Package: TesiproV (via r-universe)

October 30, 2026

Type Package

Title Reliability Analysis Methods for Structural Engineering

Version 1.0.1

Date 2026-10-29

Maintainer Til Lux <til.lux@tu-dortmund.de>

Description Calculate the failure probability of civil engineering problems with Level I up to Level III Methods. References: Spaethe (1991, ISBN:3-211-82348-4) ``Die Sicherheit tragender Baukonstruktionen", Au and Beck (2001) ``Estimation of small failure probabilities in high dimensions by subset simulation." <doi:10.1016/S0266-8920(01)00019-4>, Breitung (1989) ``Asymptotic approximations for probability integrals." <doi:10.1016/0266-8920(89)90024-6>.

URL https://mvb.ab.tu-dortmund.de/

Encoding UTF-8

License MIT + file LICENSE

Imports stats, future, future.apply, digest, numDeriv, nloptr, methods, pracma, parallel, graphics, utils, EnvStats, evd

Suggests testthat (>= 3.0.0), withr, knitr, rmarkdown, microbenchmark

VignetteBuilder knitr

Config/roxygen2/version 8.1.1

Config/testthat/edition 3

NeedsCompilation no

Author Konstantin Nille-Hauf [aut], Tânia Feiri [aut], Marcus Ricker [aut], Jan Phillip Schulze-Ardey [ctb], Til Lux [aut, cre]

Config/pak/sysreqs cmake

Repository https://smtiluxx.r-universe.dev

Date/Publication 2026-10-30 19:32:34 UTC

RemoteUrl https://github.com/cran/TesiproV

RemoteRef HEAD

RemoteSha 026783430102463fc0e61aa812df252e7907964c

Contents

TesiproV-package . . . 2
FORM . . . 3
Internal helper functions for MC_IS() . . . 5
LogStudentT . . . 5
MC_CRUDE . . . 7
MC_IS . . . 9
MC_SubSam . . . 14
MVFOSM . . . 16
PARAM_BASEVAR-class . . . 18
PARAM_DETVAR-class . . . 19
PARAM_LSF-class . . . 19
PROB_BASEVAR-class . . . 20
PROB_DETVAR-class . . . 21
PROB_MACHINE-class . . . 22
PROB_QUALCON-class . . . 23
SORM . . . 26
StudentT . . . 27
SYS_LSF-class . . . 29
SYS_PARAM-class . . . 30
SYS_PROB-class . . . 32

Index **35**

`TesiproV-package` *TesiproV: A package for the calculation of reliability and failure probability in civil engineering*

Description

The Package provides three main types of objects:

1. Objects for modeling base variables
2. Objects for modeling limit state functions and systems of them
3. Objects for modeling solving algorithms

Details

By creating and combining those objects, one is able to model quite complex problems in terms of structural reliablity calculation. For normally distributed variables there might be an workflow to calculate correlated problems (but no systems then). There is also implemented a new distribution (logStudentT, often used for conrete compression strength) to show how one can implement your very own or maybe combined multi modal distribution and use it with TesiproV.

Objects for base variables

`PROB_BASEVAR`, `PROB_DETVAR`, `PARAM_BASEVAR`, `PARAM_DETVAR`

Limit state functions

SYS_LSF, PROB_SYS, PARAM_SYS

Solving algorithms

PROB_MACHINE

Author(s)

(C) 2021-2026 K. Nille-Hauf, T. Feiri, M. Ricker, T. Lux – Hochschule Biberach (until 2022), TU Dortmund University - Chair of Structural Concrete (since 2023)

See Also

Useful links:

- https://mvb.ab.tu-dortmund.de/

FORM *First-Order Reliability Method (FORM)*

Description

First-Order Reliability Method (FORM) for the approximation of failure probabilities in structural reliability analysis.

The FORM estimates the probability of failure by transforming the basic random variables into standard normal space and approximating the limit-state function by a first-order (linear) Taylor expansion at the design point.

The reliability index is defined as the minimum distance from the origin to the limit-state surface in standard normal space. The probability of failure is then approximated using the standard normal cumulative distribution function.

Failure is defined as $g(x) < 0$. The reliability index is positive if the origin of the standard normal space lies in the safe domain and negative otherwise. The probability of failure is $p_f = \Phi(-\beta)$.

Solver selection: With an explicitly given `optim_type` exactly this algorithm is run and nothing else. If it does not converge, a warning is issued and the last iterate is returned with `converged = FALSE`. With `optim_type = "auto"` (default) a robust chain is used: SLSQP (**nloptr**), and iHLRF as a fallback. A warning is only issued if both fail.

Usage

```
FORM(
  lsf = NULL,
  lDistr,
  n_optim = 100L,
  loctol = 1e-05,
  optim_type = "auto",
```

```
  lsf_list = NULL,
  debug.level = 0
)
```

Arguments

lsf	Objective function representing the limit-state, e.g. `function(R,E){R-E}`. Supplied automatically by a SYS_ object - do not provide manually.
lDistr	List of distribution objects created by TesiproV. Supplied automatically by a SYS_ object - do not provide manually.
n_optim	Maximum number of iterations. For `"slsqp"` it is the maximum number of function evaluations of the optimizer. In `"auto"` SLSQP may use `max(200, 10 * n_optim)` evaluations.
loctol	Relative tolerance. Converged means $\|g(u)\| \leq loctol \cdot g_{scale}$ and the distance of u from the line through the origin along the gradient is $\leq loctol \cdot \max(1, \|u\|)$. SLSQP results are accepted with at most `max(loctol, 1e-5)`.
optim_type	Solver: • `"auto"` - (default) SLSQP, then iHLRF if needed • `"ihlrf"` - improved HLRF with merit-function line-search (Zhang & Der Kiureghian 1997), no fallback • `"rackfies"` - Rackwitz-Fiessler iteration (classic HLRF), no fallback • `"slsqp"` - SQP with BFGS (Kraft 1988, **nloptr**), no fallback • `"auglag"` - Augmented Lagrangian (nloptr), random starts
lsf_list	Reserved for a future system-FORM (not active).
debug.level	Verbosity level: 0 = silent, 1 = basic info, 2 = detailed output.

Value

A list (content depends on optim_type): * `beta` - Hasofer-Lind reliability index * `pf` - probability of failure $\Phi(-\beta)$ * `x_points` - design point in physical space * `u_points` - design point in standard normal space * `dy` - gradient in standard normal space at the design point * `alpha` - direction cosines * `n_sim` - number of (i)HLRF iterations performed * `converged` - TRUE if the convergence criteria are met * `solver` - algorithm that delivered the result ("HLRF", "iHLRF", "SLSQP") * `fallback` - TRUE if `optim_type = "auto"` had to switch from SLSQP to iHLRF, otherwise FALSE * `conv_info` - named vector of final convergence metrics

Author(s)

(C) 2021-2026 K. Nille-Hauf, J.P. Schulze-Ardey, T. Feiri, M. Ricker, T. Lux – Hochschule Biberach (until 2022), TU Dortmund University - Chair of Structural Concrete (since 2023)

References

Hasofer, A. M., & Lind, N. C. (1974). An exact and invariant first-order reliability format. *Journal of the Engineering Mechanics Division, ASCE*, 100(1), 111-121.

Rackwitz, R., & Fieszler, B. (1978). Structural reliability under combined random load sequences. *Computers & Structures*, 9(5), 489-494.

Zhang, Y., & Der Kiureghian, A. (1997). Finite element reliability methods for inelastic structures. Report UCB/SEMM-97/05, University of California, Berkeley.

Kraft, D. (1988). A software package for sequential quadratic programming. Technical Report DFVLR-FB 88-28, Institut fuer Dynamik der Flugsysteme, Oberpfaffenhofen.

`Internal helper functions for MC_IS()` *Internal helper functions for Monte Carlo Importance Sampling (MC_IS)*

Description

This file contains internal helper functions used by the main function [‘MC_IS()‘] to perform Monte Carlo simulations with importance sampling.

The helpers defined here support: * random number generation and cluster setup, * creation of recorder objects for iterative logging, * single limit state simulation routine (‘MC_IS_single()‘), * conversion of recorded data to data frames after completion.

Additional helper sets will be provided for system calculations (serial and parallel systems) in separate files.

Details

These functions are not intended to be called directly by users. They are exported internally to allow modular access within the package.

