Package 'mistral'

April 3, 2016

Type Package

Title Methods in Structural Reliability Analysis

Version 2.1.0

Date 2016-04-03

Author
Clement WALTER, Gilles DEFAUX, Bertrand IOOSS and Vincent MOUTOUSSAMY, with contributions from Nicolas BOUSQUET, Claire CANNAMELA and Paul LEMAITRE

Maintainer Bertrand Iooss <biooss@yahoo.fr>

Depends R (>= 3.0.0)

Imports e1071, Matrix, mvtnorm, ggplot2, doParallel, foreach, iterators, DiceKriging, emoa, quadprog

Suggests microbenchmark, kernlab

Description Various reliability analysis methods for rare event inference:
1) computing failure probability (probability that the output of a numerical model exceeds a threshold),
2) computing quantiles of low or high-order,
3) Wilks formula to compute quantile(s) from a sample or the size of the required i.i.d. sample.

License CeCILL

NeedsCompilation no

Repository CRAN

LazyData true

RoxygenNote 5.0.0

Date/Publication 2016-04-03 17:13:58

R topics documented:

mistral-package 2
AKMCS 4
ComputeDistributionParameter 7
FORM 8
IRW 10

kiureghian 12
LSVM 13
MetaIS 14
modelLSVM 18
ModifCorrMatrix 20
MonotonicQuantileEstimation 21
MonteCarlo 23
MP 25
MRM 28
plotLSVM 30
quantileWilks 32
rackwitz 33
S2MART 34
SMART 36
SubsetSimulation 40
testConvexity 43
twodof 44
updateLSVM 45
UtoX 47
waarts 49
WilksFormula 49

Index **51**

`mistral-package` *Methods In Structural Reliability Analysis*

Description

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

Details

Package:	mistral	
Type:	Package	
Version:	2.1.0 Date:	2016-04-03
License:	CeCILL	

This package provides tools for structural reliability analysis:

- Calculate failure probability with FORM method and importance sampling,
- 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,

- Wilks formula: Compute a quantile (or tolerance interval) with a given confidence level from a i.i.d. sample,
- Wilks formula: Compute the minimal sample size to estimate a quantile with a given confidence level,
- Calculate a quantile under monotonicity constraints.

Author(s)

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

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.

J. Morio and M. Balesdent. Estimation of rare event probabilities in complex aerospace and other systems, WP, 2016.

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))

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

 res <- FORM(f, u.dep = c(0,0.1), inputDist = 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, lsf, N = 5e+05, N1 = 10 * dimension, Nmax = 200,
  learn_db = NULL, lsf_value = NULL, failure = 0, precision = 0.05,
  bayesian = TRUE, meta_model = NULL, kernel = "matern5_2",
  learn_each_train = TRUE, crit_min = 2, lower.tail = TRUE,
  limit_fun_MH = NULL, failure_MH = 0, sampling_strategy = "MH",
  first_DOE = "Gaussian", seeds = NULL, seeds_eval = limit_fun_MH(seeds),
  burnin = 30, plot = FALSE, limited_plot = FALSE, add = FALSE,
  output_dir = NULL, verbose = 0)
```

Arguments

dimension	dimension of the input space.
lsf	the function defining the failure/safety domain.
N	Monte-Carlo population size.
N1	size of the first DOE.
Nmax	maximum number of calls to the LSF.
learn_db	coordinates of already known points.
lsf_value	value of the LSF on these points.
failure	failure threshold.
precision	maximum desired cov on the Monte-Carlo estimate.
bayesian	estimate the conditional expectation E_X [P[meta(X)<failure]].
meta_model	provide here a kriging metamodel from km if wanted.
kernel	specify the kernel to use for km.
learn_each_train	specify if kernel parameters are re-estimated at each train.
crit_min	minimum value of the criteria to be used for refinement.
lower.tail	as for pxxxx functions, TRUE for estimating P(lsf(X) < failure), FALSE for P(lsf(X) > failure)
limit_fun_MH	define an area of exclusion with a limit function.
failure_MH	the theshold for the limit_fun_MH function.
sampling_strategy	either MH for Metropolis-Hastings of AR for accept-reject.

first_DOE	either Gaussian or Uniform, to specify the population on which clustering is done.
seeds	if some points are already known to be in the appropriate subdomain.
seeds_eval	value of the metamodel on these points.
burnin	burnin parameter for MH.
plot	set to TRUE for a full plot, ie refresh at each iteration.
limited_plot	set to TRUE for a final plot with final DOE, metamodel and LSF.
add	if plots are to be added to a current device.
output_dir	if plots are to be saved in jpeg in a given directory.
verbose	either 0 for almost no output, 1 for medium size output and 2 for all outputs.

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:

p	the estimated failure probability.
cov	the coefficient of variation of the Monte-Carlo probability estimate.
Ncall	the total number of calls to the lsf.
learn_db	the final learning database, ie. all points where lsf has been calculated.
lsf_value	the value of the lsf on the learning database.
meta_fun	the metamodel approximation of the lsf. A call output is a list containing the value and the standard deviation.
meta_model	the final metamodel. An S4 object from **DiceKriging**. Note that the algorithm enforces the problem to be the estimation of P[lsf(X)<failure] and so using 'predict' with this object will return inverse values if lower.tail==FALSE; in this scope prefer using directly meta_fun which handles this possible issue.
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 to be consistent with `as.matrix` transformation of a vector.

Algorithm calls lsf(X) (where X is a matrix as defined previously) and expects a vector in return. This allows the user to optimise the computation of a batch of points, either by vectorial computation, or by the use of external codes (optimised C or C++ codes for example) and/or parallel computation; see examples in MonteCarlo.

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` `MetaIS` `km` (in package **DiceKriging**)

Examples

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

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

## End(Not run)
```

```
#See impact of kernel choice with serial function from Waarts:
waarts = function(u) {
  u = as.matrix(u)
  b1 = 3+(u[1,]-u[2,])^2/10 - sign(u[1,] + u[2,])*(u[1,]+u[2,])/sqrt(2)
  b2 = sign(u[2,]-u[1,])*(u[1,]-u[2,])+7/sqrt(2)
  val = apply(cbind(b1, b2), 1, min)
}

## 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
dimension = 2
U = matrix(rnorm(dimension*N),dimension,N)
G = waarts(U)
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, inputDist, N.calls, eps = 1e-7,
     Method = "HLRF", IS = FALSE, q = 0.5, copula = "unif")
```

Arguments

f	A failure fonction
u.dep	A vector, starting point to the research of the design point
inputDist	A list which contains the name of the input distribution and their parameters. For the input "i", inputDistribution[[i]] = list("name_law",c(parameters1,..., parametersN))
N.calls	Number of calls to f allowed
eps	Stop criterion : distance of two points between two iterations
Method	Choice of the method to research 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 provides the importance density). Default = "FALSE".
q	Ratio of N.calls for the research 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)

Details

This function estimate the probability that the output of the failure function is negative using FORM algorithm. The importance sampling procedure estimate a probability using a Gaussian distribution centered in the design point with a covariance matrix equal to the indentity.

