Package ‘mistral’

April 19, 2021

Type Package
Title Methods in Structural Reliability
Version 2.2.1
Author Clement WALTER, Gilles DEFAUX, Bertrand IOOSS, 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, Rcpp
Suggests microbenchmark, deSolve, scatterplot3d, KrigInv, rgenoud, kernlab, knitr, rmarkdown, markdown
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 quantile 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.
LinkingTo Rcpp
License CeCILL
NeedsCompilation yes
Repository CRAN
LazyData true
RoxygenNote 7.0.2
VignetteBuilder knitr, rmarkdown
Date/Publication 2021-04-18 23:00:02 UTC

R topics documented:

<table>
<thead>
<tr>
<th>Package</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>mistral-package</td>
<td>2</td>
</tr>
<tr>
<td>AKMCS</td>
<td>4</td>
</tr>
<tr>
<td>BMP</td>
<td>8</td>
</tr>
<tr>
<td>cantilever</td>
<td>12</td>
</tr>
</tbody>
</table>
Index

mistral-package Methods In Structural Reliability Analysis

Description

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

Package: mistral
Type: Package
Version: 2.2.1 Date: 2016-12-21
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


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

########### FORM ###########
# u.dep is a starting point for the research of the Most Probable Failing Point
# N.calls is a total number of calls
form <- mistral::FORM(dimension = 2, mistral::kiureghian, N.calls = 1000,
                     u.dep = c(0,0))
form$p

# use IS=TRUE to use an Importance Sampling scheme with a Gaussian standard
# proposal distribution centred at the MPFP
form.IS <- mistral::FORM(dimension = 2, mistral::kiureghian, N.calls = 1000,
                        u.dep = c(0,0),
                        IS = TRUE)
form.IS$p

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

Description

Estimate a failure probability with the AKMCS method.

Usage

AKMCS(
    dimension,
    lsf,
N = 5e+05,
N1 = 10 * dimension,
Nmax = 200,
Nmin = 2,
X = NULL,
y = NULL,
failure = 0,
precision = 0.05,
bayesian = TRUE,
compute.PPP = FALSE,
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.
Nmin        minimum number of calls during enrichment step.
X          coordinates of already known points.
y          value of the LSF on these points.
failure    failure threshold.
precision   maximum desired cov on the Monte-Carlo estimate.
bayesian    estimate the conditional expectation $E_X [ P[\text{meta}(X)<\text{failure}] ]$.
compute.PPP to simulate a Poisson process at each iteration to estimate the conditional expectation and the SUR criteria based on the conditional variance: $h$ (average probability of misclassification at level $\text{failure}$) and $I$ (integral of $h$ over the whole interval $[\text{failure}, \infty]$).
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 \( P(lsf(X) < \text{failure}) \), FALSE for \( P(lsf(X) > \text{failure}) \)
limit_fun_MH define an area of exclusion with a limit function.
failure_MH   the threshold for the limit_fun_MH function.
sampling_strategy
             either MH for Metropolis-Hastings or AR for accept-reject.
first_DOE   either Gaussian or Uniform, to specify the population on which clustering is
done. Set to ”No” for no initial DoE (use together with a first DoE given in \( X \) for
instance).
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.
\( \text{cov} \) the coefficient of variation of the Monte-Carlo probability estimate.
Ncall the total number of calls to the lsf.
X the final learning database, ie. all points where lsf has been calculated.
y the value of the lsf on the learning database.
h the sequence of the estimated relative SUR criteria.
I the sequence of the estimated integrated SUR criteria.
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 <clementwalter@icloud.com>

References

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

• 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
BMP

Bayesian Moving Particles

Description

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

\[ \text{BMP(} \]
  \[ \text{dimension,} \]
  \[ \text{lsf,} \]
  \[ \text{q,} \]
  \[ N = 1000, \]
  \[ N.\text{final} = N, \]
  \[ N.\text{iter} = 30, \]
  \[ \text{adaptive} = \text{FALSE}, \]
  \[ N.\text{DoE} = 5 \times \text{dimension}, \]
  \[ \text{firstDoE} = \text{"uniform"}, \]
  \[ \text{radius} = \text{qnorm}(1e-10, \text{lower.tail} = \text{FALSE}), \]
  \[ X, \]
  \[ y, \]
  \[ \text{covariance} = \text{NULL}, \]
  \[ \text{learn}\_\text{each}\_\text{train} = \text{Inf}, \]
  \[ \text{km.param} = \text{list} (\text{nugget.estim} = \text{TRUE}, \text{multistart} = 1, \text{optim.method} = \text{"BFGS"}, \text{coef.trend} = \text{q}), \]
  \[ \text{burnin} = 20, \]
  \[ \text{fast} = \text{TRUE}, \]
  \[ \text{sur} = \text{list} (\text{integrated} = \text{TRUE}, r = 1, \text{approx.pnorm} = \text{FALSE}), \]
  \[ \text{lower.tail} = \text{TRUE}, \]
  \[ \text{save.dir}, \]
  \[ \text{plot} = \text{FALSE}, \]
  \[ \text{plot.lsf} = \text{TRUE}, \]
  \[ \text{plot.lab} = \text{c} (\text{"x}_1", \text{"x}_2"), \]
  \[ \text{chi2} = \text{FALSE}, \]
  \[ \text{verbose} = 1, \]
  \[ \text{breaks} \)