Author(s)

(C) 2021-2026 K. Nille-Hauf, T. Feiri, M. Ricker, T. Lux – Hochschule Biberach (until 2022), TU Dortmund University - Chair of Structural Concrete (since 2023)

See Also

[TesiproV::MC_IS()] for the public interface that calls these helpers.

`LogStudentT` *Log-Student t Distribution*

Description

Density, distribution function, quantile function and random generation for the log-Student t distribution.

Usage

```
dlt(x, hyper.param)

plt(q, hyper.param)

qlt(p, hyper.param)

rlt(n_vals, hyper.param)
```

Arguments

x, q	Vector of quantiles.
hyper.param	Numeric vector (m, s, n, ν).
p	Vector of probabilities.
n_vals	Number of random values to generate.

Details

The cumulative distribution function is given by

$$F_X(x) = F_{t_\nu}\left(\frac{\ln X - m}{s}\sqrt{\frac{n}{n+1}}\right),$$

where $F_{t_\nu}(\cdot)$ denotes the cumulative distribution function of the Student t distribution with ν degrees of freedom.

The log-Student t distribution arises as the marginal distribution of a log-normal distributed variable with conjugate log-normal-gamma prior uncertainty in m and s.

The hyperparameter vector `hyper.param` is defined as

$$(m, s, n, \nu)$$

where

- m is the mean value of an equivalent sample of size n,
- s is the empirical standard deviation of an equivalent sample of size $\nu + 1$,
- ν denotes the degrees of freedom of the Student t distribution.

For details see: https://www.jcss-lc.org/publications/jcsspmc/concrete.pdf

Value

- `dlt()` returns the density.
- `plt()` returns the distribution function.
- `qlt()` returns the quantile function.
- `rlt()` generates random deviates.

Author(s)

(C) 2021-2026 K. Nille-Hauf, T. Feiri, M. Ricker, T. Lux – Hochschule Biberach (until 2022), TU Dortmund University - Chair of Structural Concrete (since 2023)

Examples

```
hp <- c(3.4, 0.14, 3, 10)
dlt(0.5, hp)
```

MC_CRUDE *Crude Monte Carlo Simulation*

Description

Crude Monte Carlo simulation (MC_CRUDE) method for the estimation of failure probabilities in structural reliability analysis.

The failure probability is estimated as

$$\hat{p}_f = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{\{g(\mathbf{x}_i)<0\}}.$$

Failure is defined as $g(x) < 0$. Samples with $g(x) = 0$ are counted as safe.

Although conceptually simple and unbiased, the method becomes computationally inefficient for very small failure probabilities, as a large number of samples is required to obtain stable estimates.

Usage

```
MC_CRUDE(
  lsf,
  lDistr,
  cov_user = 0.05,
  n_batch = 10000,
  n_max = 1e+07,
  use_threads = NULL,
  backend = NULL,
  dataRecord = TRUE,
  debug.level = 0,
  seed = NULL
)
```

Arguments

lsf	Limit-state function. Must follow the LSF interface specification described in the "Limit-State Function (LSF) Interface" section below.
lDistr	List of distribution objects as returned by PROB_BASEVAR$getlDistr().
cov_user	Target coefficient of variation to be achieved.
n_batch	Number of samples per batch.
n_max	Maximum number of Monte Carlo samples.
use_threads	Number of workers for backend = "parallel" (default 1). Without effect for backend = "future".
backend	Parallel backend, "future" or "parallel". If NULL (default), "parallel" with one worker is used, i.e. the calculation runs sequentially on every system.
dataRecord	Logical; if TRUE, intermediate results are recorded.
debug.level	Integer controlling verbosity (0 = silent, 2 = detailed output).
seed	Optional integer value for reproducible random numbers.

Details

Stopping criterion

The simulation stops when the coefficient of variation falls below cov_user or when n_max samples are reached. The last batch is reduced so that the total does not exceed n_max.

Parallel execution

The samples of each batch are generated in the master process and split into chunks, which are evaluated in parallel. The workers do not generate random numbers.

- backend = "future": the chunks are evaluated with future.apply::future_lapply() under the active **future** plan. The function never changes the plan. The number of workers is determined by the plan, use_threads has no effect and issues a warning if supplied.
- backend = "parallel": on Unix-like systems parallel::mclapply() with use_threads cores is used. On Windows a PSOCK cluster with use_threads workers is created for the call and stopped afterwards. With use_threads = 1 lapply() is used.

Reproducibility

With a fixed seed the random numbers are generated from a L'Ecuyer-CMRG stream in the master process. The result therefore does not depend on the number of workers. The function sets RNGkind("L'Ecuyer-CMRG") and changes .Random.seed in the global environment.

Value

MC_CRUDE returns a list with the following elements:

- method: Character string "MC_CRUDE".
- beta: Estimated reliability index $\beta = -\Phi^{-1}(p_f)$.
- pf: Estimated probability of failure.
- var: Estimated variance of the Monte Carlo estimator.

- cov_mc: Estimated coefficient of variation of pf.
- cov_user: Target coefficient of variation.
- n_mc: Total number of Monte Carlo samples generated.
- n_max: Maximum allowed number of samples.
- n_batch: Batch size.
- n_threads: Number of workers used.
- runtime: CPU time returned by proc.time().
- data: Data frame with intermediate results (only if dataRecord = TRUE).

Limit-State Function (LSF) Interface

The argument lsf must define a valid limit-state function using one of the following supported signatures:

function(x) A single numeric vector x containing all basic variables in the order defined in lDistr. Recommended.

function(Z, Fy, M, ...) Explicitly named scalar arguments corresponding to the variable names defined in the probabilistic model.

The use of function(...) as the sole argument is not supported.

Windows users

On Windows, backend = "future" with future::plan(future::multisession, workers = ...) is recommended. backend = "parallel" creates a new PSOCK cluster for every call.

Author(s)

(C) 2021-2026 K. Nille-Hauf, T. Feiri, M. Ricker, T. Lux – Hochschule Biberach (until 2022), TU Dortmund University - Chair of Structural Concrete (since 2023)

References

Spaethe, G. (1991). *Die Sicherheit tragender Baukonstruktionen*. Springer.

MC_IS *Monte Carlo Simulation with Importance Sampling*

Description

Performs Monte Carlo simulation with importance sampling (IS) to estimate the failure probability of a single limit-state function or of a system of limit-state functions.

Failure is defined as $g(x) < 0$. Samples with $g(x) = 0$ are counted as safe.

The samples are drawn from a sampling density h in standard normal space that is centred at FORM design points. The sampling is corrected by the likelihood ratio $w(u) = \varphi(u)/h(u)$, where φ is the standard normal density.

Usage

```
MC_IS(
  lsf,
  lDistr,
  cov_user = 0.05,
  n_batch = 5000,
  n_max = 1e+06,
  use_threads = NULL,
  backend = NULL,
  sys_type = "serial",
  dataRecord = TRUE,
  beta_l = 100,
  densityType = "norm",
  dps = NULL,
  debug.level = 0,
  seed = NULL,
  adaptive_alpha = FALSE,
  alpha_update_rate = 0.1,
  adaptive_batch = FALSE,
  batch_control = list(),
  min_adapt_samples = 10000,
  alpha_min = 0.02,
  stability_mode = c("robust", "fast")
)
```

Arguments

lsf	A limit-state function or a list of limit-state functions for a system. Must follow the LSF interface described below.
lDistr	List of marginal distribution objects (funlists with $d, $p, $q, $r). For a system, a list with one such list per limit-state function.
cov_user	Target coefficient of variation of the estimator.
n_batch	Number of samples per batch.
n_max	Maximum total number of samples (see stopping criterion).
use_threads	Number of chunks per batch. With backend = "parallel" also the number of cores or workers.
backend	Parallel backend, "future" or "parallel". If NULL (default), "parallel" with one worker is used, i.e. the calculation runs sequentially on every system.
sys_type	"serial" or "parallel". Only used for systems.
dataRecord	Logical. If TRUE, the results of each batch are returned in data. Single limit-state function only.
beta_l	Currently without effect.
densityType	Currently without effect. A normal sampling density in standard normal space is always used.

dps	Optional design points in physical space. If supplied, FORM is skipped for the components. Single limit-state function: a numeric vector with one value per variable (or one value for all). System: a list with one such vector per limit-state function. A numeric vector with one value per limit-state function is accepted if each function has exactly one variable.
debug.level	Verbosity level: 0 = silent, 1 = summary, 2 = detailed output.
seed	Optional integer seed for reproducible results.
adaptive_alpha	Logical. Adaptive mixture weights are not implemented. Without effect for a single limit-state function. For serial multimodal systems the calculation stops with an error once min_adapt_samples is exceeded.
alpha_update_rate	Reserved for adaptive_alpha, currently without effect.
adaptive_batch	Logical. Systems only. If TRUE, the batch size is adapted from the current effective number of failure samples, so that cov_user is expected to be reached after K_future further batches.
batch_control	List for adaptive_batch with the elements n_min and n_max (bounds of the batch size) and K_future (number of remaining batches). Systems only.
min_adapt_samples	Reserved for adaptive_alpha (see there).
alpha_min	Reserved for adaptive_alpha, currently without effect.
stability_mode	"robust" or "fast". Without influence on the result, all sums are accumulated in the log domain. For serial multimodal systems "fast" additionally returns the effective sample size per mixture component (ESS_modes).