Value

pf	Failure probability
beta	Reliability index (beta)
compt.f	Number of calls to f
design.point	Coordinates of the design point
fact.imp	Importance factors
variance	Standard error of the probability estimator (if IS = TRUE)
conf	Confidence interval of the estimator at 0.95 (if IS = TRUE)
x	A data frame containing the input design of experiments
y	A vector of model responses (corresponding to x)
dy	A data frame of model response derivatives (wrt each input and corresponding to x); for the IS sample, the derivatives are not computed

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

```
## Not run:
 distribution = list()
 distribution[[1]] = list("gamma",c(2,1))
 distribution[[2]] = list("gamma",c(3,1))

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

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

names(res)
print(res)
print(res$pf)

## End(Not run)
```

IRW *Increasing Randow Walk*

Description

Simulate the increasing random walk associated with a real-valued continuous random variable.

Usage

```
IRW(dimension, lsf, N = 10, q = Inf, Nevent = Inf, particles,
  LSF_particles = lsf(particles), K, burnin = 20, sigma = 0.3,
  last.return = TRUE, use.potential = TRUE, plot = FALSE,
  print_plot = FALSE, output_dir = NULL)
```

Arguments

dimension	dimension of the input space.
lsf	limit state function.
N	number of particules.
q	level until which the randow walk is to be generated.
Nevent	the number of desired events.
particles	to start with some given particles.
LSF_particles	value of the lsf on these particles.
K	kernel transition for conditional generations.
burnin	burnin parameter.
sigma	radius parameter for K.
last.return	if the last event should be returned.
use.potential	tu use a 'potential' matrix to select starting point not directly related to the sample to be moved with the MH algorithm.
plot	if TRUE, the algorithm plots the evolution of the particles. This requieres to evaluate the lsf on a grid and is only for visual purpose.
print_plot	if TRUE, print the updated plot after each iteration. This might be slow; use with a small N. Otherwise it only prints the final plot.
output_dir	if plots are to be saved in pdf in a given directory. This will be pasted with '_IRW.pdf'. Together with print_plot==TRUE this will produce a pdf with a plot at each iteration, enabling 'video' reconstitution of the algorithm.

Details

This function lets generate the increasing random walk associated with a continous real-valued random variable of the form `Y = lsf(X)` where `X` is vectorial random variable.

This random walk can be associated with a Poisson process with parameter `N` and hence the number of iterations before a given threshold `q` is directly related to P[lsf(X) > q]. It is the core tool of algorithms such as nested sampling, Last Particle Algorithm or Tootsie Pop Algorithm.

Bascially for `N = 1`, it generates a sample $Y = lsf(X)$ and iteratively regenerates greater than the sought value: $Y_{n+1} \sim \mu^Y(\cdot \mid Y > Y_n$. This regeneration step is done with a Metropolis-Hastings algorithm and that is why it is usefull to consider generating several chains all together (`N > 1`).

The algorithm stops when it has simulated the required number of events `Nevent` or when it has reached the sought threshold `q`.

Value

An object of class `list` containing the following data:

`L`	the events of the random walk.
`M`	the total number of iterations.
`Ncall`	the total number of calls to the `lsf`.
`particles`	a matrix containing the final particles.
`LSF_particles`	the value of `lsf` on the `particles`.
`q`	the threshold considered when generating the random walk.
`Nevent`	the target number of events when generating the random walk.
`Nwmoves`	the number of rejected transitions, ie when the proposed point was not stricly greater/lower than the current state.
`acceptance`	a vector containing the acceptance rate for each use of the MH algorithm.

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 to be consistent with `as.matrix` transformation of a vector.

Algorithm calls lsf(X) (where X is a matrix as defined previously) and expects a vector in return. This allows the user to optimise the computation of a batch of points, either by vectorial computation, or by the use of external codes (optimised C or C++ codes for example) and/or parallel computation; see examples in MonteCarlo.

Author(s)

Clement WALTER <clement.walter@cea.fr>

References

- C. Walter:
Moving Particles: a parallel optimal Multilevel Splitting method with application in quantiles estimation and meta-model based algorithms
Structural Safety, 55, 10-25.

- C. Walter:
Point Process-based Monte Carlo estimation
arXiv preprint arXiv:1412.6368.

- J. Skilling:
Nested sampling for general Bayesian computation
Bayesian Analysis, 1(4), 833-859.

- M. Huber \& S. Schott:
Using TPA for Bayesian inference
Bayesian Statistics 9, 9, 257.

- A. Guyader, N. Hengartner and E. Matzner-Lober:
Simulation and estimation of extreme quantiles and extreme probabilities
Applied Mathematics \& Optimization, 64(2), 171-196.

See Also

MP

Examples

```
# Get faililng samples for the kiureghian limit state function
# Failure is defined as lsf(X) < 0 so we have to invert the lsf
lsf <- function(x) -1*kiureghian(x)
## Not run:
fail.samp <- IRW(2, lsf, q = 0, N = 10, plot = TRUE)

## End(Not run)
```

`kiureghian` *A limit-state-function defined by Der Kiureghian*

Description

The limit-state function is defined by:

$$f(x) = b - x_2 - \kappa * (x_1 - e)^2$$

with $b = 5$, $\kappa = 0.5$ and $e = 0.1$.

Usage

```
kiureghian
```

Format

The function can handle a vector or matrix with column vectors.

References

Der Kiureghian, A and Dakessian, T:
Multiple design points in first and second-order reliability
Structural Safety, 20, 1, 37-49, 1998.

LSVM *Linear Support Vector Machine under monotonicity constraints*

Description

Produce a globally increasing binary classifier built from linear monotonic SVM

Usage

```
LSVM(x, A.model.lsvm, convexity)
```

Arguments

`x`	a set of points where the class must be estimated.
`A.model.lsvm`	a matrix containing the parameters of all hyperplanes.
`convexity`	Either -1 if the set of data associated to the label "-1" is convex or +1 otherwise.

