Package: TesiproV (via r-universe)

May 22, 2026

Type Package

Title Reliability Analysis Methods for Structural Engineering

Version 0.9.6

Date 2026-04-22

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. Have fun and enjoy. References: Spaethe (1991, ISBN:3-211-82348-4) ``Die Sicherheit tragender Baukonstruktionen", AU,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

RoxygenNote 7.3.3

Imports stats, future, future.apply, digest, nloptr, methods, ggplot2, gridExtra, pracma

Suggests testthat, knitr, rmarkdown, evd, brms

VignetteBuilder knitr

NeedsCompilation no

Author Konstantin Nille-Hauf [aut], Tânia Feiri [aut], Marcus Ricker [aut], Til Lux [aut, cre]

Config/pak/sysreqs cmake

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

Date/Publication 2026-04-22 14:13:30 UTC

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

RemoteRef HEAD

RemoteSha e02c7f390775854126ff1bb0bd30d3ab3da9f830

Contents

`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.

Usage

```
FORM(
  lsf,
  lDistr,
  n_optim = 10,
  loctol = 0.01,
  optim_type = "rackfies",
  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`	Number of optimization cycles (not required for Lagrangian algorithms).
`loctol`	Local tolerance for convergence of the solver algorithm.
`optim_type`	Optimization type: ‘"auglag"‘ (Augmented Lagrangian) or ‘"rackfies"‘ (Rackwitz-Fiessler iterative scheme).
`debug.level`	Verbosity level: 0 = silent, 1 = basic info, 2 = detailed output.

Value

A list containing: * ‘beta‘ - Hasofer-Lind reliability index * ‘pf‘ - probability of failure * ‘x_points‘ - design point in physical space * ‘dy‘ - gradient vector at design point

Author(s)

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.

`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 a log-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 crude Monte Carlo method estimates the probability of failure by direct sampling of the basic random variables and evaluation of the limit-state function.

The failure probability is estimated as

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

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 = parallel::detectCores(),
  backend = c("future", "parallel"),
  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	Batch size per iteration (used for parallel computation).
n_max	Maximum number of Monte Carlo samples (stopping criterion).
use_threads	Number of parallel threads. Set to 1 for single-core execution. Parallel execution is not supported on Windows.
backend	Parallel backend, either "future" (default) or "parallel".
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.

Value

MC_CRUDE returns an object containing the following elements:

- method: Character string identifying the method ("MCC").
- 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 specified by the user.
- n_mc: Total number of Monte Carlo samples generated.
- n_max: Maximum allowed number of samples.
- n_batch: Batch size per iteration.
- n_threads: Number of threads used.
- runtime: CPU time returned by proc.time().
- data: Optional data frame containing 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`.

Example:

```
lsf <- 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 the probabilistic model.

Example:

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

The following form is **not supported**:

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

Using `...` as the sole argument may lead to inconsistent behaviour across different reliability algorithms and is therefore prohibited.

For maximum robustness and cross-method compatibility, the vector form `function(x)` is recommended.

Windows users

On Windows systems, it is recommended to use `backend = "future"` for multi-core parallelization. The `"parallel"` backend does not support forking on Windows.

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 structural failure probabilities for single limit state functions or systems of limit state functions.

The method reduces the variance of crude Monte Carlo by sampling from a shifted (or multimodal) density in standard normal space and correcting via likelihood ratios. For systems, a multimodal sampling density based on FORM design points is used (cf. Melchers & Beck, Chapter 5).

The implementation supports:

- Single limit state functions (FORM-based IS shift)
- Serial and parallel systems (union / intersection events)
- Multimodal sampling densities for system reliability
- Optional adaptive mixture weights (adaptive α_i)
- Parallel execution (via **future** or **parallel**)
- Deterministic reproducibility via fixed seeds
- Automatic detection of vectorized limit-state functions