Details

Estimator

The failure probability and the variance of the estimator are

$$\hat{P}_f = \frac{1}{N}\sum_{i=1}^{N} I_i w_i, \qquad \widehat{\mathrm{Var}}(\hat{P}_f) = \frac{1}{N}\left(\frac{1}{N}\sum_{i=1}^{N} I_i w_i^2 - \hat{P}_f^2\right),$$

where $I_i = 1$ if sample i lies in the failure domain and $I_i = 0$ otherwise. The coefficient of variation is $\sqrt{\widehat{\mathrm{Var}}(\hat{P}_f)}/\hat{P}_f$.

Single limit-state function

The failure domain is $\{g(x) < 0\}$. The sampling density is a multivariate normal density centred at the FORM design point u^* with the covariance matrix

$$\Sigma = I + (\beta^2 - 1)\,\alpha\alpha^T, \qquad \alpha = u^*/\|u^*\|,$$

where β is the FORM reliability index, or $\|u^*\|$ if it is not available. The variance is β^2 in the direction of α and 1 perpendicular to it. FORM is only used to locate the centre. A design point from a FORM run that did not converge is accepted as long as it is finite.

Systems

For limit-state functions $g_1, \ldots, g_m$ the failure domain is $\bigcup_k \{g_k(x) < 0\}$ for a serial system and $\bigcap_k \{g_k(x) < 0\}$ for a parallel system. Basic variables that appear in several limit-state functions are identified by their name and enter the simulation only once.

The sampling density is a mixture of unit-variance normal densities

$$h(u) = \sum_j \lambda_j \, \varphi(u - u_j^*), \qquad \sum_j \lambda_j = 1.$$

- Serial systems: the centres u_j^* are the FORM design points of the components. The weights are proportional to $\Phi(-\beta_j)$ if all FORM results are finite, otherwise they are equal. If the largest distance between the design points is smaller than 1.5 and the smallest cosine between their gradient directions is larger than 0.85, a single normal density at the mean of the design points is used instead of the mixture.
- Parallel systems: FORM is run for the joint failure domain $\max_k g_k(x) < 0$. If its design point is available, it is a centre with weight 0.7 and the remaining weight is divided equally among the component design points. Otherwise the component design points are used with equal weights.

Stopping criterion

For a single limit-state function the simulation stops when the coefficient of variation falls below `cov_user` or when `n_max` samples are reached. The last batch is reduced so that the total does not exceed `n_max`.

For systems the simulation stops when the coefficient of variation falls below `cov_user`, the effective number of failure samples exceeds 30 (serial) or 50 (parallel) and more than `5 * n_batch` samples have been used. It also stops when `n_max` is reached. The batches are not reduced, so `n_mc` can exceed `n_max` by less than one batch. For parallel systems a warning is issued if the effective number of failure samples is below 50 at the end.

Numerical stability

The weighted sums are accumulated in the log domain. The result includes `log_pf`. For extremely small probabilities `pf` may underflow to zero, `beta` is then computed from `log_pf` and can remain finite. This applies to single limit-state functions and to systems.

Automatic detection of vectorized limit-state functions

Before the simulation the limit-state function is called once with a matrix of two rows (the design point in physical space). If it returns a finite numeric vector of length 2, it is evaluated for whole batches at once. Otherwise it is evaluated row by row with `apply()`. Scalar limit-state functions therefore need not be changed.

Parallel execution

The samples of each batch are generated in the master process and split into `use_threads` chunks, which are evaluated in parallel. The workers do not generate random numbers.

- `backend = "future"`: the chunks are evaluated with `future.apply::future_lapply()` under the active **future** plan. The function never changes the plan. If the plan is sequential and `use_threads > 1`, a message is shown once per session and the calculation runs sequentially. The number of workers is determined by the plan. `use_threads` only determines the number of chunks.

- backend = "parallel": on Unix-like systems parallel::mclapply() with use_threads cores is used. On Windows a PSOCK cluster with use_threads workers is created for the call and stopped afterwards. With use_threads = 1 lapply() is used.

Reproducibility

With a fixed seed the random numbers are generated from a L'Ecuyer-CMRG stream in the master process. The sample therefore does not depend on use_threads. Results for different values of use_threads agree up to rounding errors, because the sums are accumulated per chunk. The function sets RNGkind("L'Ecuyer-CMRG") and changes .Random.seed in the global environment.

Value

For a single limit-state function a list with

- method - "MCIS_Single"
- beta - reliability index $-\Phi^{-1}(\hat{P}_f)$
- pf - estimated failure probability
- log_pf - natural logarithm of pf
- log_var - natural logarithm of var
- FORM_beta, FORM_pf - FORM results (NA if dps was supplied)
- design_points - centre of the sampling density in standard normal space
- var - variance of the estimator
- cov_mc - coefficient of variation of the estimator
- cov_user - target coefficient of variation
- ESS - effective sample size $(\sum w_i)^2 / \sum w_i^2$
- ESS_ratio - ESS / n_mc
- n_mc, n_max - used and maximum number of samples
- data - results per batch (if dataRecord = TRUE)
- runtime - runtime as returned by proc.time()

For a system a list with

- method - "MCIS_System"
- sys_type - "serial" or "parallel"
- beta, pf, var, cov_mc - as above
- log_pf, log_var - as above
- RSE_est - identical to cov_mc
- ESS - effective number of failure samples $(\sum I_i w_i)^2 / \sum (I_i w_i)^2$
- ESS_modes - effective sample size per mixture component (serial systems with stability_mode = "fast", otherwise NULL)
- n_mc - number of samples used

Limit-State Function (LSF) Interface

The limit-state function follows the sign convention $g(x) > 0$ safe, $g(x) = 0$ limit state and $g(x) < 0$ failure. It must use one of the following signatures:

`function(x)` A single numeric vector `x` containing all basic variables in the order defined in `lDistr`. This form also allows vectorized evaluation and is recommended.

`function(Z, Fy, M, ...)` Explicitly named scalar arguments corresponding to the variable names defined in the probabilistic model.

The use of `function(...)` as the sole argument is not supported, because it leads to inconsistent behaviour between FORM-based and Monte Carlo-based reliability methods.

Windows users

On Windows, `backend = "future"` with `future::plan(multisession, workers = ...)` is recommended. `backend = "parallel"` creates a new PSOCK cluster for every call and issues a warning.

Author(s)

(C) 2021-2026 K. Nille-Hauf, T. Feiri, M. Ricker, T. Lux – Hochschule Biberach (until 2022), TU Dortmund University - Chair of Structural Concrete (since 2023)

References

Ditlevsen, O., & Madsen, H. O. (1996). *Structural Reliability Methods*. Wiley.

Melchers, R. E., & Beck, A. T. (2018). *Structural Reliability Analysis and Prediction*. Wiley.

Spaethe, G. (1991). *Die Sicherheit tragender Baukonstruktionen*. Springer.

`MC_SubSam` *Monte Carlo Simulation with Subset-Sampling*

Description

MonteCarlo with Subset-Sampling (MC_SubSam) method for the estimation of small failure probabilities in structural reliability analysis.

Subset Simulation is an advanced Monte Carlo technique that expresses a rare failure event as a sequence of intermediate conditional events with higher probabilities.