Details

LSVM is a monotonic binary classifier built from linear SVM under the constraint that one of the two classes of data is convex.

Value

An object of class `integer` representing the class of x

`res`	A vector of -1 or +1.

Author(s)

Vincent Moutoussamy

References

- R.T. Rockafellar:
Convex analysis
Princeton university press, 2015.

- N. Bousquet, T. Klein and V. Moutoussamy :
Approximation of limit state surfaces in monotonic Monte Carlo settings
Submitted .

See Also

modelLSVM

Examples

```
# A limit state function
f <- function(x){  sqrt(sum(x^2)) - sqrt(2)/2 }

# Creation of the data sets
n <- 200
X <- matrix(runif(2*n), nrow = n)
Y <- apply(X, MARGIN = 1, function(w){sign(f(w))})

#The convexity is known
## Not run:
  model.A <- modelLSVM(X, Y, convexity = -1)
  m <- 10
  X.test <- matrix(runif(2*m), nrow = m)
  classOf.X.test <- LSVM(X.test, model.A, convexity = -1)

## End(Not run)
```

MetaIS *Metamodel based Impotance Sampling*

Description

Estimate failure probability by MetaIS method.

Usage

```
MetaIS(dimension, lsf, N = 5e+05, N_alpha = 100, N_DOE = 10 * dimension,
  N1 = N_DOE * 30, Ru = 8, Nmin = 30, Nmax = 200, Ncall_max = 1000,
  precision = 0.05, N_seeds = 2 * dimension, Niter_seed = 10000,
  N_alphaLOO = 5000, K_alphaLOO = 2 * dimension, alpha_int = c(0.1, 10),
  k_margin = 1.96, lower.tail = TRUE, learn_db = NULL, lsf_value = NULL,
```

```
failure = 0, meta_model = NULL, kernel = "matern5_2",
learn_each_train = TRUE, limit_fun_MH = NULL, failure_MH = 0,
sampling_strategy = "MH", seeds = NULL,
seeds_eval = limit_fun_MH(seeds), burnin = 20, plot = FALSE,
limited_plot = FALSE, add = FALSE, output_dir = NULL, verbose = 0)
```

Arguments

dimension	of the input space
lsf	the failure defining the failure/safety domain
N	size of the Monte-Carlo population for P_epsilon estimate
N_alpha	initial size of the Monte-Carlo population for alpha estimate
N_DOE	size of the initial DOE got by clustering of the N1 samples
N1	size of the initial uniform population sampled in a hypersphere of radius Ru
Ru	radius of the hypersphere for the initial sampling
Nmin	minimum number of call for the construction step
Nmax	maximum number of call for the construction step
Ncall_max	maximum number of call for the whole algorithm
precision	desired maximal value of cov
N_seeds	number of seeds for MH algoritm while generating into the margin (according to MP*gauss)
Niter_seed	maximum number of iteration for the research of a seed for alphaLOO refinement sampling
N_alphaLOO	number of points to sample at each refinement step
K_alphaLOO	number of clusters at each refinement step
alpha_int	range for alpha to stop construction step
k_margin	margin width; default value means that points are classified with more than 97,5%
lower.tail	specify if one wants to estimate P[lsf(X)<failure] or P[lsf(X)>failure].
learn_db	Coordinates of alredy known points
lsf_value	Value of the LSF on these points
failure	Failure threshold
meta_model	Provide here a kriging metamodel from km if wanted
kernel	Specify the kernel to use for km
learn_each_train	Specify if kernel parameters are re-estimated at each train
limit_fun_MH	Define an area of exclusion with a limit function
failure_MH	Threshold for the limit_MH function
sampling_strategy	Either MH for Metropolis-Hastings of AR for accept-reject
seeds	If some points are already known to be in the appropriate subdomain

seeds_eval	Value of the metamodel on these points
burnin	Burnin parameter for MH
plot	Set to TRUE for a full plot, ie refresh at each iteration
limited_plot	Set to TRUE for a final plot with final DOE, metamodel and LSF
add	If plots are to be added to a current device
output_dir	If plots are to be saved in jpeg in a given directory
verbose	Either 0 for almost no output, or 1 for medium size or 2 for all outputs

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:

p	The estimated failure probability.
cov	The coefficient of variation of the Monte-Carlo probability estimate.
Ncall	The total number of calls to the lsf.
learn_db	The final learning database, ie. all points where lsf has been calculated.
lsf_value	The value of the lsf on the learning database.
meta_fun	The metamodel approximation of the lsf. A call output is a list containing the value and the standard deviation.
meta_model	The final metamodel. An S4 object from **DiceKriging**. Note that the algorithm enforces the problem to be the estimation of P[lsf(X)<failure] and so using 'predict' with this object will return inverse values if lower.tail==FALSE; in this scope prefer using directly meta_fun which handle this possible issue.
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 to be consistent with `as.matrix` transformation of a vector.

Algorithm calls lsf(X) (where X is a matrix as defined previously) and expects a vector in return. This allows the user to optimise the computation of a batch of points, either by vectorial computation, or by the use of external codes (optimised C or C++ codes for example) and/or parallel computation; see examples in MonteCarlo.

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

```
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,lsf=kiureghian,plot=TRUE)
```

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

## End(Not run)

#See impact of kernel choice with Waarts function :
waarts = function(u) {
  u = as.matrix(u)
  b1 = 3+(u[1,]-u[2,])^2/10 - sign(u[1,] + u[2,])*(u[1,]+u[2,])/sqrt(2)
  b2 = sign(u[2,]-u[1,])*(u[1,]-u[2,])+7/sqrt(2)
  val = apply(cbind(b1, b2), 1, min)
}

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

## End(Not run)
```

`modelLSVM` *Estimation of the parameters of the LSVM*

Description

Produce a matrix containing the parameters of a set of hyperplanes separating the two classes of data

Usage

```
modelLSVM(X, Y, convexity)
```

Arguments

X	a matrix containing the data sets
Y	a vector containing -1 or +1 that reprensents the class of each elements of X.
convexity	Either -1 if the set of data associated to the label "-1" is convex or +1 otherwise.

Details

modelLSVM evaluate the classifier on a set of points.

Value

An object of class `matrix` containing the parameters of a set of hyperplanes

res	A matrix where each lines contains the parameters of a hyperplane.

Author(s)

Vincent Moutoussamy

References

- R.T. Rockafellar:
  Convex analysis
  Princeton university press, 2015.

- N. Bousquet, T. Klein and V. Moutoussamy :
  Approximation of limit state surfaces in monotonic Monte Carlo settings
  Submitted .

See Also

LSVM

Examples