Arguments

- **dimension**: the dimension of the input space.
- **lsf**: the function defining the RV of interest \( Y = \text{lsf}(X) \).
- **q**: a given quantile to estimate the corresponding probability.
- **N**: the total number of Poisson processes during the refinement step.
- **N.final**: the total number of Poisson processes for the final alpha estimate.
- **N.iter**: the total number of iteration of the algorithm, ie that total number of calls to the \( \text{lsf} \) will be \( N.\text{DoE} + N.\text{iter} \times r \).
- **adaptive**: if the algorithm should stop automatically if the stopping criterion is verified, precisely the mean probability of misclassification of the particles being over a given threshold.
- **N.DoE**: the number of points for the initial Design of Experiment
- **firstDoE**: default is "uniform" for a random uniform sampling over a sphere of radius \( \text{radius} \). Also available "maximim" for a maximim LHS.
radius: the size of the radius of the sphere for uniform DoE or the semi length of the interval on each dimension for maximin LHS.

X: (optional) a first Design of Experiment to be used instead of building a new DoE.

y: the value of lsf on the X.

covariance: (optional) to give a covariance kernel for the km object.

learn_each_train: a integer: after this limit the covariance parameters are not learnt any more and the model is just updated with the new datapoints.

km.param: (optional) list of parameters to be passed to DiceKriging::km.

burnin: a burnin parameter for Markov Chain drawing of the metamodel based Poisson process (this does not change the number of calls to lsf).

fast: in current implementation it appears that the call to the metamodel is faster when doing batch computation. This parameter lets do the Markov chain the other way around: instead of first selecting a starting point and then applying burnin times the transition kernel, it creates a working population by apply the kernel to all the particles and then makes some moves with the generated discretised distribution.

sur: a list containing any parameters to be passed to estimateSUR. Default is sur$integrated=TRUE and sur$r=1 for a one step ahead integrated SUR criterion.

lower.tail: as for pxxxx functions, TRUE for estimating P(lsf(X) < q), FALSE for P(lsf(X) > q).

save.dir: (optional) a directory to save the X and y at each iteration.

plot: to plot the DoE and the updated model.

plot.lsf: to plot the contour of the true lsf. Note that this requires its evaluation on a grid and should be used only on toy examples.

plot.lab: the labels of the axis for the plot.

chi2: for a chi2 test on the number of events.

verbose: controls the level of outputs of the algorithm.

breaks: optional, for the final histogram if chi2 == TRUE.

Details

The Bayesian Moving Particles algorithm uses the point process framework for rare event to iteratively estimate the conditional expectation of the (random) limit-state function, to quantify the quality of the learning and to propose a new point to be added to the model with a SUR criterion.

Value

An object of class list containing the outputs described below:

alpha: the estimated conditional expectation of the probability.

alpha.seq: the sequence of estimated alpha during the refinement step.

cv2: an estimate of the squared coefficient of variation of alpha.
cv.seq  the sequence of the estimated coefficients of variations.

h  the sequence of the estimated upper bound of the conditional variance divided by estimated alpha.

I  the sequence of the estimated integrated h.

sur_min  a list containing the the sequence of corresponding thresholds and -log probability of the sample minimising the SUR criterion.

sur_stat  a list containing at each iterations number of points tried for the SUR criterion as well as the computational spent.

q  the reference quantile for the probability estimate.

ecdf  the empirical cdf, i.e. the estimation of the function q -> E(alpha(q)).

L_max  the farthest state reached by the random process. Validity range for the ecdf is then (-Inf, L_max] or [L_max, Inf).

PPP  the last Poisson process generated with N.final particles.

meta_fun  the metamodel approximation of the lsf. A call output is a list containing the value and the standard deviation.

model  the final metamodel. An S4 object from DiceKriging. Note that the algorithm enforces the problem to be the estimation of P[lsf(X)>q] and so using 'predict' with this object will return inverse values if lower.tail=TRUE; in this scope prefer using directly meta_fun which handles this possible issue.

model.first  the first metamodel with the intial DoE.

alpha_int  a 95% confidence intervalle on the estimate of alpha.

moves  a vector containing the number of moves for each one of the N.batch particles.

chi2  the output of the chisq.test function.

Note

Probleme should be defined in the standard space. Transformations can be made using UtoX and XtoU functions.

Author(s)

Clement WALTER <clementwalter@icloud.com>

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.

- J. Bect, L. Li and E. Vazquez: 
  *Bayesian subset simulation* 
cantilever

See Also

SubsetSimulation MonteCarlo IRW MP

Examples

# Estimate $P(g(X)<0)$
## Not run: p <- BMP(dimension = 2, lsf = kiureghian, q = 0, N = 100, N.iter = 30, plot = TRUE)

# More extreme event
## Not run: p <- BMP(dimension = 2, lsf = waarts, q = -4, N = 100, N.iter = 50, plot = TRUE)

# One can also estimate probability of the form $P(g(X)>q)$
## Not run: p <- BMP(dimension = 2, lsf = cantilever, q = 1/325, N = 100, N.iter = 30, plot = TRUE)

cantilever

A function calculating the deviation of a cantilever beam.