Failure is defined as $g(x) < 0$. The intermediate events are $g(x) \leq q_j$ with the `p0`-quantile q_j of the limit-state values of level j. Only the last level uses the failure event $g(x) < 0$.

The method combines conditional sampling with Markov Chain Monte Carlo (MCMC) techniques to efficiently estimate very small probabilities of failure, which would be computationally expensive to obtain using crude Monte Carlo simulation.

Usage

```
MC_SubSam(
  lsf,
  lDistr,
  Nsubset = 1e+05,
  p0 = 0.1,
  MaxSubsets = 10,
  Alpha = 0.05,
  variance = "uniform",
  debug.level = 0,
  seed = NULL
)
```

Arguments

lsf	limit-state function
lDistr	list of basevariables in input space
Nsubset	number of samples in each simulation level
p0	level probability or conditional probability
MaxSubsets	maximum number of simulation levels that are used to terminate the simulation procedure to avoid infinite loop when the target domain cannot be reached
Alpha	Significance level of the confidence interval. The interval covers the probability 1 - Alpha (default 0.05, i.e. 95 percent).
variance	gaussian, uniform
debug.level	If 0 no additional info if 2 high output during calculation
seed	Optional integer value. If specified, sets a fixed seed for reproducible random numbers (useful for testing).

Value

MC_SubSam returns an object containing the following elements:

- beta: Estimated reliability index.
- pf: Estimated probability of failure.
- betaCI: Confidence interval of the reliability index.
- pfCI: Confidence interval of the probability of failure.
- CoV: Coefficient of variation of the estimator.
- NumOfSubsets: Number of simulation levels.
- NumOfEvalLSF_nom: Nominal number of limit-state function evaluations (Markov chains / iterations).
- NumOfEvalLSF_eff: Effective number of limit-state function evaluations.
- runtime: Total runtime of the function.

Author(s)

(C) 2021-2026 K. Nille-Hauf, T. Feiri, M. Ricker, T. Lux – Hochschule Biberach (until 2022), TU Dortmund University - Chair of Structural Concrete (since 2023)

References

Au, S. K., & Beck, J. L. (2001). Estimation of small failure probabilities in high dimensions by subset simulation. *Probabilistic Engineering Mechanics*, 16(4), 263-277.

Marelli, S., & Sudret, B. (2014). UQLab: A framework for uncertainty quantification in Matlab. In *Proceedings of the 2nd International Conference on Vulnerability, Risk Analysis and Management (ICVRAM2014)*, Liverpool, United Kingdom, 2554-2563.

`MVFOSM` *Mean-Value First-Order Second-Moment (MVFOSM)*

Description

Mean-Value First-Order Second-Moment (MVFOSM) method for the approximation of failure probabilities in structural reliability analysis.

The limit-state function is linearised at the mean values of the basic variables. The reliability index is

$$\beta = \frac{g(\mu)}{\sqrt{\sum_i \left(\frac{\partial g}{\partial x_i}\Big|_{\mu} \sigma_i \right)^2}}$$

and the probability of failure is approximated by $p_f = \Phi(-\beta)$.

Failure is defined as $g(x) < 0$. If $g(\mu) < 0$, the reliability index is negative.

Only the mean values and standard deviations of the basic variables are used. The distribution types do not influence the result. The method is exact for linear limit-state functions of normal variables. For nonlinear limit-state functions the result depends on how the limit-state function is formulated (e.g. $R - S$ or $R/S - 1$ give different values of β for the same failure domain).

Usage

```
MVFOSM(lsf, lDistr, h = 1e-04, isExpression = FALSE, debug.level = 0)
```

Arguments

`lsf`	Limit-state function, either a function or an expression, as indicated by `isExpression`. Supplied automatically by a SYS_ object (SYS_PROB or SYS_PARAM) when the method is run through a PROB_MACHINE. Do not provide it in the machine's options.
`lDistr`	List of distributions as returned by `getlDistr()` of the basic variables. Each variable must have a finite mean and standard deviation. Supplied automatically as well.

h	Step of the central finite differences, relative to the standard deviation of each variable (default 1e-4). Only used if isExpression = FALSE.
isExpression	Logical. If TRUE, lsf must be an expression and the derivatives are computed symbolically. Otherwise lsf must be a function.
debug.level	Verbosity level: 0 = silent, 1 = basic info, 2 = detailed output.

Details

When MVFOSM is run through a PROB_MACHINE (fCall = "MVFOSM"), lsf and lDistr are supplied automatically by SYS_PROB or SYS_PARAM and must not be part of the machine's options. Only h, isExpression and debug.level are passed as options. With isExpression = TRUE the system passes the symbolic expression of the limit-state function instead of the function.

The mean values and standard deviations are taken from the distribution objects (fields mean and Sd), not from the native distribution parameters. All basic variables must therefore have a finite mean and standard deviation. This excludes for example the log-Student-t distribution ("lt"), for which the moments are not defined. In this case the function stops with an error.

With isExpression = FALSE the partial derivatives are computed by central finite differences with the step $h\sigma_i$ for variable i. The step is relative to the standard deviation, so that variables of very different magnitude (e.g. a section modulus in m^3 and a yield strength in kN/m^2) are handled equally. For a variable with $\sigma_i = 0$ the step $h \max(|\mu_i|, 1)$ is used. With isExpression = TRUE the derivatives are computed symbolically with D.

Value

MVFOSM returns a list with the following elements:

- method: Character string "MVFOSM".
- beta: Reliability index.
- pf: Probability of failure $\Phi(-\beta)$.
- design.point.x: Approximate design point in the original x-space, $\mu_i + \alpha_i \beta \sigma_i$.
- alpha: Direction cosines (sensitivity factors).
- runtime: Runtime as returned by proc.time().

Author(s)

(C) 2021-2026 K. Nille-Hauf, T. Feiri, M. Ricker, T. Lux – Hochschule Biberach (until 2022), TU Dortmund University - Chair of Structural Concrete (since 2023)

References

Freudenthal, A. M. (1956). Safety and the probability of structural failure. *Transactions of the American Society of Civil Engineers*, 121, 1337-1397.

Nowak, A. S., & Collins, K. R. (2000). *Reliability of Structures*. McGraw-Hill.

PARAM_BASEVAR-class *Parametric Random Variable (PARAM_BASEVAR)*

Description

Extends ‘PROB_BASEVAR‘ so that one of the distribution parameters (Mean, Sd or DistributionType) can be **swept** over a user-defined vector of values.

Details

The method ‘nextParam()‘ advances ‘pos‘ (circularly) and updates the basic fields (‘Mean‘, ‘Sd‘, ‘DistributionType‘, ‘DistributionParameters‘) before calling ‘prepare()‘ again. ‘getCurrentParam()‘ returns the *last* value that was used (convenient for readable run names).

Fields

`ParamValues` Numeric vector holding the values that will be assigned during the sweep.

`ParamType` Which parameter is varied: ‘"Mean"‘, ‘"Sd"‘ or ‘"DistributionType"‘.

`pos` Current index inside ‘ParamValues‘ (used internally).

Author(s)

(C) 2021-2026 K. Nille-Hauf, T. Feiri, M. Ricker, T. Lux – Hochschule Biberach (until 2022), TU Dortmund University - Chair of Structural Concrete (since 2023)

Examples

```
pvar <- PARAM_BASEVAR(
  Name = "E_mod",
  DistributionType = "norm",
  Sd = 3000,
  ParamType = "Mean",
  ParamValues = c(30e3, 35e3, 40e3)
)
pvar$getCurrentParam() # first value (30000), set by the constructor
pvar$nextParam()       # sets Mean = 35000 and prepares the distribution
pvar$getCurrentParam()
```

PARAM_DETVAR-class *Parametric Deterministic Variable (PARAM_DETVAR)*

Description

Like ‘PROB_DETVAR‘ but the deterministic value itself can be swept. The class inherits from ‘PROB_BASEVAR‘ and therefore re-uses the same transformation machinery.

Fields

`ParamValues` Vector of deterministic values that will be assigned sequentially in a parametric sweep.

`pos` Current index (used internally).