```
# A limit state function
f <- function(x){  sqrt(sum(x^2)) - sqrt(2)/2 }

# Creation of the data sets
n <- 200
X <- matrix(runif(2*n), nrow = n)
Y <- apply(X, MARGIN = 1, function(w){sign(f(w))})

#The convexity is known
## Not run:
  model.A <- modelLSVM(X, Y, convexity = -1)
```

```
## 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)
```

MonotonicQuantileEstimation

Quantile estimation under monotonicity constraints

Description

Estimate a quantile with the constraints that the function is monotone

Usage

```
MonotonicQuantileEstimation(f,
                            inputDimension,
                            inputDistribution,
                            dir.monot,
                            N.calls,
                            p,
                            method,
                            X.input = NULL,
                            Y.input = NULL)
```

Arguments

f	a failure fonction
inputDimension	dimension of the inputs
inputDistribution	a list of length 'inputDimension' which contains the name of the input distribution and their parameters. For the input "i", inputDistribution[[i]] = list("name_law",c(parameters1,..., parametersN))
dir.monot	vector of size inputDimension which represents the monotonicity of the failure function. dir.monot[i] = -1 (resp. 1) if the failure function f is decreasing (resp. increasing) according with direction i.
N.calls	Number of calls to f allowed

method	there are four methods available. "MonteCarloWB" provides the empirical quantile estimator, "MonteCarloWB" provides the empirical quantile estimator as well as two bounds for the searched quantile, "Bounds" provides two bounds for a quantile from a set of points and "MonteCarloIS" provides an estimate of a quantile based on a sequential framework of simulation.
p	the probability associated to the quantile
X.input	a set of points
Y.input	value of f on X.input

Details

MonotonicQuantileEstimation provides many methods to estimate a quantile under monotonicity constraints.

Value

An object of class `list` containing the quantile as well as:

qm	A lower bound of the quantile.
qM	A upperer bound of the quantile.
q.hat	An estimate of the quantile.
Um	A lower bounds of the probability obtained from the desing of experiments.
UM	An upper bounds of the probability obtained from the desing of experiments.
XX	Design of experiments
YY	Values of on XX

Note

Inputs X.input and Y.input are useful only for method = "Bounds"

Author(s)

Vincent Moutoussamy

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:
 inputDistribution <- list()
 inputDistribution[[1]] <- list("norm",c(4,1))
 inputDistribution[[2]] <- list("norm",c(0,1))

 inputDimension <- length(inputDistribution)
```

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

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

probability <- 1e-2

trueQuantile <- qnorm(probability,
                    inputDistribution[[1]][[2]][1] - inputDistribution[[2]][[2]][1],
                  sqrt(inputDistribution[[1]][[2]][2] + inputDistribution[[1]][[2]][2]))

resQuantile <- MonotonicQuantileEstimation(f, inputDimension, inputDistribution,
                              dir.monot, N.calls, p = probability, method = "MonteCarloIS")

quantileEstimate <- resQuantile[[1]][N.calls, 3]


## End(Not run)
```

`MonteCarlo` *Crude Monte Carlo method*

Description

Estimate a failure probability using a crude Monte Carlo method.

Usage

```
MonteCarlo(dimension, lsf, N_max = 5e+05, N_batch = 1000, q = 0,
  lower.tail = TRUE, precision = 0.05, plot = FALSE, output_dir = NULL,
  verbose = 0)
```

Arguments

`dimension`	the dimension of the input space.
`lsf`	the function defining safety/failure domain.
`N_max`	maximum number of calls to the `lsf`.
`N_batch`	number of points onte evalutae the `lsf` at each iteration.
`q`	the quantile
`lower.tail`	as for pxxxx functions, TRUE for estimating P(lsf(X) < q), FALSE for P(lsf(X) > q)
`precision`	a targeted maximum value for the coefficient of variation.
`plot`	to plot the contour of the `lsf` as well as the generated samples.
`output_dir`	to save a copy of the plot in a pdf. This name will be pasted with "_Monte_Carlo_brut.pdf"
`verbose`	to control the level of outputs in the console; either 0 or 1 or 2 for almost no outputs to a high level output.

Details

This implementation of the crude Monte Carlo method works with evaluating batchs of points sequentialy until a given precision is reached on the final estimator

Value

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

`p`	the estimated probabilty.
`ecdf_MC`	the empiracal cdf got with the generated samples.
`cov`	the coefficient of variation of the Monte Carlo estimator.
`Ncall`	the total numnber of calls to the `lsf`, ie the total number of generated samples.
`X`	the generated samples.
`Y`	the value `lsf(X)`.

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 to be consistent with `as.matrix` transformation of a vector.

Algorithm calls lsf(X) (where X is a matrix as defined previously) and expects a vector in return. This allows the user to optimise the computation of a batch of points, either by vectorial computation, or by the use of external codes (optimised C or C++ codes for example) and/or parallel computation.

Author(s)

Clement WALTER <clement.walter@cea.fr>

References

- R. Rubinstein and D. Kroese:
  Simulation and the Monte Carlo method
  Wiley (2008)

See Also

`SubsetSimulation` `foreach`

Examples

```
#First some considerations on the usage of the lsf.
#Limit state function defined by Kiureghian & Dakessian :
# Remember you have to consider the fact that the input will be a matrix ncol >= 1
lsf_wrong = function(x, b=5, kappa=0.5, e=0.1) {
  b - x[2] - kappa*(x[1]-e)^2 # work only with a vector of lenght 2
```

```
}
lsf_correct = function(x){
  apply(x, 2, lsf_wrong)
}
lsf = function(x, b=5, kappa=0.5, e=0.1) {
  x = as.matrix(x)
  b - x[2,] - kappa*(x[1,]-e)^2 # vectorial computation, run fast
}

y = lsf(X <- matrix(rnorm(20), 2, 10))
#Compare running time
## Not run:
  require(microbenchmark)
  X = matrix(rnorm(2e5), 2)
  microbenchmark(lsf(X), lsf_correct(X))

## End(Not run)

#Example of parallel computation
require(doParallel)
lsf_par = function(x){
 foreach(x=iter(X, by='col'), .combine = 'c') %dopar% lsf(x)
}

#Try Naive Monte Carlo on a given function with different failure level
## Not run:
  res = list()
  res[[1]] = MonteCarlo(2,lsf,q = 0,plot=TRUE)
  res[[2]] = MonteCarlo(2,lsf,q = 1,plot=TRUE)
  res[[3]] = MonteCarlo(2,lsf,q = -1,plot=TRUE)