Description

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

Usage

cantilever

Format

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

References

Gayton, N. and Bourinet, J.-M. and Lemaire, M.: 
CD2RS: a new statistical approach to the response surface method for reliability analysis. 
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)

estimateSUR

EstimateSUR

Description

A function for estimating a SUR criterion with a realisation of a PPP
Usage

```r
estimateSUR(
  PPP,
  xi_PPP_X,
  integrated = TRUE,
  N_ppp,
  method = "discrete",
  SUR_pop,
  r = N.batch,
  optimcontrol = list(pop.size = 50 * d, max.generations = 10 * d),
  approx.pnorm,
  J = 0,
  N.batch = foreach::getDoParWorkers(),
  verbose = 0,
  ...
)
```

Arguments

- **PPP** the Poisson point process generated to get alpha.
- **xi_PPP_X** the output of `xi(cbind(PPP$X, PPP$final_X))`.
- **integrated** boolean to specify if SUR criterion is standard or integrated.
- **N_ppp** the number of Poisson processes used for the SUR criterion estimation.
- **method** either "genoud" for an optimisation using the package `rgenoud` or "discrete" for a discrete search over `SUR_pop`.
- **SUR_pop** if `optimcontrol$method=="discrete"`, `SUR_pop` is the population onto which minimizer is sought. Should be a matrix d x n.
- **r** number of points to be added to the DoE.
- **optimcontrol** a list of control parameters for the optimisation of the SUR criterion using the `rgenoud` package.
- **approx.pnorm** (optional) an approximation of base `pnorm` function running faster.
- **J** the center of an interval of size 8 for `pnorm` approximation.
- **N.batch** Number of batches for parallel computation.
- **verbose** to control the print level of the algorithm.
- **...** further arguments to be passed to `fSUR`.

Value

- a list containing the points minimising the criterion
**FORM**  

**First-order reliability method**

**Description**

The First-Order Reliability Method computes an estimation of the failure probability by approximating the limit-state function at the Most Probable Failure Point with a hyperplane.

**Usage**

```r
FORM(
  dimension,
  lsf,
  u.dep = rep(0, dimension),
  N.calls = 100,
  eps = 1e-07,
  Method = "HLRF",
  IS = FALSE,
  IS.ratio = 0.5,
  plot = FALSE,
  plot.lsf = FALSE,
  plot.lab = c("x_1", "x_2")
)
```

**Arguments**

- `dimension`: the dimension of the input space.
- `lsf`: the limit-state function.
- `u.dep`: the starting point for the MPFP search.
- `N.calls`: the total number of calls for the whole algorithm.
- `eps`: stopping criterion: distance of two points between two iterations.
- `Method`: choice of the method to search the design point: "AR" for Abdo-Rackwitz and "HLRF" for Hasofer-Lindt-Rackwitz-Fiessler.
- `IS`: "TRUE" for using importance Sampling method with an standard Gaussian importance density centred at the MPFP.
- `IS.ratio`: ratio of N.calls for the search of the design point by FORM. Default = 0.5. 1-IS.ratio = the remaining ratio to be used for importance sampling.
- `plot`: to plot the generated samples.
- `plot.lsf`: a boolean indicating if the lsf should be added to the plot. This requires the evaluation of the lsf over a grid and consequently should be used only for illustration purposes.
- `plot.lab`: the x and y labels for the plot.
Details

The FORM method has to be used in the standard Gaussian input space. It is designed to estimate probability of the form $P[g(X) < 0]$ with $g$ the limit-state function. This function has to be modified accordingly to fit into this framework.

Furthermore, it should be able to handle matrix input of column vectors. See the mistral vignette for more info about lsf definition.

Value

A list containing the following objects

- **p**: Failure probability
- **indice.reliab**: Reliability index
- **Ncall**: 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)
- **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 and Clement WALTER <clementwalter@icloud.com>

References


Examples

```r
## Not run:
# u.dep is a starting point for the research of the Most Probable Failing Point
# N.calls is a total number of calls
form <- mistral::FORM(dimension = 2, mistral::kiureghian, N.calls = 1000,
                     u.dep = c(0,0))
form$p

# use IS=TRUE to use an Importance Sampling scheme with a Gaussian standard
# proposal distribution centred at the MPFP
form.IS <- mistral::FORM(dimension = 2, mistral::kiureghian, N.calls = 1000,
                         u.dep = c(0,0),
                         IS = TRUE)
form.IS$p

## End(Not run)
```
FORMv0

FORM method (old version)

Description

Calculate failure probability by FORM method and important sampling.

Usage

FORMv0(f, u.dep, inputDist, N.calls, eps = 1e-7,
Method = "HLRF", IS = FALSE, q = 0.5, copula = "unif")

Arguments

f
A failure function

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 identity.

