of the densityof each input and their parameters
N.calls	Number of calls to f allowed
eps	stop criterion : distance of two points between two iterations
Method	Choice of the method to the research of the design point: "AR" for Abdo-Rackwitz and "HLRF" for Hasofer-Lindt-Rackwitz-Fiessler
IS	"TRUE" for using importance Sampling method (applied after FORM which gives the importance density). Default ="FALSE".
q	ratio of N.calls for the reserach of the design point by FORM. Default = 0.5. 1-q = the remaining ratio to use importance sampling.
copula	choice of the copula. Default = "unif" (uniform copula)

Value

proba.def	Failure probability
indice.reliab	reliability index
compt.f	Number of calls of f
Design.Point	Coordinates of the design point
fact.imp	Importance factors
variance	Standard error of the probability estimator (if IS = TRUE)
Interval.conf	Confidence interval of the estimator at 0.95 (if IS = TRUE)
DOE	List which contains the design of experiments

Author(s)

Vincent Moutoussamy

References

O. Ditlevsen and H.O. Madsen. Structural reliability methods, Wiley, 1996

M. Lemaire, A. Chateauneuf and J. Mitteau. Structural reliability, Wiley Online Library, 2009.

Examples

```
distribution = list()
distribution[[1]] = list("gamma",c(2,1))
distribution[[2]] = list("gamma",c(3,1))

func <- function(X){
   X[1]/sum(X) - qbeta((1e-5),2,3)
}

res.list <- FORM(func,
                 u.dep = c(0,0.1),
                 choice.law = distribution,
                 N.calls = 1000,
                 eps = 1e-7,
                 Method = "HLRF",
                 IS = "TRUE",
                 q = 0.1,
                 copula = "unif")
```

`MetaIS` *Metamodel based Impotance Sampling*

Description

Estimate failure probability by MetaIS method.

Usage

```
MetaIS(dimension,
      limit_state_function,
      N                 = 500000,
      N_alpha           = 100,
      N_DOE             = 2*dimension,
      N1                = N_DOE*30,
      Ru                = 8,
      Nmin              = 30,
      Nmax              = 200,
      Ncall_max         = 1000,
      precision         = 0.05,
      N_seeds           = 1,
      Niter_seed        = 10000,
      N_alphaLOO        = 5000,
```

```
K_alphaLOO        = 2*dimension,
alpha_int         = c(0.1,10),
k_margin          = 1.96,
learn_db          = NULL,
lsf_value         = NULL,
failure           = 0,
meta_model        = NULL,
kernel            = "matern5_2",
learn_each_train  = FALSE,
limit_fun_MH      = NULL,
sampling_strategy = "MH",
seeds             = NULL,
seeds_eval        = NULL,
burnin            = 20,
thinning          = 4,
plot              = FALSE,
limited_plot      = FALSE,
add               = FALSE,
output_dir        = NULL,
z_MH              = NULL,
z_lsf             = NULL,
verbose         = 0)
```

Arguments

`dimension`	an integer giving the dimension of the input space.
`limit_state_function`	the failure fonction.
`N`	an integer defining the size of Monte-Carlo population for augmented failure probability estimation.
`N_alpha`	an integer defining the initial size of Monte-Carlo population for alpha estimate.
`N_DOE`	an integer defining the size of initial DOE got by clustering of the `N1` samples.
`N1`	an integer defining the size of the initial uniform population sampled in a hypersphere of radius Ru.
`Ru`	a real defining the radius of the hypersphere for the initial uniform sampling.
`Nmin`	an integer defining the minimum number of calls to the `limit_state_function` for the construction step.
`Nmax`	an integer defining the maximum number of calls to the `limit_state_function` for the construction step.
`Ncall_max`	an integer defining the maximum number of calls to the `limit_state_function` for the whole algorithm.
`precision`	a real defining the targeted maximal value of the coefficient of variation.
`N_seeds`	an integer defining the number of seeds for Metropolis-Hastings algorithm while generating into the margin (according to MarginProbability*gauss density).
`Niter_seed`	maximum number of iterations for the research of a seed for alphaLOO refinement sampling.

`N_alphaLOO` an integer defining the number of points to sample at each refinement step.

`K_alphaLOO` an integer defining the number of clusters at each refinement step.

`alpha_int` a 2-d real vector defining the range for alphaLOO criterion : `alpha_int`[1] < alphaLOO < `alpha_int`[2].

`k_margin` a real defining the margin width according to standard gaussian law; default is 1.96 which means points outside of the margin are classified with more than 97,5%.

`learn_db` optional. A matrix of already known points, with dim : `dimension` x number_of_vector.

`lsf_value` values of the limit state function on the vectors given in `learn_db`.

`failure` the value defining the failure domain F = { x | `limit_state_function`(x) < `failure` }.

`meta_model` optional. If a kriging based metamodel has already been fitted to the data (from **DiceKriging** package) it can be given as an input to keep the same parameters.

`kernel` a specified kernel to be used for the metamodel. See **DiceKriging** for available options.

`learn_each_train`
specify if hyperparameters of the model should be evaluated each time points are added to the learning database ("TRUE") or only the first time ("FALSE").

`limit_fun_MH` optional. If the working space is to be reduced to some subset defining by a function, eg. in case of use in a Subset Simulation algorithm. As for the `limit_state_function`, failure domain is defined by points whom values of `limit_fun_MH` are negative.