## End(Not run)


#Try Naive Monte Carlo on a given function and change number of points.
## Not run:
  res = list()
  res[[1]] = MonteCarlo(2,lsf,N_max = 10000)
  res[[2]] = MonteCarlo(2,lsf,N_max = 100000)
  res[[3]] = MonteCarlo(2,lsf,N_max = 500000)

## End(Not run)
```

MP *Moving Particles*

Description

This function runs the Moving Particles algorithm for estimating extreme probability and quantile.

Usage

```
MP(dimension, lsf, N = 100, N.batch = 1, p, q, lower.tail = TRUE,
  Niter_1fold, alpha = 0.05, compute_confidence = FALSE, verbose = 0,
  chi2 = FALSE, breaks = N.batch/5, ...)
```

Arguments

dimension	the dimension of the input space.
lsf	the function defining the RV of interest Y = lsf(X).
N	the total number of particles,
N.batch	the number of parallel batches for the algorithm. Each batch will then have N/N.batch particles. Typically this could be detectCores() or some other machine-derived parameters.
p	a given probability to estimate the corresponding quantile (as in qxxxx functions).
q	a given quantile to estimate the corresponding probability (as in pxxxx functions).
lower.tail	as for pxxxx functions, TRUE for estimating P(lsf(X) < q), FALSE for P(lsf(X) > q).
Niter_1fold	a function = fun(N) giving the deterministic number of iterations for the first pass.
alpha	when using default Niter_1fold function, this is the risk not to have simulated enough samples to produce a quantile estimator.
compute_confidence	if TRUE, the algorithm runs a little bit longer to produces a 95% interval on the quantile estimator.
verbose	to control level of print (either 0, or 1, or 2).
chi2	for a chi2 test on the number of events.
breaks	for the final histogram is chi2 == TRUE.
...	further arguments past to IRW.

Details

MP is a wrap up of IRW for probability and quantile estimation. By construction, the several calls to IRW are parallel (**foreach**) and so is the algorithm. Especially, with N.batch=1, this is the Last Particle Algorithm, which is a specific version of SubsetSimulation with p_0 = 1-1/N. However, note that this algorithm not only gives a quantile or a probability estimate but also an estimate of the whole cdf until the given threshold q.

The probability estimator only requires to generate several random walks as it is the estimation of the parameter of a Poisson random variable. The quantile estimator is a little bit more complicated and requires a 2-passes algorithm. It is thus not exactly fully parallel as cluster/cores have to communicate after the first pass. During the first pass, particles are moved a given number of times, during the second pass particles are moved until the farthest event reach during the first pass. Hence, the random process is completely simulated until this given state.

For an easy user experiment, all the parameters are defined by default with the optimised values as described in the reference paper (see References below) and a typical use will only specify `N` and `N.batch`.

Value

An object of class `list` containing the outputs described below:

`p`	the estimated probability or the reference for the quantile estimate.
`q`	the estimated quantile or the reference for the probability estimate.
`ecdf_MP`	the empirical cdf.
`L_max`	the farthest state reached by the random process. Validity range for the `ecdf_MP` is then (-Inf, L_max] or [L_max, Inf).
`times`	the *times* of the random process.
`Ncall`	the total number of calls to the `lsf`.
`particles`	the `N` particles in their final state.
`LSF_particles`	the value of the `lsf(particles)`.
`moves`	a vector containing the number of moves for each batch.
`p_int`	a 95% confidence intervalle on the probability estimate.
`q_int`	a 95% confidence intervall on the quantile estimate.
`chi2`	the output of the chisq.test function.

Note

The `alpha` parameter is set to 0.05 by default. Indeed it should not be set too small as it is defined approximating the Poisson distribution with the Gaussian one. However if no estimate is produce then the algorithm can be restarted for the few missing events. In any cases, setting `Niter_1fold = -N/N.batch*log(p)` gives 100% chances to produces a quantile estimator.

Author(s)

Clement WALTER <clement.walter@cea.fr>

References

- A. Guyader, N. Hengartner and E. Matzner-Lober:
Simulation and estimation of extreme quantiles and extreme probabilities
Applied Mathematics \& Optimization, 64(2), 171-196.

- C. Walter:
Moving Particles: a parallel optimal Multilevel Splitting method with application in quantiles estimation and meta-model based algorithms
Structural Safety, 55, 10-25.

- E. Simonnet:
Combinatorial analysis of the adaptive last particle method
Statistics and Computing, 1-20.

See Also

SubsetSimulation MonteCarlo IRW

Examples

```
## Not run:
# Estimate some probability and quantile with the parabolic lsf
p.est <- MP(2, kiureghian, N = 100, q = 0) # estimate P(lsf(X) < 0)
p.est <- MP(2, kiureghian, N = 100, q = 7.8, lower.tail = FALSE) # estimate P(lsf(X) > 7.8)

q.est <- MP(2, kiureghian, N = 100, p = 1e-3) # estimate q such that P(lsf(X) < q) = 1e-3
q.est <- MP(2, kiureghian, N = 100, p = 1e-3, lower.tail = FALSE) # estimate q such
# that P(lsf(X) > q) = 1e-3


# plot the empirical cdf
plot(xplot <- seq(-3, p.est$L_max, l = 100), sapply(xplot, p.est$ecdf_MP))

# check validity range
p.est$ecdf_MP(p.est$L_max - 1)
# this example will fail because the quantile is greater than the limit
tryCatch({
   p.est$ecdf_MP(p.est$L_max + 0.1)},
   error = function(cond) message(cond))

# Run in parallel
library(doParallel)
registerDoParallel()
p.est <- MP(2, kiureghian, N = 100, q = 0, N.batch = getDoParWorkers())