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)
generateK

Confidence interval of the estimator at 0.95 (if IS = TRUE)

A data frame containing the input design of experiments

A vector of model responses (corresponding to x)

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 and Bertrand Iooss

References


Examples

```r
## 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 <- mistral:::FORMv0(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)
```

### Description

Generate Standard Gaussian samples with a Gaussian transition kernel

### Usage

```r
generateK(X, N = 100, thinning = 4, sigma = 1, lsf, burnin = 20)
```
Arguments

- **X**
  - the seeds for the Markov Chain. There are as many MC drawn as given seeds
- **N**
  - the number of desired samples
- **thinning**
  - the proportion of kept samples, i.e., 1 each thinning draw.
- **sigma**
  - the exploration parameter for the transition kernel
- **lsf**
  - a boolean limit-state function for defining a subdomain of the input space.
- **burnin**
  - the burnin parameter, i.e., the number of discarded samples before keeping one.

Details

This function generates standard Gaussian samples with a Markov Chain using a suitable transition kernel.

Value

A matrix `X` with the number of desired samples

Author(s)

Clement WALTER <clementwalter@icloud.com>

Examples

```r
# Get a seed in dimension 2
X <- matrix(rnorm(2), nrow = 2)
X <- generateK(X, N = 1000)

library(ggplot2)
ggplot(as.data.frame(t(X)), aes(x_1,x_2)) + geom_point()

# One can also specify a limit-state function
lsf <- function(X){
  sqrt(colSums(X^2)) > 2
}
X <- matrix(c(2, 2), nrow = 2)
X <- generateK(X, N = 1000, lsf = lsf)

ggplot(as.data.frame(t(X)), aes(x_1,x_2)) + geom_point()
```
**Description**

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

**Usage**

```r
IRW(
  dimension,
  lsf,
  N = 10,
  q = Inf,
  Nevent = Inf,
  X,
  y = lsf(X),
  K,
  burnin = 20,
  sigma = 0.3,
  last.return = TRUE,
  use.potential = TRUE,
  plot = FALSE,
  plot.lsf = FALSE,
  print_plot = FALSE,
  output_dir = NULL,
  plot.lab = c("x_1", "x_2")
)
```

**Arguments**

- `dimension`: dimension of the input space.
- `lsf`: limit state function.
- `N`: number of particles.
- `q`: level until which the random walk is to be generated.
- `Nevent`: the number of desired events.
- `X`: to start with some given particles.
- `y`: value of the `lsf` on `X`.
- `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`: to 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 requires to evaluate the lsf on a grid and is only for visual purpose.

plot.lsf a boolean indicating if the lsf should be added to the plot. This requires the evaluation of the lsf over a grid and consequently should be used only for illustration purposes.

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.

plot.lab the x and y labels for the plot

Details

This function lets generate the increasing random walk associated with a continuous real-valued random variable of the form $Y = \text{lsf}(X)$ 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 $P[\text{lsf}(X) > q]$. It is the core tool of algorithms such as nested sampling, Last Particle Algorithm or Tootsie Pop Algorithm.

Basically for $N = 1$, it generates a sample $Y = \text{lsf}(X)$ and iteratively regenerates greater than the found 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 useful to consider generating several chains all together ($N > 1$).

The algorithm stops when it has simulated the required number of events $N_{\text{event}}$ 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.
- $N_{\text{call}}$ the total number of calls to the lsf.
- $X$ a matrix containing the final particles.
- $y$ the value of lsf on $X$.
- $q$ the threshold considered when generating the random walk.
- $N_{\text{event}}$ the target number of events when generating the random walk.
- $N_{\text{moves}}$ the number of rejected transitions, i.e., when the proposed point was not strictly 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 \texttt{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 \texttt{as.matrix} transformation of a vector.

Algorithm calls \texttt{lsf(X)} (where \texttt{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 \texttt{MonteCarlo}.

Author(s)

Clement WALTER <clementwalter@icloud.com>

References

- C. Walter:  
  \textit{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:  
  \textit{Point Process-based Monte Carlo estimation}  

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

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

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

See Also

\texttt{MP}

Examples

```r
# Get failing 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)
```
kiureghian

## 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

```r
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*  

---

**LSVM**

*Linear Support Vector Machine under monotonicity constraints*

### Description

Produce a globally increasing binary classifier built from linear monotonic SVM

### Usage

```r
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

\[
\text{res} \quad \text{A vector of -1 or +1.}
\]

Author(s)

Vincent Moutoussamy

References

- R.T. Rockafellar:  
  *Convex analysis*  

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

See Also

modelLSVM

Examples

```r
# 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 Importance 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 = Inf,
    N_alphaLOO = 5000,
    K_alphaLOO = 1,
    alpha_int = c(0.1, 10),
    k_margin = 1.96,
    lower.tail = TRUE,
    X = NULL,
    y = 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,
    compute.PPP = FALSE,
    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 algorithm while generating into the margin (according to MP*gauss)
N_iter_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].
X Coordinates of already known points
y 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
compute_PPP to simulate a Poisson process at each iteration to estimate the conditional expectation and the SUR criteria based on the conditional variance: \(h\) (average probability of misclassification at level failure) and \(I\) (integral of \(h\) over the whole interval [failure, infinity])
MetaIS

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 surrogate
 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 normalizing 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 com-
bined 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 instrumental 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.
X
   The final learning database, ie. all points where lsf has been calculated.
y
   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 ‘pre-
dict’ 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.
h
   the sequence of the estimated relative SUR criteria.
I
   the sequence of the estimated integrated SUR criteria.
Note

Problem is supposed to be defined in the standard space. If not, use \texttt{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 \texttt{as.matrix} transformation of a vector.

Algorithm calls \texttt{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 \texttt{MonteCarlo}.

Author(s)

Clement WALTER <clementwalter@icloud.com>

References

- V. Dubourg:
  Meta-models adaptatifs pour l’analyse de fiabilité et l’optimisation sous contrainte fiabiliste

- 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

\texttt{SubsetSimulation MonteCarlo km} (in package \texttt{DiceKriging})

Examples