Author(s)

(C) 2021-2026 K. Nille-Hauf, T. Feiri, M. Ricker, T. Lux – Hochschule Biberach (until 2022), TU Dortmund University - Chair of Structural Concrete (since 2023)

Examples

```
pdet <- PARAM_DETVAR(
  Name = "E_mod",
  ParamValues = c(30e3, 35e3, 40e3)
)
pdet$getCurrentParam() # first value (30000), set by the constructor
pdet$nextParam()       # sets Mean = 35000 and prepares the distribution
pdet$getCurrentParam()
```

PARAM_LSF-class *Parametric Limit-State Function (PARAM_LSF)*

Description

Identical to ‘SYS_LSF‘ but inherits from it to make the class hierarchy clear when a parametric study is performed. No additional fields or methods are required.

Author(s)

(C) 2021-2026 K. Nille-Hauf, T. Feiri, M. Ricker, T. Lux – Hochschule Biberach (until 2022), TU Dortmund University - Chair of Structural Concrete (since 2023)

PROB_BASEVAR-class *Object to store the distribution model for basic variables (PROB_BASEVAR)*

Description

Stores the distribution model for a basic random variable that can be used in any limit-state function. The class knows how to transform between *mean / standard deviation* and the native distribution parameters.

Details

The method ‘prepare()‘ calculates missing parameters and checks for consistency. ‘getlDistr()‘ returns a list containing *four* functions (‘d‘, ‘p‘, ‘q‘, ‘r‘) together with meta-information; the result is cached to avoid repeated ‘loadNamespace()‘ calls.

Fields

`Id` Integer identifier (position inside the LSF vector).

`Name` Human-readable name (e.g. ‘"f_ck"‘).

`Description` Short textual description.

`DistributionType` Distribution identifier (‘"norm"‘, ‘"lnorm"‘, ‘"slnorm"‘, ‘"gumbel"‘, ‘"gamma"‘, ‘"exp"‘, ‘"beta"‘, ‘"st"‘, ‘"binom"‘,‘"emp"‘, ‘"unif"‘, ‘"weibull"‘, ‘"lt"‘).

`Mean` Mean value of the variable.

`Sd` Standard deviation.

`Cov` Coefficient of variation (‘Sd / Mean‘).

`x0` Shifting parameter (used for shifted log-normal distribution ‘"slnorm"‘).

`Package` Name of the R package that provides the density functions (‘"stats"‘, ‘"evd"‘, ‘"EnvStats"‘ or ‘"TesiproV"‘).

`DistributionParameters` Numeric vector with the *native* distribution parameters (e.g. ‘c(mu, sigma)‘ for a normal).

`.cache` Private environment that stores cached density functions (used internally, **do not** access from user code).

Methods

`fromJCSS(BaseVar, NominalValue, ComponentType = "ready-mixed", ...)` Sets the distribution according to the JCSS Probabilistic Model Code and modifies this object. BaseVar = ’fc’: concrete compressive strength, NominalValue is the strength class (e.g. ’C30’), ComponentType is ’ready-mixed’ or ’pre-cast’. The result is a log-Student-t distribution (’lt’). BaseVar = ’fy’: yield strength, NominalValue is the steel grade (e.g. ’B500’), ComponentType is ’reinforcing steel’. The result is a normal distribution. Not all combinations are tabulated.

`genHyperParam(n.gen.num = 15, n.gen.samp = 10000, ...)` Returns a new PROB_BASEVAR with a Student-t ('st') or log-Student-t ('lt') distribution for this normal or lognormal variable. The hyperparameters c(m', s', n', v') are estimated by maximum likelihood (Rackwitz 1983) from n.gen.samp simulated samples of size n.gen.num. The result depends on the random numbers, use set.seed() for reproducible values.

`plot( x_range = NULL, n = 500, add = FALSE, compare = FALSE, prior = FALSE, unit = NULL, ... )` Plots the probability density function of this basic variable. If prior = TRUE and this object was produced by updateBasevar(), the density of the prior is drawn into the same plot (prior vs. posterior). unit is an optional character string with the unit of the variable (e.g. "mm"). It is added to the x axis label as [unit]. If compare = TRUE and this object was produced by updateBasevar(), the fitted parametric curve (if a fit was attempted) and the raw empirical resample density are overlaid for visual comparison.

`prepare()` Performs transformations between mean/sd and distribution parameters. Ensures that Mean, Sd and Cov are consistent and valid.

`updateBasevar(PROB_QUALCON, debug.level = 0, min_r2 = 0.99)` Bayesian update with the OC curve of a PROB_QUALCON object. This variable must have DistributionType 'st' or 'lt' with the hyperparameters c(m', s', n', v'). Returns a new PROB_BASEVAR with the distribution after acceptance of the lot, this object is not changed. The outgoing distribution is fitted by the same distribution type. If the fit gives R^2 < min_r2, an empirical distribution ('emp') is returned instead. debug.level = 1 prints progress messages.

Author(s)

(C) 2021-2026 K. Nille-Hauf, T. Feiri, M. Ricker, T. Lux – Hochschule Biberach (until 2022), TU Dortmund University - Chair of Structural Concrete (since 2023)

Examples

```
## Normal random variable ------------------------------------------------
var1 <- PROB_BASEVAR(
  Name = "f_ck", DistributionType = "norm",
  Mean = 30, Sd = 1
)
var1$prepare()

## Gumbel random variable (package `evd`) ---------------------------------
var2 <- PROB_BASEVAR(
  Name = "M", DistributionType = "gumbel",
  Package = "evd", Mean = 2000, Sd = 200
)
var2$prepare()
```

`PROB_DETVAR-class` *Object to store a deterministic parameters (PROB_DETVAR)*

Description

A convenience subclass of ‘PROB_BASEVAR‘ that represents a deterministic value (i.e. a variable with practically zero variance). The object is automatically converted to a normal distribution with an infinitesimal standard deviation (‘Sd = max(|Value| / 1e7, 1e-12)‘).

Inherited fields

Inherits all fields of ‘PROB_BASEVAR‘ (see that documentation for a full list).

Additional fields

`Value` The deterministic value used internally as `Mean`.

Author(s)

(C) 2021-2026 K. Nille-Hauf, T. Feiri, M. Ricker, T. Lux – Hochschule Biberach (until 2022), TU Dortmund University - Chair of Structural Concrete (since 2023)

Examples

```
det <- PROB_DETVAR(Name = "E_mod", Value = 30e3)
det$prepare()
```

PROB_MACHINE-class *Reliability Algorithm (PROB_MACHINE)*

Description

Stores the definition of a reliability method (FORM, SORM, Monte-Carlo, etc.) that can be executed by ‘SYS_PROB‘ or ‘SYS_PARAM‘. The actual algorithm is called via the character string ‘fCall‘ (e.g. ‘"FORM"‘ or ‘"MC_IS"‘). Optional arguments are passed through the list ‘options‘.

Details

The limit-state function (‘lsf‘) and the basevariables (‘lDistr‘) are supplied by the system (‘SYS_PROB‘ or ‘SYS_PARAM‘) when the machine is run. They must not be part of ‘options‘. ‘options‘ only holds the remaining arguments of the algorithm. Arguments that are not set use the defaults of the algorithm.

Arguments that can be set via ‘options‘ (defaults in brackets, see the help page of the respective function for details):

fCall	Options
‘FORM‘	‘n_optim‘ (100), ‘loctol‘ (1e-5), ‘optim_type‘ ("auto"), ‘lsf_list‘, ‘debug.level‘
‘SORM‘	‘debug.level‘
‘MVFOSM‘	‘h‘ (1e-4), ‘isExpression‘ (FALSE), ‘debug.level‘
‘MC_CRUDE‘	‘cov_user‘ (0.05), ‘n_batch‘ (10000), ‘n_max‘ (1e7), ‘use_threads‘, ‘backend‘, ‘dataRecord‘ (TRUE), ‘debu

‘MC_IS‘	as ‘MC_CRUDE‘, plus ‘sys_type‘, ‘beta_l‘, ‘densityType‘, ‘dps‘, ‘adaptive_alpha‘, ‘alpha_update_rate‘, ‘ad
‘MC_SubSam‘	‘Nsubset‘ (1e5), ‘p0‘ (0.1), ‘MaxSubsets‘ (10), ‘Alpha‘ (0.05), ‘variance‘ ("uniform"), ‘debug.level‘, ‘seed‘

The helper method ‘getMethodLevel()‘ returns the level of the method: 2 for MVFOSM, FORM and SORM, 3 for the Monte Carlo methods.

Fields

`name` Descriptive name of the machine (appears in reports).