`sampling_strategy`
either "AR" or "MH", to specify which sampling strategy is to be used when generating Monte-Carlo population in a case of subset simulation : "AR" stands for 'accept-reject' while "MH" stands for Metropolis-Hastings.

`seeds` optional. If `sampling_strategy`=="MH", seeds from which MH algorithm starts. This should be a matrix with 'nrow' = `dimension` and 'ncol' = number of vector.

`seeds_eval` optional. The value of the `limit_fun_MH` on the `seeds`.

`burnin` a burnin parameter for Metropolis-Hastings algorithm. To be used only for Monte-Carlo population sampling and set to 0 elsewhere.

`thinning` a thinning parameter for Metropolis-Hastings algorithm. `thinning` = 0 means no thinning. To be used only for Monte-Carlo population sampling and set to 0 elsewhere.

`plot` a boolean parameter specifying if function and samples should be plotted. The plot is refreshed at each iteration with the new data. Note that this option is only to be used when working on 'light' limit state functions as it requires the calculus of this function on a grid of size 161x161 (plot is done a -8:8 x -8:8 grid with 161 meshes.

`limited_plot` only a final plot with `limit_state_function`, final DOE and metamodel. Should be used with `plot`==FALSE.

add	optional. "TRUE" if plots are to be added to the current active device.
output_dir	optional. If plots are to be saved in .jpeg in a given directory. This variable will be pasted with "_MetaIS.jpeg" to get the full output directory.
z_MH	optional. For plots, if metamodel has already been evaluated on the grid then z_MH (from outer function) can be provided to avoid extra computational time.
z_lsf	optional. For plots, if LSF has already been evaluated on the grid then z_lsf (from outer function) can be provided to avoid extra computational time.
verbose	Eiher 0 for an almost no output message, or 1 for medium size or 2 for full size

Details

MetaIS is an Important Sampling based probability estimator. It makes use of a kriging surogate to approximate the optimal density function, replacing the indicatrice by its kriging pendant, the probability of being in the failure domain. In this context, the normallizing constant of this quasi-optimal PDF is called the 'augmented failure probability' and the modified probability 'alpha'.

After a first uniform Design of Experiments, MetaIS uses an alpha-Leave-One-Out criterion combined with a margin sampling strategy to refine a kriging-based metamodel. Samples are generated according to the weighted margin probability with Metropolis-Hastings algorithm and some are selected by clustering; the N_seeds are got from an accept-reject strategy on a standard population.

Once criterion is reached or maximum number of call done, the augmented failure probability is estimated with a crude Monte-Carlo. Then, a new population is generated according to the quasi-optimal instrumenal PDF; burnin and thinning are used here and alpha is evaluated. While the coefficient of variation of alpha estimate is greater than a given threshold and some computation spots still available (defined by Ncall_max) the estimate is refined with extra calculus.

The final probability is the product of p_epsilon and alpha, and final squared coefficient of variation is the sum of p_epsilon and alpha one's.

Value

An object of class list containing the failure probability and some more outputs as described below:

proba	The estimated failure probability.
cov	The coefficient of variation of the Monte-Carlo probability estimate.
Ncall	The total number of calls to the limit_state_function.
learn_db	The final learning database, ie. all points where limit_state_function has been calculated.
lsf_value	The value of the limit_state_function on the learning database.
meta_fun	The metamodel approximation of the limit_state_function. A call output is a list containing the value and the standard deviation.
meta_model	The final metamodel. An S4 object from **DiceKriging**.
points	Points in the failure domain according to the metamodel.
meta_eval	Evaluation of the metamodel on these points.
z_meta	If plot==TRUE, the evaluation of the metamodel on the plot grid.

Note

Problem is supposed to be defined in the standard space. If not, use `UtoX` to do so. Furthermore, each time a set of vector is defined as a matrix, 'nrow' = `dimension` and 'ncol' = number of vector.

Author(s)

Clement Walter
<clement.walter@cea.fr>

References

- V. Dubourg:
Meta-modeles adaptatifs pour l'analyse de fiabilite et l'optimisation sous containte fiabiliste
PhD Thesis, Universite Blaise Pascal - Clermont II,2011

- V. Dubourg, B. Sudret, F. Deheeger:
Metamodel-based importance sampling for structural reliability analysis Original Research Article
Probabilistic Engineering Mechanics, Volume 33, July 2013, Pages 47-57

- V. Dubourg, B. Sudret:
Metamodel-based importance sampling for reliability sensitivity analysis.
Accepted for publication in Structural Safety, special issue in the honor of Prof. Wilson Tang.(2013)

- V. Dubourg, B. Sudret and J.-M. Bourinet:
Reliability-based design optimization using kriging surrogates and subset simulation.
Struct. Multidisc. Optim.(2011)

See Also

`SubsetSimulation` `MonteCarlo` `km` (in package **DiceKriging**)