```r
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)
```
The modified Metropolis-Hastings algorithm

### Description

The function implements the specific modified Metropolis-Hastings algorithm as described first by Au & Beck and including another scaling parameter for an extended search in initial steps of the SMART algorithm.

### Usage

```r
MetropolisHastings(
  x0,
  eval_x0 = -1,
)```
chain_length,
modified = TRUE,
sigma = 0.3,
proposal = "Uniform",
lambda = 1,
limit_fun = function(x) { -1 },
burnin = 20,
thinning = 4
)

Arguments

- **x0**: the starting point of the Markov chain
- **eval_x0**: the value of the limit-state function on x0
- **chain_length**: the length of the Markov chain. At the end the chain will be chain_length + 1 long
- **modified**: a boolean to use either the original Metropolis-Hastings transition kernel or the coordinate-wise one
- **sigma**: a radius parameter for the Gaussian or Uniform proposal
- **proposal**: either "Uniform" for a Uniform random variable in an interval [-sigma, sigma] or "Gaussian" for a centred Gaussian random variable with standard deviation sigma
- **lambda**: the coefficient to increase the likelihood ratio
- **limit_fun**: the limite-state function delimiting the domain to sample in
- **burnin**: a burnin parameter, ie a number of initial discards samples
- **thinning**: a thinning parameter, ie that one sample over thinning samples is kept along the chain

Details

The modified Metropolis-Hastings algorithm is supposed to be used in the Gaussian standard space. Instead of using a proposed point for the multidimensional Gaussian random variable, it applies a Metropolis step to each coordinate. Then it generates the multivariate candidate by checking if it lies in the right domain.

This version proposed by Bourinet et al. includes an scaling parameter lambda. This parameter is multiplied with the likelihood ratio in order to increase the chance of accepting the candidate. While it biases the output distribution of the Markov chain, the authors of SMART suggest its use (lambda > 1) for the exploration phase. Note such a value disable to possiblity to use the output population for Monte Carlo estimation.

Value

A list containing the following entries:

- **points**: the generated Markov chain
- **eval**: the value of the limit-state function on the generated samples
**Description**

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

**Usage**

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

**Arguments**

- `X`: a matrix containing the data sets
- `Y`: a vector containing -1 or +1 that represents the class of each element 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 line contains the parameters of a hyperplane.

**Author(s)**

Vincent Moutoussamy

**References**

- R.T. Rockafellar:  
  *Convex analysis*  

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

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

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)
Note

The NATAF distribution is reviewed from the (normal) copula viewpoint as a particular and con-
venient 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 \( \rho_{ij} \).

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
- V. Dubourg, Meta-modeles adaptatifs pour l’analyse de fiabilite et l’optimisation sous con-

See Also

UtoX

Examples

```r
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

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

Arguments

- `f` a failure function
- `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.
MonotonicQuantileEstimation

UM An upper bounds of the probability obtained from the designing 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


Examples

```r
## 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 = foreach::getDoParWorkers(),
  q = 0,
  lower.tail = TRUE,
  precision = 0.05,
  plot = FALSE,
  output_dir = NULL,
  save.X = TRUE,
  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 evaluated 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".
save.X to save all the samples generated as a matrix. Can be set to FALSE to reduce output size.
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 batches of points sequentially 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 probability.
- `ecdf` the empirical cdf got with the generated samples.
- `cov` the coefficient of variation of the Monte Carlo estimator.
- `Ncall` the total number of calls to the `lsf`, i.e., 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 <clementwalter@icloud.com>