`fCall` Function name that implements the algorithm (must be available in the package namespace, e.g. ‘FORM‘, ‘SORM‘, ‘MC_CRUDE‘, ‘MC_IS‘, ‘MC_SubSam‘).

`options` Additional options for the algorithm (list, e.g. ‘list(n_max = 1e6, cov_user = 0.05)‘).

Author(s)

(C) 2021-2026 K. Nille-Hauf, T. Feiri, M. Ricker, T. Lux – Hochschule Biberach (until 2022), TU Dortmund University - Chair of Structural Concrete (since 2023)

Examples

```
## FORM machine ------------------------------------------------------------
form_rf <- PROB_MACHINE(
  name = "FORM - Rack-Fies",
  fCall = "FORM",
  options = list(n_optim = 20, loctol = 1e-3, optim_type = "rackfies")
)

## Monte-Carlo importance sampling -----------------------------------------
mcis <- PROB_MACHINE(
  name = "MC-IS",
  fCall = "MC_IS",
  options = list(cov_user = 0.05, n_max = 3e5, seed = 1234)
)
```

PROB_QUALCON-class *Sampling plan and OC curve (PROB_QUALCON)*

Description

Defines a quality-control sampling plan and computes the corresponding Operating Characteristic (OC) curve, i.e. the acceptance probability $P_a(\theta)$ as a function of the defect fraction θ.

A `PROB_QUALCON` object refers to a `PROB_BASEVAR` via the field `var`, because the quality control relates to one specific structural variable. The OC curve is used by `PROB_BASEVAR$updateBasevar()` to derive the posterior distribution of that variable after acceptance.

Two plan types are supported (field `ConformityType`):

- **Variable plan** (`"fck_EN-206"`): conformity criterion of EN 206 for the concrete compressive strength. A lot is accepted if the mean of `n` samples is at least $f_{ck} + 1.48s$ and the smallest value at least $f_{ck} - 4$. `f_ck` is given as `DefectLimitLower`, `ConformityParam = c(n)`. The OC curve is simulated, the sample size per defect fraction is set with the field `NSim`.
- **Attribute plan** (`"dim_k||n"`): binomial model, a lot is accepted if at most `c_accept` of `n` units are defective. `ConformityParam = c(n, c_accept)`. The plan can be used for any characteristic whose compliance with tolerance values is checked. A unit is defective if the variable lies below `DefectLimitLower` or above `DefectLimitUpper`. Both limits are freely selectable. They do not influence the OC curve, only the Bayesian update.

The fields `DefectLimitLower` and `DefectLimitUpper` define when a single unit counts as defective and thereby the defect fraction θ. They are not the acceptance criterion. How a lot is accepted is determined by the sampling plan alone.

The OC curve can only be computed for a variable `var` with `DistributionType` `"st"` or `"lt"` (Student-t and log-Student-t with hyperparameters $c(m', s', n', v')$).

Fields

`Id` Numeric identifier.

`Name` Short name of the plan (character).

`Description` Optional description (character).

`ConformityType` Type of the sampling plan, `"fck_EN-206"` or `"dim_k||n"` (character).

`ConformityParam` Parameters of the sampling plan (numeric). For `"fck_EN-206"`: `c(n)` with the sample size `n`. For `"dim_k||n"`: `c(n, c_accept)`. It is not needed if the OC curve is specified manually.

`NSim` Monte-Carlo sample size per defect fraction for the simulation of the OC curve (numeric, default 100000). It is only used by `"fck_EN-206"`.

`DefectLimitLower` Lower limit (numeric, default `-Inf`). A unit below it is defective. For `"fck_EN-206"` it is the characteristic compressive strength f_{ck} in the unit of the variable and must be set, also for a manually specified OC curve.

`DefectLimitUpper` Upper limit (numeric, default `Inf`). A unit above it is defective. For `"fck_EN-206"` it must stay `Inf`. For `"dim_k||n"` at least one of the two limits must be finite.

`var` The `PROB_BASEVAR` the plan applies to (any). It is `NULL` as long as no variable is attached. `getOC()` requires it.

`theta_grid` Defect fractions at which the OC curve is evaluated (numeric). A default grid from 0.001 to 0.9 is set if nothing is given. `getOC(force = TRUE)` resets it to this default grid.

`Pa_grid` Acceptance probabilities belonging to `theta_grid` (numeric). It is `NA` until `getOC()` has been run. A manually specified OC curve can be given as `Pa_grid` together with `theta_grid` of the same length.

`.prior_snapshot` Internal. Digest of the distribution of `var` at the time of the last `getOC()` call, used by `isStale()`.

Methods

`getOC(force = FALSE)` Computes the OC curve and stores it in $Pa_grid. ...

`isStale()` Returns TRUE if $var's distribution has changed since Pa_grid was last computed via getOC() (e.g. by later re-assigning Mean/Sd on the same object). Returns NA if Pa_grid was set manually or $var is missing, since no snapshot exists to compare against.

`plot(log_x = TRUE, ...)` Plots the OC curve (Pa_grid vs. theta_grid) using base R graphics. Runs $getOC() first if Pa_grid has not been computed yet.

`summary()` Prints a short summary of the PROB_QUALCON object.

Methods

`getOC(force = FALSE)` Computes the OC curve and stores it in `Pa_grid`. An existing curve is only recomputed with `force = TRUE`. With `force = TRUE` the default `theta_grid` is used again, a grid or curve given by the user is discarded.

`plot(log_x = TRUE, ...)` Plots `Pa_grid` against `theta_grid`. Runs `getOC()` first if necessary.

`isStale()` `TRUE` if the distribution of `var` has changed since `getOC()`, `NA` if no snapshot exists (manual curve or no variable).

`show()`, `summary()` Short printed representations.

Author(s)

(C) 2026 T. Lux, M. Ricker – TU Dortmund University - Chair of Structural Concrete

Examples

```
## Not run:
v <- PROB_BASEVAR$new(Name = "fc", DistributionType = "lt",
                      DistributionParameters = c(3.75, 0.105, 3, 10))
plan <- PROB_QUALCON$new(Name = "EN206_n15",
                         ConformityType = "fck_EN-206",
                         ConformityParam = 15, NSim = 2000,
                         DefectLimitLower = 30, var = v)
plan$getOC()
plan$plot()
post <- v$updateBasevar(plan)

# Attribute plan: at most 1 of 10 units outside the limits 27 and 33
d <- PROB_BASEVAR$new(Name = "cover", DistributionType = "st",
                      DistributionParameters = c(30, 2, 3, 10))
plan2 <- PROB_QUALCON$new(Name = "cover_plan", ConformityType = "dim_k||n",
                          ConformityParam = c(10, 1),
                          DefectLimitLower = 27, DefectLimitUpper = 33, var = d)
plan2$getOC()
post2 <- d$updateBasevar(plan2)

## End(Not run)
```

SORM *Second-Order Reliability Method (SORM)*

Description

Second-Order Reliability Method (SORM) for the approximation of failure probabilities in structural reliability analysis.

The SORM extends the First-Order Reliability Method (FORM) by incorporating second-order information of the limit-state function at the design point. By accounting for local curvature effects, SORM generally provides more accurate failure probability estimates for nonlinear limit-state functions.

Failure is defined as $g(x) < 0$ (see FORM). The curvature correction is applied to the FORM result.

The method evaluates the principal curvatures of the limit-state surface in standard normal space and applies an asymptotic correction to the FORM probability of failure.

The curvatures are determined numerically from second derivatives of the limit-state function in standard normal space. For distributions that are not smooth, e.g. empirical distributions (`"emp"`), finite differences are dominated by sampling noise. If the second derivatives depend strongly on the step size, a local quadratic regression around the design point is used instead. The curvatures, and thus the SORM result, are then less precise than for smooth distributions and should be checked against FORM or a simulation method.

Usage

```
SORM(lsf, lDistr, debug.level = 0)
```

Arguments

`lsf`	objective function with limit state function in form of `function(x) { x[1] + x[2] + ... }`.
`lDistr`	list of distributions (distribution objects of TesiproV)
`debug.level`	If 0 no additional info if 2 high output during calculation

Value

SORM returns an object containing the following elements:

- `beta`: Hasofer-Lind reliability index.
- `pf`: Estimated probability of failure.
- `u_points`: Design point(s) in the standard normal space (u-space).
- `dy`: Gradient(s) of the limit-state function at the design point.