Examples

```
#Limit state function defined by Kiureghian & Dakessian :
kiureghian = function(x, b=5, kappa=0.5, e=0.1) {
x = as.matrix(x)
b - x[2,] - kappa*(x[1,]-e)^2
}

## Not run: res = MetaIS(dimension=2,limit_state_function=kiureghian,plot=TRUE)

#Compare with crude Monte-Carlo reference value
N = 500000
U = matrix(rnorm(dimension*N),dimension,N)
G = apply(U,2,kiureghian)
P = mean(G<0)
```

```
cov = sqrt((1-P)/(N*P))

## End(Not run)

#See impact of kernel choice with Waarts function :
waarts = function(x) {
x = as.matrix(x)
apply(x, 2, function(u) {min(
   (3+(u[1]-u[2])^2/10 - (u[1]+u[2])/sqrt(2)),
(3+(u[1]-u[2])^2/10 + (u[1]+u[2])/sqrt(2)),
u[1]-u[2]+7/sqrt(2),
u[2]-u[1]+7/sqrt(2))})
}

## Not run:
res = list()
res$matern5_2 = MetaIS(2,waarts,plot=TRUE)
res$matern3_2 = MetaIS(2,waarts,kernel="matern3_2",plot=TRUE)
res$gaussian = MetaIS(2,waarts,kernel="gauss",plot=TRUE)
res$exp = MetaIS(2,waarts,kernel="exp",plot=TRUE)

#Compare with crude Monte-Carlo reference value
N = 500000
U = matrix(rnorm(dimension*N),dimension,N)
G = apply(U,2,waarts)
P = mean(G<0)
cov = sqrt((1-P)/(N*P))

## End(Not run)
```

`ModifCorrMatrix` *Modification of a correlation matrix to use in UtoX*

Description

`ModifCorrMatrix` modifies a correlation matrix originally defined using SPEARMAN correlation coefficients to the correlation matrix to be used in the NATAF transformation performed in `UtoX`.

Usage

```
ModifCorrMatrix(Rs)
```

Arguments

`Rs`	Original correlation matrix defined using SPEARMAN correlation coefficient : $R_s = [\rho_{ij}^s]$

Value

R0	Modified correlation matrix

Note

The NATAF distribution is reviewed from the (normal) copula viewpoint as a particular and convenient means to describe a joint probabilistic model assuming that the normal copula fits to the description of the input X. The normal copula is defined by a symmetric positive definite matrix R0. Even though the off-diagonal terms in this matrix are comprised in]-1; 1[and its diagonal terms are equal to 1, it shall not be confused with the more usual correlation matrix. Lebrun and Dutfoy point out that the SPEARMAN (or rank) correlation coefficient is better suited to parametrize a copula because it leads to a simpler closed-form expression for ρ_{ij}.

Author(s)

Gilles DEFAUX, <gilles.defaux@cea.fr>

References

- M. Lemaire, A. Chateauneuf and J. Mitteau. Structural reliability, Wiley Online Library, 2009
- Lebrun, R. and A. Dutfoy. A generalization of the Nataf transformation to distributions with elliptical copula. Prob. Eng. Mech., 24(2), 172-178.
- V. Dubourg, Meta-modeles adaptatifs pour l'analyse de fiabilite et l'optimisation sous containte fiabiliste, PhD Thesis, Universite Blaise Pascal - Clermont II,2011

See Also

UtoX

Examples

```
Dim <- 2
input.Rho    <- matrix( c(1.0, 0.5,
                          0.5, 1.0),nrow=Dim)
input.R0     <- ModifCorrMatrix(input.Rho)
print(input.R0)
```

MonteCarlo *Crude Monte Carlo method*

Description

Estimate a failure probability using crude Monte Carlo method

Usage

```
MonteCarlo(dimension,
          LimitStateFunction,
          N_max         = 5e+05,
          N_batch       = 1000,
          failure       = 0,
          precision     = 0.05,
          plot          = FALSE,
          output_dir    = NULL,
          verbose       = 0)
```

Arguments

dimension	an integer giving the dimension of the input space.
LimitStateFunction	the failure function.
N_max	Maximum number of calls to the LimiStateFunction.
N_batch	Size of batch for Monte-Carlo population increase.
failure	the value defining the failure domain F = { x | LimitStateFunction(x) < failure }.
precision	A target maximum value of the coefficient of variation
plot	a boolean parameter specifying if function and samples should be plotted. Note that this option is only to be used when working on 'light' limit state functions as it requires the calculus of this function on a grid of size 161x161 (plot is done a -8:8 x -8:8 grid with 161 meshes.
output_dir	optional. If plots are to be saved in .jpeg in a given directory. This variable will be pasted with "_Monte_Carlo_brut.jpeg" to get the full output directory.
verbose	Eiher 0 for an almost no output message, or 1 for medium size or 2 for full size

Value

An object of class list containing the failure probability and some more outputs as described below:

proba	The estimated failure probability.
cov	The coefficient of variation of the Monte-Carlo probability estimate.
Ncall	The total number of calls to the LimitStateFunction.

Note

Problem is supposed to be defined in the standard space. If not, use UtoX to do so. Furthermore, each time a set of vector is defined as a matrix, 'nrow' = dimension and 'ncol' = number of vector.