References

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

See Also

- `SubsetSimulation`
- `foreach`

Examples

```R
# 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 length 2
}
lsf_correct = function(x){
  apply(x, 2, lsf_wrong)
}
lsf = function(x, b=5, kappa=0.5, e=0.1) {
  b - x[2] - kappa*(x[1]-e)^2 # work only with a vector of length 2
}
\begin{verbatim}
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)
\end{verbatim}

---

**MP**

*Moving Particles*

**Description**

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

**Usage**

\[ \text{MP(} \]
dimension, lsf, 
N = 100, 
N.batch = foreach::getDoParWorkers(), 
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. Note that N/N.batch has to be an integer.
p a given probability to estimate the corresponding quantile (as in qxxxx func-
tions).
q a given quantile to estimate the corresponding probability (as in pxxxx func-
tions).
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

\texttt{MP} is a wrap up of \texttt{IRW} for probability and quantile estimation. By construction, the several calls to \texttt{IRW} are parallel (\texttt{foreach}) and so is the algorithm. Especially, with \texttt{N.batch}=1, this is the Last Particle Algorithm, which is a specific version of \texttt{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 \( N.b \). batch.

Value

An object of class \texttt{list} containing the outputs described below:

\begin{itemize}
  \item \texttt{p} the estimated probability or the reference for the quantile estimate.
  \item \texttt{q} the estimated quantile or the reference for the probability estimate.
  \item \texttt{cv} the coefficient of variation of the probability estimator.
  \item \texttt{ecdf} the empirical cdf.
  \item \texttt{L} the states of the random walk.
  \item \texttt{L_max} the farthest state reached by the random process. Validity range for the \texttt{ecdf} is then \((-\text{Inf}, \text{L_max}]\) or \([\text{L_max}, \text{Inf})\).
  \item \texttt{times} the \textit{times} of the random process.
  \item \texttt{Ncall} the total number of calls to the \texttt{lsf}.
  \item \texttt{X} the \( N \) particles in their final state.
  \item \texttt{y} the value of the \texttt{lsf}(X).
  \item \texttt{moves} a vector containing the number of moves for each batch.
  \item \texttt{p_int} a 95\% confidence intervall on the probability estimate.
  \item \texttt{cov} the coefficient of variation of the estimator
  \item \texttt{q_int} a 95\% confidence intervall on the quantile estimate.
  \item \texttt{chi2} the output of the chisq.test function.
\end{itemize}

Note

The \texttt{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 \texttt{Niter_1fold} = \(-N/N.b\). batch*\log(p) gives 100\% chances to produces a quantile estimator.
Author(s)

Clement WALTER <clementwalter@icloud.com>

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*

See Also

SubsetSimulation MonteCarlo IRW

Examples

```r
## 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(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())
```

MRM

## MRM method

### Description
Estimate a failure probability by MRM method.

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

### Arguments
- **f**: a failure function
- **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 adaption 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


Examples

```r
## 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)
```
## Class of Ordinary Kriging

**Description**

An implementation of Ordinary Kriging based upon a km-class object that should be faster than usual predict method.

**Usage**

`ok(model, beta = NULL)`

**Arguments**

- `model`: a kriging model object from DiceKriging::km-class
- `beta`: the trend of the model

**Details**

The Ordinary Kriging is a special case of kriging where the trend is supposed to be an unknown constant. Consequently some linear algebra operations can be reduced by knowing that the vector of parameter beta is indeed a real.

The ok class defines three functions: `xi` the kriging predictor, `updateSd` and `updateSdfast` two methods for updating the kriging variance when some points are virtually added to the model. These two last functions differ in their implementation: the first one allows for the user to specify which are the predicted points and which are the added points. The second one outputs a matrix where the kriging variances of all the points is updated when each one is iteratively added to the Design of Experiments.

The faster between looping `updateSd` and using `updateSdfast` is indeed problem dependent (depending on parallel computer, size of the data, etc.) and should be benchmark by the user.

**Value**

An object of S3 class 'ok' containing

- `Kinv`: the inverse of the covariance matrix of the data
- `beta`: the estimated coefficient of the trend
- `y_centred`: the data centred according to the estimated trend
- `sigma_beta`: the standard deviation of the estimation of beta
- `xi`: the kriging predictor
- `updateSd`: a function to calculate the updated kriging variance when `Xnew` points are added to the Design of Experiments
- `updateSdfast`: a function to calculate the update kriging variance when the SUR criterion is minimised over a population which is also the one used to estimate it.
oscillator_d6

A limit-state-function defined with a non-linear oscillator in dimension 6.

Description

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

Usage

oscillator_d6

Format

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

References

Echard, B and Gayton, N and Lemaire, M and Relun, N:
A combined Importance Sampling and Kriging reliability method for small failure probabilities with time-demanding numerical models
plotLSVM

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 represents 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
See Also

LSVM modelLSVM

Examples

```r
# 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)
```

Description

precomputeUpdateData

Usage

```r
precomputeUpdateData(model, integration.points)
```

Arguments

- model: a object from `km`
- integration.points: the points onto which the updated variance will be computed
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


Examples

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

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
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,
  lower.tail = TRUE,
  ...,  
  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
lower.tail as for pxxxx functions, TRUE for estimating P(lsf(X) < failure), FALSE for P(lsf(X) > failure)
... 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 = \((N_n + \text{number of calls to refine the metamodel}) \times \text{(number of subsets)}\):

- \(N_n\) calls to find the next threshold value: the bigger \(N_n\), the more accurate the 'decreasing speed' specified by the \(\alpha_{\text{quantile}}\) 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.
- \(\text{cov}\) The coefficient of variation of the Monte-Carlo probability estimate.
- \(\text{Ncall}\) The total number of calls to the \(\text{lsf}\).
- \(x\) The final learning database, ie. all points where \(\text{lsf}\) has been calculated.
- \(y\) The value of the \(\text{lsf}\) on the learning database.
- \(\text{meta\_model}\) 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 \(\text{lsf}(X)\) (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 <clementwalter@icloud.com>

**References**

SMART Support-vector Margin Algorithm for Reliability Estimation

- S.-K. Au, J. L. Beck:
  *Estimation of small failure probabilities in high dimensions by Subset Simulation*

- A. Der Kiureghian, T. Dakessian:
  *Multiple design points in first and second-order reliability*

- P.-H. Waarts:
  *Structural reliability using finite element methods: an appraisal of DARS: Directional Adaptive Response Surface Sampling*

See Also

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

Examples