## End(Not run)
```

MRM *MRM method*

Description

Estimate a failure probability by MRM method.

Usage

```
MRM(f, inputDimension, inputDistribution, dir.monot, N.calls, Method, silent = FALSE)
```

Arguments

f	a failure fonction
inputDimension	dimension of the inputs

`inputDistribution`
: a list of length 'inputDimension' which contains the name of the input distribution and their parameters. For the input "i", inputDistribution[[i]] = list("name_law",c(parameters1,..., parametersN))

`dir.monot`
: vector of size `inputDimension` which represents the monotonicity of the failure function. dir.monot[i] = -1 (resp. 1) if the failure function f is decreasing (resp. increasing) according with direction i.

`N.calls`
: Number of calls to f allowed

`Method`
: there is two methods available. "MC" is an adapation of the Monte Carlo method under constraints of monotony. "MRM" is based on a sequential sampling.

`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

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

`X`
: design of experiments

`Y`
: value of f on X

`N.tot`
: Total number of simulation (only 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:

  inputDistribution <- list()
  inputDistribution[[1]] <- list("norm",c(4,1))
  inputDistribution[[2]] <- list("norm",c(0,1))
  inputDistribution[[3]] <- list("norm",c(-1,3))

  inputDimension <- length(inputDistribution)
```

```
p <- 1e-5

threshold <- qnorm(p, 3, sqrt(11))

f <- function(Input){
  sum(Input) - threshold
}

dir.monot <- c(1, 1, 1)

N.calls <- 300

res.MRM <- MRM(f, inputDimension, inputDistribution,
                dir.monot, N.calls, Method = "MRM", silent = FALSE)

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

plot(N, res.MRM[[1]][, 1],
      col = "black", lwd=2, type='l', ylim=c(0, 50*p),
      xlab="Number of runs to the failure function",
      ylab="")
lines(N, res.MRM[[1]][, 2], col = "black", lwd = 2)
lines(N, res.MRM[[1]][, 3], col = "red", lwd = 2)
lines(N, res.MRM[[1]][, 7], col = "blue", lwd = 2, lty = 2)
lines(N, rep(p, length(N)), lwd= 2, col= "orange", lty=3 )
legend("topright",
        c("Exact Bounds", "MLE","p.hat", "p"),
        col = c("black", "red", "blue", "orange"),
        text.col = c("black", "red", "blue", "orange"),
        lty = c(1, 1, 2, 3),
        merge = TRUE)


## End(Not run)
```

---

plotLSVM *plot of LSVM*

Description

Make a plot of the data and the LSVM classifier

Usage

```
plotLSVM(X,
        Y,
        A.model.lsvm,
        hyperplanes = FALSE,
```

```
            limit.state.estimate = TRUE,
            convexity)
```

Arguments

X	a matrix containing the data sets
Y	a vector containing -1 or +1 that reprensents the class of each elements of X.
A.model.lsvm	a matrix containing the parameters of all hyperplanes.
hyperplanes	A boolean. If TRUE, plot the hyperplanes obtained.
limit.state.estimate	A boolean. If TRUE, plot the estimate of the limit state.
convexity	Either -1 if the set of data associated to the label "-1" is convex or +1 otherwise.

Details

plotLSVM makes a plot of the data as well as the estimate limit state and the hyperplanes involved in this construction.

Note

This function is useful only in dimension 2.

Author(s)

Vincent Moutoussamy

References

- R.T. Rockafellar:
  Convex analysis
  Princeton university press, 2015.

- N. Bousquet, T. Klein and V. Moutoussamy :
  Approximation of limit state surfaces in monotonic Monte Carlo settings
  Submitted .

See Also

LSVM modelLSVM

Examples

```
# A limit state function
f <- function(x){  sqrt(sum(x^2)) - sqrt(2)/2 }

# Creation of the data sets
```

```
n <- 200
X <- matrix(runif(2*n), nrow = n)
Y <- apply(X, MARGIN = 1, function(w){sign(f(w))})

## Not run:
  model.A <- modelLSVM(X,Y, convexity = -1)
  plotLSVM(X, Y, model.A, hyperplanes = FALSE, limit.state.estimate = TRUE, convexity = -1)

## End(Not run)
```

`quantileWilks` *Computing quantiles with the Wilks formula*

Description

From the Wilks formula, compute a quantile (or a tolerance interval) with a given confidence level from a i.i.d. sample, or compute the minimal sample size to estimate a quantile (or a tolerance interval) with a given confidence level.

Usage

```
quantileWilks(alpha=0.95,beta=0.95,data=NULL,bilateral=FALSE)
```

Arguments

`alpha`	level of the unilateral or bilateral quantile (default = 0.95)
`beta`	level of the confidence interval on quantile value(s) (default = 0.95)
`data`	the data sample (vector format) to compute the quantile(s); if data=NULL (by default), the function returns the minimal sample size to compute the required quantile
`bilateral`	TRUE for bilateral quantile (default = unilateral = FALSE)

Value

4 output values if 'data' is specified; 1 output value (nmin) if 'data' is not specified

`lower`	lower bound of the bilateral tolerance interval; if bilateral=FALSE, no value
`upper`	upper bound of the tolerance interval (bilateral case) or quantile value (unilateral case)
`nmin`	minimal size of the required i.i.d. sample for given alpha and beta: - bilateral case: tolerance interval will be composed with the min and max of the sample; - unilateral case: the quantile will correspond to max of the sample.
`ind`	the index (unilateral case) or indices (bilateral case) of the quantiles in the ordered sample (increasing order)

Author(s)

Claire Cannamela 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 <- quantileWilks(alpha=0.95,beta=0.95)
print(N)
```

---

rackwitz *A limit-state-function defined by Rackwitz*

Description

The function is defined in the standard space and internal normal-lognormal transformation is done. Its definition with iid lognormal random variables is:

$$d + a\sigma\sqrt{d} - \sum_{i=1}^{d} x_i$$

Default values are: $a = 1$, `mean=1` and $\sigma = 0.2$.

Usage

```
rackwitz
```

Format

The function can handle a vector or a matrix with column vectors.

References

Rackwitz, R:
Reliability analysis: a review and some perspectives
Structural Safety, 23, 4, 365-395, 2001.

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

Description

S2MART 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, lsf, Nn = 100, alpha_quantile = 0.1, failure = 0, ...,
  plot = FALSE, output_dir = NULL, verbose = 0)
```

Arguments

dimension	the dimension of the input space
lsf	the function defining the failure domain. Failure is lsf(X) < failure
Nn	number of samples to evaluate the quantiles in the subset step
alpha_quantile	cutoff probability for the subsets
failure	the failure threshold
...	All others parameters of the metamodel based algorithm
plot	to produce a plot of the failure and safety domain. Note that this requires a lot of calls to the lsf and is thus only for training purpose
output_dir	to save the plot into the given directory. This will be pasted with "_S2MART.pdf"
verbose	either 0 for almost no output, 1 for medium size output and 2 for all outputs

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:

`p`	The estimated failure probability.
`cov`	The coefficient of variation of the Monte-Carlo probability estimate.
`Ncall`	The total number of calls to the `lsf`.
`learn_db`	The final learning database, ie. all points where `lsf` has been calculated.
`lsf_value`	The value of the `lsf` on the learning database.
`meta_model`	The final metamodel. An object from **e1071**.