Author(s)

Clement Walter
<clement.walter@cea.fr>

See Also

SubsetSimulation

Examples

```
#Try Naive Monte Carlo on a given function with different failure level
#Limit state function defined by Kiureghian & Dakessian :
kiureghian = function(x, b=5, kappa=0.5, e=0.1) {b - x[2] - kappa*(x[1]-e)^2}

## Not run:
res = list()
res[[1]] = MonteCarlo(2,kiureghian,failure = 0,plot=TRUE)
res[[2]] = MonteCarlo(2,kiureghian,failure = 1,plot=TRUE)
res[[3]] = MonteCarlo(2,kiureghian,failure = -1,plot=TRUE)


## End(Not run)


#Try Naive Monte Carlo on a given function and change number of points.
#Limit state function defined by Waarts :
waarts = function(u) { min(
   (3+(u[1]-u[2])^2/10 - (u[1]+u[2])/sqrt(2)),
(3+(u[1]-u[2])^2/10 + (u[1]+u[2])/sqrt(2)),
u[1]-u[2]+7/sqrt(2),
u[2]-u[1]+7/sqrt(2))
}
## Not run:
res = list()
res[[1]] = MonteCarlo(2,waarts,N_max = 10000)
res[[2]] = MonteCarlo(2,waarts,N_max = 100000)
res[[3]] = MonteCarlo(2,waarts,N_max = 500000)

## End(Not run)
```

MRM *MRM method*

Description

Calculate failure probability by MRM method.

Usage

```
MRM(f, ndim, choice.law, dir.monot, N.calls, Method, ordre.p = 0, silent = TRUE)
```

Arguments

f	the failure fonction
ndim	dimension of the inputs
choice.law	a list of length 'ndim' which contain name of input distribution and their parameters. For the input 'i', choice.law[[i]] = list('name_law',c(parameters1,..., parametersN))
dir.monot	vector of size ndim which represent the monotonicity of the failure function. dir.monot[i] = -1 (resp. 1) if the code is decreasing (resp. increasing) in direction i.
N.calls	Number of calls to f allowed
Method	there are two methods available. "MC_monotone"" is an adapation of the Monte Carlo method under constraints of monotony. "MRM" is a sequential sampling method.
ordre.p	order of magnitude of the search probability (for ndim >= 3)
silent	if silent = TRUE, print curent number of call to f. Default: FALSE.

Details

These methods compute the probability that the output of the failure function is negative

ordre.p is useful to compute the bounds in dimension >= 3, its computation is a santard Monte Carlo method. For dimension 2, there is an exat computation without sampling.

Value

Um	Exact lower bounds of the failure probability
UM	Exact upper bounds of the failure probability
MLE	Maximum likelihood estimator of the failure probability
IC.inf	Lower bound of the confidence interval of the failure probability based on MLE
IC.sup	Upper bound of the confidence interval of the failure probability based on MLE
CV.MLE	Coefficient of variation of the MLE
N.tot	Total number of simulation (just for "MC_monotone")

Author(s)

Vincent Moutoussamy and Nicolas Bousquet

References

Bousquet, N. (2012) Accelerated monte carlo estimation of exceedance probabilities under monotonicity constraints. Annales de la Faculte des Sciences de Toulouse. XXI(3), 557-592.

Examples

```
## Not run:
 choice.law <- list()
 choice.law[[1]] <- list("norm",c(4,1))
 choice.law[[2]] <- list("norm",c(0,1))

 ndim <- 2
 dir.monot <- c(1, -1)
 N.calls <- 80

  f <- function(Input){
    return(Input[1] - Input[2])
  }

  res.MRM <- MRM(f, ndim, choice.law, dir.monot, N.calls, Method = "MRM",
                 ordre.p = 0, silent = FALSE)

  N <- 1:dim(res.MRM)[1]

  plot(N, res.MRM[, 1],
       col = "blue", lwd=2, type='l', ylim=c(0,1e-2),
       xlab="Number of call to the failure function",
       ylab="")
  lines(N, res.MRM[, 2], col = "blue", lwd = 2)
  lines(N, res.MRM[, 3], col = "red", lwd = 2)
  lines(N, res.MRM[, 4], col = "orange", lwd = 2, lty = 2)
  lines(N, res.MRM[, 5], col = "orange", lwd = 2, lty = 2)
  legend("topright",
          c("Exact Bounds", "MLE", "Confidence interval"),
          col = c("blue", "red", "orange"),
          text.col = c("blue", "red", "orange"),
          lty = c(1, 1, 2),
          merge = TRUE)


  #It can be long:
  res.MC_monotone <- MRM(f, ndim, choice.law, dir.monot, N.calls, Method = "MC_monotone",
                         ordre.p = 0, silent = FALSE)

## End(Not run)
```

S2MART *Subset by Support vector Margin Algorithm for Reliability esTimation*

Description

S2MART lets easily combine metamodel based reliability algorithm SMART and subset simulation algorithm. Basically, it introduces a metamodeling step at each subset simulation threshold, making number of necessary samples lower and the probability estimation better according to subset simulation by itself.