```r
## 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)
```
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 = 2e+05,
    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,
    X = NULL,
    y = NULL,
    failure = 0,
    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 the dimension of the input space
lsf the limit-state function
N1 Number of samples for the (L)ocalisation step
N2 Number of samples for the (S)tabilisation step
SMART

N3
Nu
lambda_1
lambda_2
lambda_3
tune_cost
tune_gamma
clusterInMargin
alpha_margin
k_1
k_2
k_3
X
y
failure
limit_fun_MH
sampling_strategy
seeds
seeds_eval
burnin
thinning
plot
limited_plot
add
output_dir
z_MH
z_lsf
verbose

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 calculates 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 surrogate of the limit state function.

Concerning the refinement strategy, it distinguishes 3 stages, known as Localisation, Stabilisation 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.
- **Ncall**: The total number of calls to the `limit_state_function`.
- **x**: The final learning database, i.e. all points where `lsf` has been calculated.
- **y**: 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 <clementwalter@icloud.com>

References


See Also

*SubsetSimulation MonteCarlo svm* (in package `e1071`) S2MART
**Description**

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

**Usage**

```r
SubsetSimulation(
  dimension,
  lsf,
  p_0 = 0.1,
  N = 10000,
  q = 0,
  lower.tail = TRUE,
  K,
  thinning = 20,
  save.all = FALSE,
  plot = FALSE,
  plot.level = 5,
  plot.lsf = TRUE,
  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, i.e., 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 or rejected. Default kernel is the one defined $K(X) = (X + \sigma^2 W)/\sqrt{1 + \sigma^2}$ with $W \sim \mathcal{N}(0, 1)$.
- `thinning`: a thinning parameter for 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.
SubsetSimulation

plot to plot the generated samples.

plot.level maximum number of expected levels for color consistency. If number of levels exceeds this value, the color scale will change according to ggplot2 default policy.

plot.lsf a boolean indicating if the lsf should be added to the plot. This requires the evaluation of the lsf over a grid and consequently should be used only for illustration purposes.

output_dir to save the plot into a pdf file. This variable will be pasted 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 conditionally 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.

\( cv \) 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 <clementwalter@icloud.com>

References

- S.-K. Au, J. L. Beck:  
  *Estimation of small failure probabilities in high dimensions by Subset Simulation*

- 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*

See Also

IRW MP MonteCarlo

Examples

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

```r
## 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]]$p # get a reference value for p
p_0 <- 0.1 # the default value recommended by Au & Beck
N_mp <- 100
# to get approximately the same number of calls to the lsf
N_ss <- ceiling(N_mp*log(p)/log(p_0))
comp <- replicate(50, {
  ss <- SubsetSimulation(2, kiureghian, N = N_ss)
  mp <- MP(2, kiureghian, N = N_mp, q = 0)
  comp <- c(ss$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 represents 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

See Also

LSVM modelLSVM
Examples

```r
# 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
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*

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

**See Also**

- `LSVM modelLSVM`

**Examples**

```r
# 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)
```
**updateSd**

**Description**

Update kriging variance when adding new points to the DoE

**Usage**

```
updateSd(
  X.new,
  integration.points.oldsd,
  model,
  precalc.data,
  integration.points
)
```

**Arguments**

- `X.new` : the d x N matrix containing the points added to the model for the update of the kriging variance.
- `integration.points.oldsd` : a vector containing the standard deviation of the points to be added to the meta-model learning database.
- `model` : the current kriging model (a km object).
- `precalc.data` : precomputed data from KrigInv::precomputeUpdateData.
- `integration.points` : points where the updated sd is to be calculated.

**Value**

a vector containing the kriging sd at points integration.points

**updateSd.old**

**Description**

UpdateSd.old

**Usage**

```
updateSd.old(X.new, newdata.oldsd, model, precalc.data, integration.points)
```
**Arguments**

- **X.new**: the d x N matrix containing the points added to the model for the update of the kriging variance.
- **newdata.oldsd**: a vector containing the standard deviation of the points to be added to the meta-model learning database.
- **model**: the current kriging model (a km object).
- **precalc.data**: precomputed data from KrigInv::precomputeUpdateData.
- **integration.points**: points where the updated sd is to be calculated.