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 to be consistent with `as.matrix` transformation of a vector.

Algorithm calls lsf(X) (where X is a matrix as defined previously) and expects a vector in return. This allows the user to optimise the computation of a batch of points, either by vectorial computation, or by the use of external codes (optimised C or C++ codes for example) and/or parallel computation; see examples in MonteCarlo.

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

```
## Not run:
  res = S2MART(dimension = 2,
               lsf = 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 :
## Not run:
  res = list()
  # SMART stands for the pure metamodel based algorithm targeting directly the
  # failure domain. This is not recommended by its authors which for this purpose
  # designed S2MART : Subset-SMART
  res$SMART = mistral:::SMART(dimension  = 2, lsf = waarts, plot=TRUE)
  res$S2MART = S2MART(dimension = 2,
                      lsf = 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. This should not be used by itself but only through S2MART.

Usage

```
SMART(dimension,
      lsf,
      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.
lsf	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**) `S2MART`

`SubsetSimulation` *Subset Simulation Monte Carlo*

Description

Estimate a probability of failure with the Subset Simulation algorithm (also known as Multilevel Splitting or Sequential Monte Carlo for rare events).

Usage

```
SubsetSimulation(dimension, lsf, p_0 = 0.1, N = 10000, q = 0,
  lower.tail = TRUE, K, burnin = 20, save.all = FALSE, plot = FALSE,
  output_dir = NULL, plot.lab = c("x", "y"), verbose = 0)
```

Arguments

`dimension`	the dimension of the input space.
`lsf`	the function defining failure/safety domain.
`p_0`	a cutoff probability for defining the subsets.
`N`	the number of samples per subset, ie the population size for the Monte Carlo estimation of each conditional probability.
`q`	the quantile defining the failure domain.
`lower.tail`	as for pxxxx functions, TRUE for estimating P(lsf(X) < q), FALSE for P(lsf(X) > q)
`K`	a transition Kernel for Markov chain drawing in the regeneration step. K(X) should propose a matrix of candidate sample (same dimension as X) on which `lsf` will be then evaluated and transition accepted of rejected. Default kernel is the one defined K(X) = (X + sigma*W)/sqrt(1 + sigma^2) with W ~ N(0, 1).
`burnin`	a burnin parameter for the the regeneration step.
`save.all`	if TRUE, all the samples generated during the algorithms are saved and return at the end. Otherwise only the working population is kept at each iteration.

plot	to plot the contour of the lsf and the generated sample.
output_dir	to save the plot into a pdf file. This variable will be paster with "_Subset_Simulation.pdf"
plot.lab	the x and y labels for the plot
verbose	Either 0 for almost no output, 1 for medium size output and 2 for all outputs

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 p_0. 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:

p	the estimated failure probability.
cov	the estimated coefficient of variation of the estimate.
Ncall	the total number of calls to the lsf.
X	the working population.
Y	the value lsf(X).
Xtot	if save.list==TRUE, all the Ncall samples generated by the algorithm.
Ytot	the value lsf(Xtot).
sigma.hist	if default kernel is used, sigma is initialized with 0.3 and then further adaptively updated to have an average acceptance rate of 0.3

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 to be consistent with as.matrix transformation of a vector.

Algorithm calls lsf(X) (where X is a matrix as defined previously) and expects a vector in return. This allows the user to optimise the computation of a batch of points, either by vectorial computation, or by the use of external codes (optimised C or C++ codes for example) and/or parallel computation; see examples in MonteCarlo.

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)

- A. Guyader, N. Hengartner and E. Matzner-Lober:
Simulation and estimation of extreme quantiles and extreme probabilities
Applied Mathematics \& Optimization, 64(2), 171-196.

- F. Cerou, P. Del Moral, T. Furon and A. Guyader:
Sequential Monte Carlo for rare event estimation
Statistics and Computing, 22(3), 795-808.

See Also

IRW MP MonteCarlo

Examples

```
#Try Subset Simulation Monte Carlo on a given function and change number of points.

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

## End(Not run)

# Compare SubsetSimulation with MP
## Not run:
p <- res[[3]]$p # get a reference value for p
p_0 <- 0.1 # the default value recommended by Au & Beck
N_mp <- 100
# to get approxumately the same number of calls to the lsf
N_ss <- ceiling(N_mp*log(p)/log(p_0))
comp <- replicate(50, {
ss <- SubsetSimulation(2, kiureghian, N = N_ss)
mp <- MP(2, kiureghian, N = N_mp, q = 0)
comp <- c(ss$p, mp$p, ss$Ncall, mp$Ncall)
names(comp) = rep(c("SS", "MP"), 2)
comp
})
boxplot(t(comp[1:2,])) # check accuracy
sd.comp <- apply(comp,1,sd)
print(sd.comp[1]/sd.comp[2]) # variance increase in SubsetSimulation compared to MP

colMeans(t(comp[3:4,])) # check similar number of calls
```

```
## End(Not run)
```

`testConvexity` *Test the convexity of set of data*

Description

Provides the

Usage

```
testConvexity(X,Y)
```

Arguments

`X`	a matrix containing the data sets
`Y`	a vector containing -1 or +1 that reprensents the class of each elements of X.

Details

testConvexity test if one of the two data set is potentially convex.

Value

An object of class `list` containing the number of the class which is convex and the parameters of a set of hyperplanes separating the two classes

Author(s)

Vincent Moutoussamy

References

- R.T. Rockafellar:
  Convex analysis
  Princeton university press, 2015.

See Also

`LSVM` `modelLSVM`

Examples

```
# A limit state function
f <- function(x){  sqrt(sum(x^2)) - sqrt(2)/2 }

# Creation of the data sets
n <- 200
X <- matrix(runif(2*n), nrow = n)
Y <- apply(X, MARGIN = 1, function(w){sign(f(w))})

## Not run:
  TEST.Convexity <- testConvexity(X, Y)
  if(length(TEST.Convexity) == 2){
    Convexity <- TEST.Convexity[[1]]
    model.A   <- TEST.Convexity[[2]]
  }
  if(length(TEST.Convexity) == 1){
    # The problem is not convex
    Convexity <- 0 #the problem is not convex
  }

## End(Not run)
```

---

twodof *A limit-state-function defined with a two degrees of freedom damped oscillator*

Description

The limit-state function is defined in the standard space and isoprobabilistic transformation is used internally.