Usage

```
S2MART(dimension,
      limit_state_function,
      Nn          = 100,
      alpha_quantile  = 0.1,
      ...,
      plot       = FALSE,
      limited_plot = FALSE,
      output_dir    = NULL,
      verbose    = 0)
```

Arguments

dimension	an integer giving the dimension of the input space.
limit_state_function	the failure fonction.
Nn	the number of samples to evaluate the quantiles in the subset step.
alpha_quantile	cutoff probability for the subsets.
...	parameters of SMART algorithm.
plot	a boolean parameter specifying if function and samples should be plotted. The plot is refreshed at each iteration with the new data. Note that this option is only to be used when working on 'light' limit state functions as it requires the calculus of this function on a grid of size 161x161 (plot is done a -8:8 x -8:8 grid with 161 meshes.
limited_plot	only a final plot with limit_state_function, final DOE and metamodels. Should be used with plot==FALSE. As for plot it requires the calculus of the limit_state_function on a grid of size 161x161.
output_dir	optional. If plots are to be saved in .jpeg in a given directory. This variable will be pasted with "_Subset.jpeg" to get the full output directory.
verbose	Eiher 0 for an almost no output message, or 1 for medium size or 2 for full size

Details

S2MART algorithm is based on the idea that subset simulations conditional probabilities are estimated with a relatively poor precision as it requires calls to the expensive-to-evaluate limit state function and does not take benefit from its numerous calls to the limit state function in the Metropolis-Hastings algorithm. In this scope, the key concept is to reduce the subset simulation population to its minimum and use it only to estimate crudely the next quantile. Then the use of a metamodel-based algorithm lets refine the border and calculate an accurate estimation of the conditional probability by the mean of a crude Monte-Carlo.

In this scope, a compromise has to be found between the two sources of calls to the limit state function as total number of calls = (`Nn` + number of calls to refine the metamodel) x (number of subsets) :

- `Nn` calls to find the next threshold value : the bigger `Nn`, the more accurate the 'decreasing speed' specified by the `alpha_quantile` value and so the smaller the number of subsets
- total number of calls to refine the metamodel at each threshold

Value

An object of class `list` containing the failure probability and some more outputs as described below:

`proba`	The estimated failure probability.
`cov`	The coefficient of variation of the Monte-Carlo probability estimate.
`Ncall`	The total number of calls to the `limit_state_function`.
`learn_db`	The final learning database, ie. all points where `limit_state_function` has been calculated.
`lsf_value`	The value of the `limit_state_function` on the learning database.
`meta_model`	The final metamodel. An S4 object from **DiceKriging**.

Note

Problem is supposed to be defined in the standard space. If not, use `UtoX` to do so.

Author(s)

Clement Walter
<clement.walter@cea.fr>

References

- J.-M. Bourinet, F. Deheeger, M. Lemaire:
Assessing small failure probabilities by combined Subset Simulation and Support Vector Machines
Structural Safety (2011)
- F. Deheeger:
Couplage m?cano-fiabiliste : 2SMART - m?thodologie d'apprentissage stochastique en fiabilit?
PhD. Thesis, Universit? Blaise Pascal - Clermont II, 2008

- S.-K. Au, J. L. Beck:
  Estimation of small failure probabilities in high dimensions by Subset Simulation
  Probabilistic Engineering Mechanics (2001)
- A. Der Kiureghian, T. Dakessian:
  Multiple design points in first and second-order reliability
  Structural Safety, vol.20 (1998)
- P.-H. Waarts:
  Structural reliability using finite element methods: an appraisal of DARS: Directional Adaptive Response Surface Sampling
  PhD. Thesis, Technical University of Delft, The Netherlands, 2000

See Also

SMART SubsetSimulation MonteCarlo km (in package **DiceKriging**) svm (in package **e1071**)

Examples

```
#Limit state function defined by Kiureghian & Dakessian :
kiureghian = function(x, b=5, kappa=0.5, e=0.1) {
x = as.matrix(x)
b - x[2,] - kappa*(x[1,]-e)^2
}

## Not run:
res = S2MART(dimension = 2,
            limit_state_function = kiureghian,
            N1 = 1000, N2 = 5000, N3 = 10000,
            plot = TRUE)

#Compare with crude Monte-Carlo reference value
reference = MonteCarlo(2, kiureghian, N_max = 500000)

## End(Not run)

#See impact of metamodel-based subset simulation with Waarts function :
waarts = function(x) {
x = as.matrix(x)
apply(x, 2, function(u) {min(
   (3+(u[1]-u[2])^2/10 - (u[1]+u[2])/sqrt(2)),
(3+(u[1]-u[2])^2/10 + (u[1]+u[2])/sqrt(2)),
u[1]-u[2]+7/sqrt(2),
u[2]-u[1]+7/sqrt(2))})
}

## Not run:
res = list()
res$SMART = SMART(dimension  = 2, limit_state_function = waarts, plot=TRUE)
res$S2MART = S2MART(dimension = 2,
                    limit_state_function = waarts,
                    N1 = 1000, N2 = 5000, N3 = 10000,
                    plot=TRUE)
```

```
res$SS = SubsetSimulation(dimension = 2, waarts, n_init_samples = 10000)
res$MC = MonteCarlo(2, waarts, N_max = 500000)

## End(Not run)
```

SMART *Support-vector Margin Algoritm for Reliability esTimation*

Description

Calculate a failure probability with SMART method.