---

**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 $\rho(X_i, X_j) = \rho_{ij}$ and the marginal distributions of $X_i$. 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
  
  \[ \text{dist} X \leftarrow \text{list} (\text{type} = "\text{Norm}", \text{MEAN} = 0.0, \text{STD} = 1.0, \text{NAME} = "X1") \]

- **LOGNORMAL**: distribution, defined by its internal parameters P1=meanlog and P2=sdlog (plnorm)
  
  \[ \text{dist} X \leftarrow \text{list} (\text{type} = "\text{Lnorm}", P1 = 10.0, P2 = 2.0, \text{NAME} = "X2") \]
• UNIFORM: distribution, defined by its internal parameters P1=min and P2=max (punif)
  \[
  \text{distX} \leftarrow \text{list}(\text{type} = "\text{Unif}", P1 = 2.0, P2 = 6.0, \text{NAME} = "X3")
  \]
• GUMBEL: distribution, defined by its internal parameters P1 and P2
  \[
  \text{distX} \leftarrow \text{list}(\text{type} = "\text{Gumbel}", P1 = 6.0, P2 = 2.0, \text{NAME} = "X4")
  \]
• WEIBULL: distribution, defined by its internal parameters P1=shape and P2=scale (pweibull)
  \[
  \text{distX} \leftarrow \text{list}(\text{type} = "\text{Weibull}", P1 = \text{NULL}, P2 = \text{NULL}, \text{NAME} = "X5")
  \]
• GAMMA: distribution, defined by its internal parameters P1=shape and P2=scale (pgamma)
  \[
  \text{distX} \leftarrow \text{list}(\text{type} = "\text{Gamma}", P1 = 6.0, P2 = 6.0, \text{NAME} = "X6")
  \]
• BETA: distribution, defined by its internal parameters P1=shape1 and P2=shape2 (pbeta)
  \[
  \text{distX} \leftarrow \text{list}(\text{type} = "\text{Beta}", P1 = 6.0, P2 = 6.0, \text{NAME} = "X7")
  \]

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 con-

See Also
  ModifCorrMatrix, ComputeDistributionParameter

Examples

Dim = 2

\[
\text{distX1} \leftarrow \text{list}(\text{type} = "\text{Norm}", \text{MEAN}=0.0, \text{STD}=1.0, P1=\text{NULL}, P2=\text{NULL}, \text{NAME} = "X1")
\]
\[
\text{distX2} \leftarrow \text{list}(\text{type} = "\text{Norm}", \text{MEAN}=0.0, \text{STD}=1.0, P1=\text{NULL}, P2=\text{NULL}, \text{NAME} = "X2")
\]

\[
\text{input.margin} \leftarrow \text{list(\text{distX1, distX2})}
\]
\[
\text{input.Rho} \leftarrow \text{matrix(} \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix}, \text{nrow=Dim})
\]
\[
\text{input.R0} \leftarrow \text{ModifCorrMatrix(input.Rho)}
\]
\[
\text{L0} \leftarrow \text{t(chol(input.R0))}
\]
waarts

A limit-state-function defined by Waarts

Description

The limit-state function is defined by:

\[ b_1 = 3 + (u_1 - u_2)^2 / 10 - \text{sign}(u_1 + u_2) \ast (u_1 + u_2) / \sqrt{2} \]

\[ b_2 = \text{sign}(u_2 - u_1) \ast (u_1 - u_2) + 7 / \sqrt{2} \]

\[ f(u) = \min(b_1, b_2) \]

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  
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


Examples

```r
N <- WilksFormula(0.95, 0.95, order=1)
print(N)
```
**XtoU**  
*From X to standard space*

**Description**

XtoU lets transform datapoint in the original space X to the standard Gaussian space U with isoprobabilistic transformation.

**Usage**

```
XtoU(X, input.margin, L0)
```

**Arguments**

- **X**  
The matrix d x n of the input points
- **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`)

**Author(s)**

Clement WALTER <clement.walter@cea.fr>

**See Also**

`UtoX`
Index

* datasets
  cantilever, 12
  kiureghian, 23
  oscillator_d6, 45
  rackwitz, 49
  twodof, 60
  waarts, 66
* package
  mistral-package, 2

AKMCS, 4
BMP, 8
cantilever, 12
ComputeDistributionParameter, 13, 65
estimateSUR, 13
foreach, 37
FORM, 15
FORMv0, 17
generateK, 18
IRW, 12, 20, 39–41, 58
kiureghian, 23
km, 8, 28, 52
LPA (MP), 38
LSVM, 23, 32, 47, 59, 62
MetaIS, 8, 25
MetropolisHastings, 29
mistral (mistral-package), 2
mistral-package, 2
modellSVM, 24, 31, 47, 59, 62
ModifCorrMatrix, 32, 64, 65, 68
MonotonicQuantileEstimation, 33
MonteCarlo, 7, 8, 12, 22, 28, 36, 41, 51, 52, 55, 57, 58
MP, 12, 22, 38, 58
MRM, 42
NestedSampling (IRW), 20
ok, 44
oscillator_d6, 45
oscillator_d8 (twodof), 60
pbeta, 65
pgamma, 65
plnorm, 64
plotLSVM, 46
precomputeUpdateData, 47
punif, 65
pweibull, 65
quantileWilks, 48
rackwitz, 49
S2MART, 50, 55
SMART, 52, 52
ss (SubsetSimulation), 56
subset (SubsetSimulation), 56
SubsetSimulation, 8, 12, 28, 37, 40, 41, 52, 55, 56
svm, 52, 55
testConvexity, 59
TPA (IRW), 20
twodof, 60
updateLSVM, 61
updateSd, 63
updateSd.old, 63
UtoX, 7, 22, 28, 33, 37, 51, 55, 57, 64
waarts, 66
WilksFormula, 67
XtoU, 68