Usage

```
MC_IS(
  lsf,
  lDistr,
  cov_user = 0.05,
  n_batch = 5000,
  n_max = 1e+06,
  use_threads = 4,
  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 single limit-state function `function(x)` or a list of such functions for system reliability. Must follow the LSF interface specification described below.
`lDistr`	List of marginal distribution objects (funlists with `$d`, `$p`, `$q`, `$r`) corresponding to input variables.
`cov_user`	Target coefficient of variation (CoV) for the Monte Carlo estimator. Simulation stops once this threshold is reached.
`n_batch`	Number of samples generated per iteration.
`n_max`	Maximum total number of samples (upper stopping limit).
`use_threads`	Number of worker threads (for parallel execution).
`backend`	Parallel backend, either `"future"` or `"parallel"`.
`sys_type`	Character string, either `"serial"` or `"parallel"`, specifying system configuration.
`dataRecord`	Logical; if `TRUE`, intermediate results per iteration are stored and returned.
`beta_l`	Optional threshold; limit states with $\beta > beta_l$ may be excluded in system analysis.
`densityType`	Sampling density type (currently `"norm"` supported).
`dps`	Optional vector of design points in physical space; if supplied, FORM analysis is skipped.
`debug.level`	Integer verbosity level (0 = silent, 1 = summary, 2 = detailed).
`seed`	Optional integer seed for reproducible random numbers.
`adaptive_alpha`	Logical; if `TRUE`, adaptive mixture weights are used (system case only).
`alpha_update_rate`	Damping factor $\lambda \in [0, 1]$ controlling adaptation speed of mixture weights.
`adaptive_batch`	Logical; if TRUE, the batch size is adapted dynamically based on the current effective sample size (ESS) and target RSE.
`batch_control`	List controlling adaptive batch behaviour: **n_min** Minimum batch size **n_max** Maximum batch size **K_future** Desired number of remaining iterations
`min_adapt_samples`	Integer. Minimum total number of MC samples required before adaptive mixtures are updated. The adaption mechanism is activated only after this threshold has been exceeded to ensure statistically stable estimates.
`alpha_min`	Numeric scalar in $(0, 1)$. Lower bound for mixture weights α_i during adaptive updating. This prevents individual components of the importance sampling density from collapsing to zero and ensures numerical stability in multimodal system analyses.
`stability_mode`	Character string specifying the numerical stabilization strategy used for importance sampling weight accumulation.

"robust" Default. Uses full log-domain accumulation of weights across Monte Carlo iterations. This improves numerical stability for high-dimensional systems and very small failure probabilities.

"fast" Uses shifted exponentiation with batch-level scaling. Computationally equivalent under normal engineering conditions but may be slightly less stable in extreme rare-event scenarios.

Both modes are mathematically equivalent for typical reliability problems. The `"robust"` mode is recommended for research applications and extreme reliability levels.

Details

Single limit state case: The importance sampling density is constructed in standard normal space using the FORM design point u^*. Samples are generated from $u \sim \mathcal{N}(u^*, I)$ and transformed to physical space.

System case: For a system with limit states g_i, the failure event is $F = \cup_i F_i$ (serial system) or $F = \cap_i F_i$ (parallel system).

The sampling density follows Melchers & Beck (Eq. 5.26):

$$h(u) = \sum_{i=1}^{m} \alpha_i \, \phi(u - u_i^*)$$

where u_i^* are the FORM design points in standard normal space and α_i are mixture weights.

By default, mixture weights are proportional to individual failure probabilities:

$$\alpha_i \propto P(F_i).$$

If `adaptive_alpha = TRUE`, mixture weights are updated iteratively based on weighted failure frequency estimates:

$$\alpha_i^{new} = (1 - \lambda)\alpha_i^{old} + \lambda \frac{\hat{P}(F_i)}{\sum_k \hat{P}(F_k)}$$

where λ is given by `alpha_update_rate`.

The failure probability is estimated using a self-normalized IS estimator:

$$\hat{P}_f = \frac{\sum I(u) w(u)}{\sum w(u)}$$

with likelihood ratio

$$w(u) = \frac{\phi(u)}{h(u)}.$$

Numerical stability is ensured via global log-weight stabilization.

Stopping Criterion

The simulation terminates when both:

- The estimated coefficient of variation (CoV) falls below `cov_user`, and
- The effective sample size (ESS) exceeds a minimum threshold.

For parallel systems with very small failure probabilities, the ESS-based safeguard prevents premature termination due to unstable variance estimates.

Numerical stability

The default `"robust"` mode performs global log-domain accumulation of importance sampling weights using a log-sum-exp formulation. This avoids overflow and underflow effects in high-dimensional or rare-event settings.

The alternative `"fast"` mode uses shifted exponentiation at the batch level and is computationally equivalent for most engineering reliability problems.

Automatic detection of vectorized limit-state functions

For performance reasons, the algorithm automatically checks whether the supplied limit-state function (LSF) supports vectorized evaluation.

A short internal test is performed before the simulation starts. If the LSF accepts a matrix of input samples and returns a numeric vector of matching length, it is evaluated in fully vectorized form:

$$g(X_{1:n}) \rightarrow \{g(x_1), \ldots, g(x_n)\}$$

Otherwise, the algorithm falls back to row-wise evaluation using `apply()`.

This mechanism is fully automatic and backward compatible. Users do not need to modify existing scalar LSF definitions.

For computationally expensive LSFs (e.g., nonlinear models or surrogate FEM models), vectorized evaluation can significantly improve performance.

Parallel execution and future plan handling

When `backend = "future"` is used, the function respects the currently active **future** plan.

If no parallel plan has been set by the user (i.e., the active plan is sequential), a temporary plan is created internally using:

- `multicore` on Unix-like systems,
- `multisession` on Windows.

The original plan is automatically restored after completion of the simulation.

If the user has already defined a parallel strategy via `future::plan()`, it will not be modified.

This design ensures CRAN compliance while preserving full flexibility for advanced users.

Reproducibility

Reproducible results across different numbers of cores are ensured through:

- L'Ecuyer-CMRG random number streams,
- deterministic master sampling,
- worker-invariant parallel chunking,
- controlled seed propagation.

If `seed` is supplied, results are reproducible regardless of the number of worker threads.

Value

MC_IS returns an object containing the following elements:

- `method` - method identifier
- `pf` - estimated failure probability
- `beta` - reliability index $\beta = -\Phi^{-1}(P_f)$
- `var` - variance estimate
- `cov_mc` - coefficient of variation
- `n_mc` - number of samples used
- `ESS` - effective sample size
- `ESS_modes` - effective sample size per mixture component (system case)
- `data` - optional iteration history (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`.

This form is fully compatible with vectorized evaluation in Monte Carlo and importance sampling algorithms.

`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. Such usage may cause inconsistent behaviour between FORM-based and Monte Carlo-based reliability methods.

The vector form `function(x)` is recommended for best performance and consistency.

Windows users

On Windows systems, it is recommended to use `backend = "future"` for multi-core parallelization. The `"parallel"` backend does not support forking on Windows.

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.

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	confidence level
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 Markov chains (subsets).
- `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 MVFOSM method linearises the limit-state function at the mean values of the basic random variables and estimates the reliability index and probability of failure based on first-order moment information.

This classical approach provides a computationally efficient approximation but may be inaccurate for strongly nonlinear limit-state functions.

Usage

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

Arguments

lsf	LSF Definition, can be Expression or Function. Defined by the FLAG isExpression (see below)
lDistr	List of Distributions
h	If isExpression is False, than Finite Difference Method is used for partial deviation. h is the Windowsize
isExpression	Boolean, If TRUE lsf has to be typeof expression, otherwise lsf has to be type of function()
debug.level	If 0 no additional info if 2 high output during calculation

Value

MVFOSM returns an object containing the following elements:

- beta: Estimated reliability index.
- pf: Estimated probability of failure.
- design.point: Design point in the original x-space.
- alphas: Direction cosines (importance factors) in the standard normal space.
- runtime: Total runtime of the algorithm.

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.

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)

Examples