Usage

```
SMART(dimension,
      limit_state_function,
      N1             = 10000,
      N2              = 50000,
      N3              = 200000,
      Nu               = 50,
      lambda1       = 7,
      lambda2       = 3.5,
      lambda3        = 1,
      tune_cost       = c(1,10,100,1000),
      tune_gamma       = c(0.5,0.2,0.1,0.05,0.02,0.01),
      clusterInMargin = TRUE,
      alpha_margin     = 1,
      k1              = round(6*(dimension/2)^(0.2)),
      k2              = round(12*(dimension/2)^(0.2)),
      k3              = k2 + 16,
      learn_db        = NULL,
      lsf_value       = NULL,
      failure         = 0,
      limit_fun_MH     = NULL,
      sampling_strategy = "MH",
      seeds         = NULL,
      seeds_eval       = NULL,
      burnin         = 30,
      thinning       = 4,
      plot           = FALSE,
      limited_plot     = FALSE,
      add           = FALSE,
      output_dir       = NULL,
      z_MH           = NULL,
      z_lsf         = NULL,
      verbose       = 0)
```

Arguments

dimension	an integer giving the dimension of the input space.
limit_state_function	the failure fonction.
N1	an integer defining the number of uniform samples for (L)ocalisation stage.
N2	an integer defining the number of uniform samples for (S)tabilisation stage.
N3	an integer defining the number of gaussian standard samples for (C)onvergence stage, and so Monte-Carlo population size.
Nu	an integer defining the size of the first Design Of Experiment got by uniforme sampling in a sphere of radius the maximum norm of N3 standard samples.
lambda1	a real defining the relaxing paramater in the Metropolis-Hastings algorithm for stage L.
lambda2	a real defining the relaxing paramater in the Metropolis-Hastings algorithm for stage S.
lambda3	a real defining the relaxing paramater in the Metropolis-Hastings algorithm for stage C. This shouldn't be modified as Convergence stage population is used to estimate failure probability.
tune_cost	a vector containing proposed values for the cost parameter of the SVM.
tune_gamma	a vector containing proposed values for the gamma parameter of the SVM.
clusterInMargin	margin points to be evaluated during refinements steps are got by mean of clustering of the N1, N2 or N3 points lying in the margin. Thus, they are not necessarily located into the margin. This boolean, if TRUE, enforces the selection of margin points by selecting points randomly in each cluster.
alpha_margin	a real value defining the margin. While 1 is the 'real' margin for a SVM, one can decide here to stretch it a bit.
k1	Rank of the first iteration of step S (ie stage L from 1 to k1-1).
k2	Rank of the first iteration of step C (ie stage S from k1 to k2-1).
k3	Rank of the last iteration of step C (ie stage C from k2 to k3).
learn_db	optional. A matrix of already known points, with dim : dimension x number_of_vector.
lsf_value	values of the limit state function on the vectors given in learn_db.
failure	the value defining the failure domain F = { x | limit_state_function(x) < failure }.
limit_fun_MH	optional. If the working space is to be reduced to some subset defining by a function, eg. in case of use in a Subset Simulation algorithm. As for the limit_state_function, failure domain is defined by points whom values of limit_fun_MH are negative.
sampling_strategy	either "AR" or "MH", to specify which sampling strategy is to be used when generating Monte-Carlo population in a case of subset simulation : "AR" stands for 'accept-reject' while "MH" stands for Metropolis-Hastings.

seeds	optional. If sampling_strategy=="MH", seeds from which MH algorithm starts. This should be a matrix with 'nrow' = dimension and 'ncol' = number of vector.
seeds_eval	optional. The value of the limit_fun_MH on the seeds.
burnin	a burnin parameter for Metropolis-Hastings algorithm. This is used only for the last C step population while it is set to 0 elsewhere.
thinning	a thinning parameter for Metropolis-Hastings algorithm. This is used only for the last C step population while it is set to 0 elsewhere. thinning = 0 means no thinning.
plot	a boolean parameter specifying if function and samples should be plotted. The plot is refreshed at each iteration with the new data. Note that this option is only to be used when working on 'light' limit state functions as it requires the calculus of this function on a grid of size 161x161 (plot is done a -8:8 x -8:8 grid with 161 meshes.
limited_plot	only a final plot with limit_state_function, final DOE and metamodel. Should be used with plot==FALSE.
add	optional. "TRUE" if plots are to be added to the current active device.
output_dir	optional. If plots are to be saved in .jpeg in a given directory. This variable will be pasted with "_SMART.jpeg" to get the full output directory.
z_MH	optional. For plots, if metamodel has already been evaluated on the grid then z_MH (from outer function) can be provided to avoid extra computational time.
z_lsf	optional. For plots, if LSF has already been evaluated on the grid then z_lsf (from outer function) can be provided to avoid extra computational time.
verbose	Eiher 0 for an almost no output message, or 1 for medium size or 2 for full size

Details

SMART is a reliability method proposed by J.-M. Bourinet et al. It makes uses of a SVM-based metamodel to approximate the limit state function and calculate the failure probability with a crude Monte-Carlo method using the metamodel-based limit state function. As SVM is a classification method, it makes use of limit state function values to create two classes : greater and lower than the failure threshold. Then the border is taken as a surogate of the limit state function.

Concerning the refinement strategy, it distinguishes 3 stages, known as Localisation, Stalibilsation and Convergence stages. The first one is proposed to reduce the margin as much as possible, the second one focuses on switching points while the last one works on the final Monte-Carlo population and is designed to insure a strong margin ; see F. Deheeger PhD thesis for more information.

Value

An object of class list containing the failure probability and some more outputs as described below:

proba	The estimated failure probability.
cov	The coefficient of variation of the Monte-Carlo probability estimate.
gamma	The gamma value corresponding to the correlation between Monte-Carlo samples got from Metropolis-Hastings algorithm.