Author(s)

(C) 2021-2026 K. Nille-Hauf, T. Feiri, M. Ricker, T. Lux – Hochschule Biberach (until 2022), TU Dortmund University - Chair of Structural Concrete (since 2023)

References

Breitung, K. (1989). Asymptotic approximations for probability integrals. *Probabilistic Engineering Mechanics*, 4(4), 187-190.

Cai, G. Q., & Elishakoff, I. (1994). Refined second-order reliability analysis. *Structural Safety*, 14(4), 267-276.

Hohenbichler, M., Gollwitzer, S., Kruse, W., & Rackwitz, R. (1987). New light on first- and second-order reliability methods. *Structural Safety*, 4, 267-284.

Tvedt, L. (1990). Distribution of quadratic forms in normal space - Applications to structural reliability. *Journal of Engineering Mechanics*, 116(6), 1183-1197.

Marelli, S., & Sudret, B. (2014). UQLab: A framework for uncertainty quantification in Matlab. In *Proceedings of the 2nd International Conference on Vulnerability, Risk Analysis and Management (ICVRAM2014)*, Liverpool, United Kingdom, 2554-2563.

Lacaze, S., & Missoum, S. (2015). CODES: A toolbox for computational design (Version 1.0). Retrieved from https://sites.arizona.edu/codes/

Wu, X. Z. (2017). Implementing statistical fitting and reliability analysis for geotechnical engineering problems in R. *Georisk: Assessment and Management of Risk for Engineered Systems and Geohazards*, 11(2), 173-188.

`StudentT` *Student t Distribution*

Description

Density, distribution function, quantile function and random generation for the Student t distribution.

Usage

```
dst(x, hyper.param)

pst(q, hyper.param)

qst(p, hyper.param)

rst(n_vals, hyper.param)
```

Arguments

`x`, `q`	Vector of quantiles.
`hyper.param`	Numeric vector (m, s, n, ν).
`p`	Vector of probabilities.
`n_vals`	Number of random values to generate.

Details

The cumulative distribution function is given by

$$F_X(x) = F_{t_\nu}\left(\frac{x-m}{s}\sqrt{\frac{n}{n+1}}\right),$$

where $F_{t_\nu}(\cdot)$ denotes the cumulative distribution function of a Student t distribution with ν degrees of freedom.

The Student t distribution arises as the marginal distribution of a normal distributed variable with conjugate normal-gamma prior uncertainty in m and s.

The hyperparameter vector `hyper.param` is defined as

$$(m, s, n, \nu)$$

where

- m is the mean value of an equivalent sample of size n,
- s is the empirical standard deviation of an equivalent sample of size $\nu + 1$,
- ν denotes the degrees of freedom of the Student t distribution.

For details see: https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/TDist

Value

- `dst()` returns the density.
- `pst()` returns the distribution function.
- `qst()` returns the quantile function.
- `rst()` generates random deviates.

Author(s)

(C) 2021-2026 K. Nille-Hauf, T. Feiri, M. Ricker, T. Lux – Hochschule Biberach (until 2022), TU Dortmund University - Chair of Structural Concrete (since 2023)

Examples

```
hp <- c(40, 4, 3, 10)
dst(0.5, hp)
```

SYS_LSF-class *System Limit-State Function (SYS_LSF)*

Description

A reference-class that stores a user-defined limit-state function. The function can be written either as an **expression** (‘expr‘) or as a **regular R function** (‘func‘). All random variables that appear in the function must be supplied in the ‘vars‘ list as ‘PROB_BASEVAR‘ objects.

Details

The method ‘getLSF()‘ returns a **closure** that expects a numeric vector ‘x‘ (ordered exactly like ‘vars‘) and evaluates the limit-state function.

Sign convention: $g(x) > 0$ is the safe domain, $g(x) = 0$ the limit state and $g(x) < 0$ failure. All reliability methods of TesiproV use this convention.

For consistency across FORM and Monte Carlo reliability methods, limit-state functions must either accept a single numeric vector argument or explicitly named scalar arguments. The use of `function(...)` is not supported.

Fields

`name` Optional human-readable name of the limit-state function.

`expr` Symbolic expression (e.g. ‘expression(f_ck - d_nom)‘) that is later converted to a callable function via ‘ExpressionToFunction()‘.

`func` Actual R function implementing the limit-state equation. May be ‘NULL‘ until defined by user or converted from ‘expr‘.

`vars` List of ‘PROB_BASEVAR‘ objects that provide the random variables.

Methods

`ExpressionToFunction()` Transforms a valid expression into a objective function. Need the set of Variables with correct spelled names and IDs

`check()` Checks all variables. You dont need to execute this, since the system object will do anyway.

Limit-State Function (LSF) API

A limit-state function must follow one of the two supported interface patterns:

`function(x)` A single numeric vector argument `x` containing all basic variables in the order specified in `vars`.

Example:

```
lsf$func <- function(x) {
  x[1] - sum((x[-1]^2) / seq_along(x)[-1])
}
```

`function(Z, Fy, M, ...)` Explicitly named scalar arguments corresponding to the variable names defined in `vars`.

Example:

```
lsf$func <- function(Z, Fy, M) {
  Z * Fy - M
}
```

The following form is **not supported**:

```
function(...)
```

Using `...` as the sole argument leads to inconsistent behaviour between FORM-based and Monte Carlo-based reliability methods and is therefore prohibited.

For maximum robustness and compatibility across all reliability algorithms, the use of the vector form `function(x)` is recommended.

Author(s)

(C) 2021-2026 K. Nille-Hauf, T. Feiri, M. Ricker, T. Lux – Hochschule Biberach (until 2022), TU Dortmund University - Chair of Structural Concrete (since 2023)

Examples

```
vars <- list(
  PROB_BASEVAR(Name = "f_ck", DistributionType = "norm", Mean = 30, Sd = 1),
  PROB_BASEVAR(Name = "d_nom", DistributionType = "norm", Mean = 0.2, Sd = 0.01)
)
lsf <- SYS_LSF(name = "Bending resistance", expr = expression(f_ck - d_nom), vars = vars)
lsf$ExpressionToFunction()
lsf$func(c(30, 0.2))
```

SYS_PARAM-class *Object for parametric Studies (SYS_PARAM)*

Description

Extends ‘SYS_PROB‘ to perform a **parametric sweep** over one or more ‘PARAM_BASEVAR‘ objects. For each combination of parameter values a full reliability analysis is executed and the results are stored in ‘beta_params‘ and ‘res_params‘.

Details

The method ‘runMachines()‘ is almost identical to the one in ‘SYS_PROB‘, but it loops over **all** parameter values, creates a readable run name, and stores the whole result hierarchy.

Fields

`beta_params` List of β-matrices (one matrix per parameter run).

`res_params` List of detailed result objects (one per run).

`nParams` Numeric vector: number of parameter values for each 'PARAM_BASEVAR'.

`nLsfs` Number of limit-state functions in the system.

`nMachines` Number of analysis machines (methods) used.

Methods

`printResults(path = "")` TesiproV can create a report file with all the necessary data for you. If you provide a path (or filename, without ending) it will store the data there, otherwise it will report to the console. Set the path via setwd() or check it via getwd().

`runMachines()` Starts solving all given problems (sys_input) with all given algorithms (probMachines). First checks that all analysis methods are loaded before validating limit-state functions.

Author(s)

(C) 2021-2026 K. Nille-Hauf, T. Feiri, M. Ricker, T. Lux – Hochschule Biberach (until 2022), TU Dortmund University - Chair of Structural Concrete (since 2023)

Examples

```
# Simple 2-parameter sweep ------------------------------------------------
# two random variables that will be varied
pv1 <- PARAM_BASEVAR(
  Name = "E_mod",
  DistributionType = "norm",
  Sd = 3000,
  ParamType = "Mean",
  ParamValues = c(30e3, 35e3, 40e3)
)
pv2 <- PARAM_BASEVAR(
  Name = "f_ck",
  DistributionType = "norm",
  Sd = 5,
  ParamType = "Mean",
  ParamValues = c(30, 35, 40)
)

lsf <- SYS_LSF(vars = list(pv1, pv2), name = "Bending")
ps <- SYS_PARAM(
  sys_input = list(lsf),
  probMachines = list(PROB_MACHINE(name = "FORM", fCall = "FORM")),
  sys_type = "serial"
)

## Not run:
ps$runMachines()
```

```
ps$beta_params # matrix of \eqn{\beta}-values for every run

## End(Not run)
```

SYS_PROB-class *System Probability Solution Object (SYS_PROB)*

Description

The SYS_PROB class represents a probabilistic system consisting of one or more limit-state functions (SYS_LSF) and a set of reliability algorithms (PROB_MACHINE).