Parameters `mean_Fs` and `p` can be specified and default are 27.5 and 3 respectively.

Usage

```
twodof
```

Format

The function can handle a vector or a matrix with column vectors.

References

Dubourg, V and Deheeger, F and Sudret, B:
Metamodel-based importance sampling for the simulation of rare events
arXiv preprint arXiv:1104.3476, 2011.

updateLSVM *Update LSVM classifier*

Description

Update the existing classifier LSVM with a new set of data.

Usage

```
updateLSVM(X.new,
           Y.new,
           X,
           Y,
           A.model.lsvm,
           convexity,
           PLOTSVM = FALSE,
           step.plot.LSVM = 1,
           hyperplanes = FALSE,
           limit.state.estimate = TRUE)
```

Arguments

X.new	a matrix containing a new data sets
Y.new	a vector containing -1 or +1 that reprensents the class of each elements of X.new.
X	a matrix containing the data sets
Y	a vector containing -1 or +1 that reprensents the class of each elements of X.
A.model.lsvm	a matrix containing the parameters of all hyperplanes.
convexity	Either -1 if the set of data associated to the label "-1" is convex or +1 otherwise.
PLOTSVM	A boolean. If TRUE, plot the data.
step.plot.LSVM	A plot is made each step.plot.LSVM steps.
hyperplanes	A boolean. If TRUE, plot the hyperplanes obtained.
limit.state.estimate	A boolean. If TRUE, plot the estimate of the limit state.

Details

updateLSVM allows to make an update of the classifier LSVM.

Value

An object of class matrix containing the parameters of a set of hyperplanes

Note

The argument PLOTSVM is useful only in dimension 2.

Author(s)

Vincent Moutoussamy

References

- R.T. Rockafellar:
  Convex analysis
  Princeton university press, 2015.

- N. Bousquet, T. Klein and V. Moutoussamy :
  Approximation of limit state surfaces in monotonic Monte Carlo settings
  Submitted .

See Also

LSVM modelLSVM

Examples

```
# A limit state function
f <- function(x){  sqrt(sum(x^2)) - sqrt(2)/2 }

# Creation of the data sets

n <- 200
X <- matrix(runif(2*n), nrow = n)
Y <- apply(X, MARGIN = 1, function(w){sign(f(w))})

## Not run:
  model.A <- modelLSVM(X,Y, convexity = -1)
  M <- 20
  X.new <- matrix(runif(2*M), nrow = M)
  Y.new <- apply(X.new, MARGIN = 1, function(w){ sign(f(w))})

  X.new.S <- X.new[which(Y.new > 0), ]
  Y.new.S <- Y.new[which(Y.new > 0)]
  model.A.new <- updateLSVM(X.new.S, Y.new.S, X, Y,
                            model.A, convexity = -1, PLOTSVM = TRUE, step.plot.LSVM = 5)

## 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

$$distX < -list(type = "Norm", MEAN = 0.0, STD = 1.0, NAME = "X1")$$

- LOGNORMAL: distribution, defined by its internal parameters P1=meanlog and P2=sdlog (plnorm)

$$distX < -list(type = "Lnorm", P1 = 10.0, P2 = 2.0, NAME = "X2")$$

- UNIFORM: distribution, defined by its internal parameters P1=min and P2=max (punif)

$$distX < -list(type = "Unif", P1 = 2.0, P2 = 6.0, NAME = "X3")$$

- GUMBEL: distribution, defined by its internal parameters P1 and P2

$$distX < -list(type =' Gumbel', P1 = 6.0, P2 = 2.0, NAME =' X4')$$

- WEIBULL: distribution, defined by its internal parameters P1=shape and P2=scale (pweibull)

$$distX < -list(type =' Weibull', P1 = NULL, P2 = NULL, NAME =' X5')$$

- GAMMA: distribution, defined by its internal parameters P1=shape and P2=scale (pgamma)

$$distX <- list(type =' Gamma', P1 = 6.0, P2 = 6.0, NAME =' X6')$$

- BETA: distribution, defined by its internal parameters P1=shape1 and P2=shapze2 (pbeta)

$$distX <- list(type =' Beta', P1 = 6.0, P2 = 6.0, NAME =' X7')$$

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)
```

waarts *A limit-state-function defined by Waarts*

Description

The limit-state function is defined by:

$$b1 = 3 + (u_1 - u_2)^2/10 - sign(u_1 + u_2) * (u_1 + u_2)/sqrt(2)$$

$$b2 = sign(u_2 - u_1) * (u_1 - u_2) + 7/sqrt(2)$$

$$f(u) = min(b1, b2)$$

Usage

```
waarts
```

Format

The function can handle a vector or matrix with column vectors.

References

Waarts, PH:
An appraisal of DARS: directional adaptive response surface sampling
Delft University Press, The Netherlands, 2000.

WilksFormula *Sample size by Wilks formula*

Description

Compute Wilks formula for setting size of a i.i.d. 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

∗Topic **datasets**
kiureghian, 12
rackwitz, 33
twodof, 44
waarts, 49
∗Topic **package**
mistral-package, 2

AKMCS, 4

ComputeDistributionParameter, 7, *48*

foreach, *24*
FORM, 8

IRW, 10, *26*, *28*, *42*

kiureghian, 12
km, *6*, *17*, *36*

LPA *(*MP*)*, 25
LSVM, 13, *19*, *31*, *43*, *46*

MetaIS, *6*, 14
mistral *(*mistral-package*)*, 2
mistral-package, 2
modelLSVM, *14*, 18, *31*, *43*, *46*
ModifCorrMatrix, 20, *47*, *48*
MonotonicQuantileEstimation, 21
MonteCarlo, *6*, *11*, *17*, 23, *28*, *35*, *36*, *40–42*
MP, *12*, 25, *42*
MRM, 28

NestedSampling *(*IRW*)*, 10

pbeta, *48*
pgamma, *48*
plnorm, *47*
plotLSVM, 30
punif, *47*
pweibull, *47*

quantileWilks, 32

rackwitz, 33

S2MART, 34, *40*
SMART, *36*, 36
ss *(*SubsetSimulation*)*, 40
subset *(*SubsetSimulation*)*, 40
SubsetSimulation, *6*, *17*, *24*, *26*, *28*, *36*, *40*, 40
svm, *36*, *40*

testConvexity, 43
TPA *(*IRW*)*, 10
twodof, 44

updateLSVM, 45
UtoX, *6*, *11*, *17*, *21*, *24*, *35*, *39*, *41*, 47

waarts, 49
WilksFormula, 49