Ncall	The total number of calls to the limit_state_function.
learn_db	The final learning database, ie. all points where limit_state_function has been calculated.
lsf_value	The value of the limit_state_function on the learning database.
meta_fun	The metamodel approximation of the limit_state_function. A call output is a list containing the value and the standard deviation.
meta_model	The final metamodel.
points	Points in the failure domain according to the metamodel.
meta_eval	Evaluation of the metamodel on these points.
z_meta	If plot==TRUE, the evaluation of the metamodel on the plot grid.

Note

Problem is supposed to be defined in the standard space. If not, use UtoX to do so. Furthermore, each time a set of vector is defined as a matrix, 'nrow' = dimension and 'ncol' = number of vector.

Author(s)

Clement Walter
<clement.walter@cea.fr>

References

- J.-M. Bourinet, F. Deheeger, M. Lemaire:
Assessing small failure probabilities by combined Subset Simulation and Support Vector Machines
Structural Safety (2011)
- F. Deheeger:
Couplage mecano-fiabiliste : 2SMART - methodologie d'apprentissage stochastique en fiabilite
PhD. Thesis, Universite Blaise Pascal - Clermont II, 2008

See Also

SubsetSimulation MonteCarlo svm (in package **e1071**)

Examples

```
#Limit state function defined by Kiureghian & Dakessian :
kiureghian = function(x, b=5, kappa=0.5, e=0.1) {
x = as.matrix(x)
b - x[2,] - kappa*(x[1,]-e)^2
}

## Not run:
SMART_estim = SMART(dimension=2,limit_state_function=kiureghian,plot=TRUE)
MC_estim = MonteCarlo(2, kiureghian, N_max = 500000)
```

```
## End(Not run)

#Limit state function defined by Waarts :
waarts = function(x) {
x = as.matrix(x)
apply(x, 2, function(u) {min(
   (3+(u[1]-u[2])^2/10 - (u[1]+u[2])/sqrt(2)),
(3+(u[1]-u[2])^2/10 + (u[1]+u[2])/sqrt(2)),
u[1]-u[2]+7/sqrt(2),
u[2]-u[1]+7/sqrt(2))})
}

## Not run:
SMART_estim = SMART(dimension=2,limit_state_function=waarts,plot=TRUE)
MC_estim = MonteCarlo(2, waarts, N_max = 500000)

## End(Not run)
```

SubsetSimulation *Subset Simulation Monte-Carlo*

Description

Estimate a failure probability with a Monte-Carlo method applied on nested subsets.

Usage

```
SubsetSimulation(dimension,
                 limit_state_function,
                 proposal_pdf,
                 pdf            = dnorm,
                 rpdf           = rnorm,
                 cutoff_prob    = 0.1,
                 n_init_samples = 10000,
          burnin        = 20,
          thinning    = 4,
                 plot           = FALSE,
                 output_dir     = NULL,
           verbose      = 0)
```

Arguments

dimension	an integer giving the dimension of the input space.
limit_state_function	the failure function.
proposal_pdf	proposal PDF for Metropolis-Hastings algorithm. Default random walk is uniform on a radius of 2.
pdf	PDF of the input space to be used in Metropolis-Hastings algorithm.

rpdf	a random generator for the input space PDF.
cutoff_prob	the cut-off probability for each subset.
n_init_samples	number of samples to be used for each subset.
burnin	burnin parameter for Metropolis-Hastings algorithm.
thinning	thinning parameter for Metropolis-Hastings algorithm.
plot	a boolean parameter specifying if function and samples should be plotted. The plot is refreshed at each iteration with the new data. Note that this option is only to be used when working on 'light' limit state functions as it requires the calculus of this function on a grid of size 161x161 (plot is done a -8:8 x -8:8 grid with 161 meshes.
output_dir	optional. If plots are to be saved in .jpeg in a given directory. This variable will be pasted with "_Subset_Simulation.jpeg" to get the full output directory.
verbose	Eiher 0 for an almost no output message, or 1 for medium size or 2 for full size

Details

This algorithm uses the property of conditional probabilities on nested subsets to calculate a given probability defined by a limit state function.

It operates iteratively on 'populations' to estimate the quantile corresponding to a probability of cutoff_prob. Then, it generates samples conditionnaly to this threshold, until found threshold be lower than 0.

Finally, the estimate is the product of the conditional probabilities.

Value

An object of class list containing the failure probability and some more outputs as described below:

proba	The estimated failure probability.
cov	The coefficient of variation of the Monte-Carlo probability estimate.
Ncall	The total number of calls to the limit_state_function.

Note

Problem is supposed to be defined in the standard space. If not, use UtoX to do so. Furthermore, each time a set of vector is defined as a matrix, 'nrow' = dimension and 'ncol' = number of vector.