It allows the execution of reliability analyses for serial or parallel systems, aggregation of single limit-state results, and system-level probability evaluation.

Each limit-state function may include multiple random variables (PROB_BASEVAR) with different probability distributions. The transformation between mean/standard deviation and the native distribution parameters is handled automatically.

- Normal ("norm"): mean = μ, standard deviation = σ. Parameters: mean, sd.
- Lognormal ("lnorm"): defined by mean m and sd s. Parameters: $\mu = \log\left(\frac{m}{\sqrt{1+(s/m)^2}}\right)$, $\sigma = \sqrt{\log\left(1+(s/m)^2\right)}$.
- shifted Lognormal ("slnorm"): defined by Mean m, Sd s and $x0$. Parameters: $\mu = \log\left(\frac{m-x0}{\sqrt{1+(s/(m-x0))^2}}\right)$, $\sigma = \sqrt{\log\left(1+(s/(m-x0))^2\right)}$.
- Gumbel ("gumbel", package "evd"): location-scale type I extreme value distribution. Parameters: location = $\mu - \gamma \cdot \beta$, where γ is the Euler-Mascheroni constant, and $\beta = \frac{\sigma\sqrt{6}}{\pi}$.
- Gamma ("gamma"): shape-scale parameterization. Parameters: $k = \frac{\mu^2}{\sigma^2}$, $\theta = \frac{\sigma^2}{\mu}$.
- Exponential ("exp"): rate λ or scale $\theta = 1/\lambda$. Parameter: $\lambda = 1/\mu$.
- Beta ("beta"): bounded on $[0, 1]$. Parameters: $\alpha = \mu t$, $\beta = (1-\mu)t$, with $t = \frac{\mu(1-\mu)}{\sigma^2} - 1$.
- Weibull ("weibull"): shape k and scale λ. k is determined numerically from the coefficient of variation, with the empirical approximation $k \approx \left(\frac{\sigma}{\mu}\right)^{-1.086}$ as fallback, $\lambda = \frac{\mu}{\Gamma\left(1+\frac{1}{k}\right)}$.
- Empirical ("emp"): non-parametric distribution defined by observed sample. Parameter: obs = numeric vector of observations. The empirical CDF is constructed from the ordered sample. Duplicate values are allowed but may trigger a warning.
- Student t ("st"): marginal density of a normal-gamma model describing uncertainty in μ and σ. Hyperparameters: (m, s, n, ν).
- Log-Student t ("lt"): marginal density of a log-normal-gamma model describing uncertainty in μ and σ. Hyperparameters: (m, s, n, ν).

The types ‘"binom"‘ and ‘"unif"‘ are reserved for future extensions. The package automatically checks consistency between mean, standard deviation and coefficient of variation.

Methods available in this class:

* ‘$runMachines()‘ - executes all solution algorithms for each limit state function. * ‘$calculateSystemProbability()‘ - computes system reliability using bounds or Monte Carlo based methods (‘MC_IS‘, ‘MC_CRUDE‘, ‘MC_SubSam‘). * ‘$printResults()‘ - prints a detailed report to console or file. * ‘$saveProject(level)‘ - saves results at different detail levels.

Details

The class is built on `setRefClass` and therefore uses mutable fields. All objects (‘PROB_BASEVAR‘, ‘SYS_LSF‘, ...) are expected to be already **checked** (via the ‘check()‘ method) before they are passed to ‘SYS_PROB‘.

Fields

`sys_input` List of ‘SYS_LSF‘ objects - the individual limit-state functions.

`sys_type` Character string, ‘"serial"‘ or ‘"parallel"‘. Used by ‘calculateSystemProbability()‘.

`probMachines` List of ‘PROB_MACHINE‘ objects - the analysis methods.

`res_single` List of results per machine for each limit-state function.

`res_sys` List of results for the system-level reliability calculation.

`beta_single` Matrix of β-values obtained from the single-problem analyses.

`beta_sys` Matrix of β-values obtained from the system reliability calculations.

`params_sys` List that stores the parameter sets used in Monte-Carlo runs.

`debug.level` Numeric verbosity level (0 = silent, 1 = basic, 2 = detailed).

Methods

`calculateSystemProbability(calcType = "simpleBounds", params = list())` Calculates the system probability if more than one lsf is given and a system_type (serial or parallel) is set. If calcType is empty (or simpleBounds), only simple bounds are computed from the single results of runMachines() (the machines can be FORM, SORM, ...). If calcType is MCIS, an importance sampling method for systems is used (serial and parallel systems). If calcType is MCC, a crude Monte Carlo simulation of the system limit-state function is used. If calcType is MCSUS, subset simulation of the system limit-state function is used. A system analysis with FORM or SORM is not available. You can pass arguments to methods via the params field, while the argument has to be a named list (for example check the vignette).

`printResults(path = "")` TesiproV can create a report file with all the necessary data for you. If you provide a path (or filename, without ending) it will store the data there, otherwise it will report to the console. Set the path via setwd() or check it via getwd().

`runMachines()` Starts solving all given problems (sys_input) with all given algorithms (probMachines). First checks that all analysis methods are loaded before validating limit-state functions.

`saveProject(level, filename = "tesiprov_project")` You can save your calculation project with saveProject(). There are four different levels of detail to save 1st Level: Only the beta values 2nd Level: The result Objects of single or systemcalculation 3th Level: All The Probablity System Object, including limit state functions, machines and solutions 4th Level: An image of your entire workspace

Author(s)

(C) 2021-2026 K. Nille-Hauf, T. Feiri, M. Ricker, T. Lux – Hochschule Biberach (until 2022), TU Dortmund University - Chair of Structural Concrete (since 2023)

Examples

```
ps <- SYS_PROB(
  sys_input = list(SYS_LSF(), SYS_LSF()),
  probMachines = list(PROB_MACHINE()),
  sys_type = "serial"
)
## Not run:
ps$runMachines()
ps$beta_sys
ps$res_sys
ps$printResults("example_1")
ps$saveProject(4, "example_1")

## End(Not run)
```

Index

D, *17*
dlt *(*LogStudentT*)*, 5
dst *(*StudentT*)*, 27

FORM, 3, *26*

Internal helper functions for MC_IS(), 5

LogStudentT, 5
logStudentT *(*LogStudentT*)*, 5
lt *(*LogStudentT*)*, 5

MC_CRUDE, 7
MC_IS, 9
MC_SubSam, 14
MVFOSM, 16

PARAM_BASEVAR *(*PARAM_BASEVAR-class*)*, 18
PARAM_BASEVAR-class, 18
PARAM_DETVAR *(*PARAM_DETVAR-class*)*, 19
PARAM_DETVAR-class, 19
PARAM_LSF *(*PARAM_LSF-class*)*, 19
PARAM_LSF-class, 19
plt *(*LogStudentT*)*, 5
PROB_BASEVAR *(*PROB_BASEVAR-class*)*, 20
PROB_BASEVAR-class, 20
PROB_DETVAR *(*PROB_DETVAR-class*)*, 21
PROB_DETVAR-class, 21
PROB_MACHINE *(*PROB_MACHINE-class*)*, 22
PROB_MACHINE-class, 22
PROB_QUALCON *(*PROB_QUALCON-class*)*, 23
PROB_QUALCON-class, 23
pst *(*StudentT*)*, 27

qlt *(*LogStudentT*)*, 5
qst *(*StudentT*)*, 27

rlt *(*LogStudentT*)*, 5
rst *(*StudentT*)*, 27

SORM, 26
st *(*StudentT*)*, 27
StudentT, 27
SYS_LSF *(*SYS_LSF-class*)*, 29
SYS_LSF-class, 29
SYS_PARAM *(*SYS_PARAM-class*)*, 30
SYS_PARAM-class, 30
SYS_PROB *(*SYS_PROB-class*)*, 32
SYS_PROB-class, 32

TesiproV *(*TesiproV-package*)*, 2
TesiproV-package, 2