```
pvar <- PARAM_BASEVAR(
  Name = "E_mod",
  DistributionType = "norm",
  ParamType = "Mean",
  ParamValues = c(30e3, 35e3, 40e3)
)
pvar$nextParam() # sets Mean = 30.000 and prepares the distribution
pvar$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)

Examples

```
pdet <- PARAM_DETVAR(
  Name = "E_mod",
  ParamValues = c(30e3, 35e3, 40e3)
)
pdet$nextParam() # sets Mean = 30.000, Sd = Mean/1e7 and prepares
pdet$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 ‘"brms"‘).

`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

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

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 = Mean / 1e7‘).

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)

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 helper method ‘getMethodLevel()‘ returns an integer that indicates the *complexity* of the method (1 = first-order, 2 = second-order, 3 = Monte-Carlo).

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)

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)
)
```

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.

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.

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 ob distribiutions regarding the distribution object 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 http://www.codes.arizona.edu/toolbox/

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)

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.

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", vars = vars)
lsf$ExpressionToFunction()
```

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",
  ParamType = "Mean",
  ParamValues = c(30e3, 35e3, 40e3)
)
pv2 <- PARAM_BASEVAR(
  Name = "f_ck",
  DistributionType = "norm",
  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. Parameters: empirical approximation $k \approx \left(\frac{\sigma}{\mu}\right)^{-1.086}$, $\lambda = \frac{\mu}{\Gamma\left(1+\frac{1}{k}\right)}$.
- 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. * '$plotGraph(plotType)' - experimental plotting routine for simulation performance.

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 that defines whether the system is *serial* or *parallel* (not yet implemented).

`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 probablity if more than one lsf is given and a system_type (serial or parallel) is set. If calcType is empty (or simpleBounds), only simpleBounds are applied to further calculation of single soultions. If calcType is MCIS, than a Monte Carlo Importance Sampling Method is used (only for parallel systems available). If calcType is MCC, than a Crude Monte Carlo Simulation is used. If calcType is MCSUS, than the Subset Sampling Algorithm ll be used. You can pass arguments to methods via the params field, while the argument has to be a named list (for example check the vignette).

`plotGraph(plotType = "sim.performance")` not finally implemented. Do not use.

`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