Author(s)

Clement Walter
<clement.walter@cea.fr>

References

- S.-K. Au, J. L. Beck:
Estimation of small failure probabilities in high dimensions by Subset Simulation
Probabilistic Engineering Mechanics (2001)

See Also

MonteCarlo S2MART

Examples

```
#Try Subset Simulation Monte Carlo on a given function and change number of points.
#Limit state function defined by Kiureghian & Dakessian :
kiureghian = function(x, b=5, kappa=0.5, e=0.1) {
x = as.matrix(x)
b - x[2,] - kappa*(x[1,]-e)^2
}

## Not run:
res = list()
res[[1]] = SubsetSimulation(2,kiureghian,n_init_samples=10000)
res[[2]] = SubsetSimulation(2,kiureghian,n_init_samples=100000)
res[[3]] = SubsetSimulation(2,kiureghian,n_init_samples=500000)

## End(Not run)

#Try Subset Simulation Monte Carlo on a given function with different failure level
#Limit state function defined by Waarts :
waarts = function(x) {
x = as.matrix(x)
apply(x, 2, function(u) {min(
   (3+(u[1]-u[2])^2/10 - (u[1]+u[2])/sqrt(2)),
(3+(u[1]-u[2])^2/10 + (u[1]+u[2])/sqrt(2)),
u[1]-u[2]+7/sqrt(2),
u[2]-u[1]+7/sqrt(2))})
}

## Not run:
res = list()
res[[1]] = SubsetSimulation(2,waarts,failure=0,plot=TRUE)
res[[2]] = SubsetSimulation(2,waarts,failure=1,plot=TRUE)
res[[3]] = SubsetSimulation(2,waarts,failure=-1,plot=TRUE)

## End(Not run)
```

UtoX *Iso-probabilistic transformation from U space to X space*

Description

UtoX performs as iso-probabilistic transformation from standardized space (U) to physical space (X) according to the NATAF transformation, which requires only to know the means, the standard deviations, the correlation matrix $\rho(Xi, Xj) = \rho_{ij}$ and the marginal distributions of Xi. In standard space, all random variables are uncorrelated standard normal distributed variables whereas

they are correlated and defined using the following distribution functions: Normal (or Gaussian), Lognormal, Uniform, Gumbel, Weibull and Gamma.

Usage

```
UtoX(U, input.margin, L0)
```

Arguments

U	a matrix containing the realisation of all random variables in U-space
input.margin	A list containing one or more list defining the marginal distribution functions of all random variables to be used
L0	the lower matrix of the Cholesky decomposition of correlation matrix R0 (result of ModifCorrMatrix)

Details

Supported distributions are :

- NORMAL: distribution, defined by its mean and standard deviation
- LOGNORMAL: distribution, defined by its internal parameters P1=meanlog and P2=sdlog (plnorm)
- UNIFORM: distribution, defined by its internal parameters P1=min and P2=max (punif)
- GUMBEL: distribution, defined by its internal parameters P1 and P2
- WEIBULL: distribution, defined by its internal parameters P1=shape and P2=scale (pweibull)
- GAMMA: distribution, defined by its internal parameters P1=shape and P2=scale (pgamma)
- BETA: distribution, defined by its internal parameters P1=shape1 and P2=shapze2 (pbeta)

Value

X	a matrix containing the realisation of all random variables in X-space

Author(s)

gilles DEFAUX, <gilles.defaux@cea.fr>

References

- M. Lemaire, A. Chateauneuf and J. Mitteau. Structural reliability, Wiley Online Library, 2009
- V. Dubourg, Meta-modeles adaptatifs pour l'analyse de fiabilite et l'optimisation sous containte fiabiliste, PhD Thesis, Universite Blaise Pascal - Clermont II,2011

See Also

ModifCorrMatrix, ComputeDistributionParameter

Examples

```
Dim = 2

distX1 <- list(type='Norm',  MEAN=0.0, STD=1.0, P1=NULL, P2=NULL, NAME='X1')
distX2 <- list(type='Norm',  MEAN=0.0, STD=1.0, P1=NULL, P2=NULL, NAME='X2')

input.margin <- list(distX1,distX2)
input.Rho    <- matrix( c(1.0, 0.5,
                          0.5, 1.0),nrow=Dim)
input.R0     <- ModifCorrMatrix(input.Rho)
L0           <- t(chol(input.R0))

lsf = function(U) {
    X <- UtoX(U, input.margin, L0)
    G <- 5.0 - 0.2*(X[1,]-X[2,])^2.0 - (X[1,]+X[2,])/sqrt(2.0)
    return(G)
}

u0 <- as.matrix(c(1.0,-0.5))
lsf(u0)
```

`WilksFormula` *Wilks formula*

Description

Compute Wilks formula of 1941 for setting size of sample for quantile estimation with confidence level or for tolerance intervals.

Usage

```
WilksFormula(alpha=0.95,beta=0.95,bilateral=FALSE,order=1)
```

Arguments

`alpha`	order of the quantile (default = 0.95)
`beta`	level of the confidence interval (default = 0.95)
`bilateral`	TRUE for bilateral quantile (default = unilateral = FALSE)
`order`	order of the Wilks formula (default = 1)

Value

`N`	The minimal sample size to apply Wilks formula

Author(s)

Paul Lemaitre and Bertrand Iooss

References

H.A. David and H.N. Nagaraja. Order statistics, Wiley, 2003.

W.T. Nutt and G.B. Wallis. Evaluation of nuclear safety from the outputs of computer codes in the presence of uncertainties. Reliability Engineering and System Safety, 83:57-77, 2004.

S.S. Wilks. Determination of Sample Sizes for Setting Tolerance Limits. Annals Mathematical Statistics, 12:91-96, 1941.

Examples

```
N <- WilksFormula(0.95,0.95,order=1)
print(N)
```

Index