Package: minimaxALT (via r-universe)

October 30, 2026

Type Package

Title Generate Optimal Designs of Accelerated Life Test using PSO-Based Algorithm

Version 1.0.4

Encoding UTF-8

License GPL (>= 3)

Description A computationally efficient solution for generating optimal experimental designs in Accelerated Life Testing (ALT). Leveraging a Particle Swarm Optimization (PSO)-based hybrid algorithm, the package identifies optimal test plans that minimize estimation variance under specified failure models and stress profiles. For more detailed, see Lee et al. (2025), Optimal Robust Strategies for Accelerated Life Tests and Fatigue Testing of Polymer Composite Materials <doi:10.1214/25-AOAS2075>.

SystemRequirements GNU Scientific Library (GSL), OpenMP

Imports Rcpp (>= 1.0.11), ggplot2 (>= 3.0.0), parallel (>= 4.0.0), stats, graphics

Depends R (>= 4.0.0)

LinkingTo Rcpp (>= 1.0.11), RcppArmadillo (>= 14.0.0.1), RcppGSL (>= 0.3.13)

RoxygenNote 7.3.2

URL https://github.com/hoanglinh171/minimaxALT

BugReports https://github.com/hoanglinh171/minimaxALT/issues

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation yes

Author Hoai-Linh Hoang [aut, cre], I-Chen Lee [aut], Ping-Yang Chen [aut], Ray-Bing Chen [aut], Weng Kee Wong [aut]

Maintainer Hoai-Linh Hoang <hoailinh.hoang17@gmail.com>

VignetteBuilder knitr

Config/pak/sysreqs libgsl0-dev

Repository https://hoanglinh171.r-universe.dev

Date/Publication 2026-08-31 14:58:25 UTC

RemoteUrl https://github.com/hoanglinh171/minimaxalt

RemoteRef HEAD

RemoteSha dd9b7b761aacb88f48e35d2dcd4cd803e8c9a87c

Contents

check_optimality . . . 2
extract_design . . . 4
find_optimal_alt . . . 5
initialize_values . . . 7
plot.OptimalALT . . . 8
print.DesignInfo . . . 9
print.InitialValue . . . 9
print.OptimalALT . . . 10
print.OptimalityCheck . . . 11
print.PSOInfo . . . 11
pso_setting . . . 12
set_design_info . . . 13
summary.OptimalALT . . . 14
update_optimality_check . . . 15

Index **17**

`check_optimality` *Check Optimality for Optimal Design*

Description

Evaluates whether a design is optimal by the equivalence theorem.

Usage

```
check_optimality(best_design, model_set, design_info, seed = 42)
```

Arguments

`best_design`	A matrix containing stress levels and allocated proportion of the design, each row corresponds to stress levels of each factor, while the last row is their proportions.
`model_set`	A matrix of models, including parameters and distribution, that maximize the optimality criteria with the given best particle's position, each columns corresponds to model coefficients, while the last column is lifetime distribution.

design_info	A 'DesignInfo' object created by function set_design_info.
seed	Seed for reproducibility

Value

An OptimalityCheck object containing optimality check results

References

1. Müller, C. H., & Pázman, A. (1998). Applications of necessary and sufficient conditions for maximin efficient designs. Metrika, 48, 1–19.
2. Huang, M.-N. L., & Lin, C.-S. (2006). Minimax and maximin efficient designs for estimating the location-shift parameter of parallel models with dual responses. Journal of Multivariate Analysis, 97(1), 198–210.

Examples

```
design_info <- set_design_info(
    k_levels=2,
    j_factor=1,
    n_unit=300,
    censor_time=183,
    p=0.1,
    use_cond=0,
    sigma=0.6
)

best_design <- rbind(
  c(0.682, 1),
  c(0.706, 0.294)
)

model_set <- rbind(
  c(0.01, 0.9, "weibull"),
  c(0.01, 0.99, "lognormal")
)

equi <- check_optimality (
    best_design = best_design,
    model_set = model_set,
    design_info = design_info
)

print(equi)
```

`extract_design` *Extract Optimal Design*

Description

Extract design from an OptimalALT object

Usage

```
extract_design(optimal_alt)
```

Arguments

`optimal_alt`	An object of class 'OptimalALT'.

Value

A matrix containing stress levels and allocated proportion of the design, each row corresponds to stress levels of each factor, while the last row is their proportions.

Examples

```
design_info <- set_design_info(
    k_levels=2,
    j_factor=1,
    n_unit=300,
    censor_time=183,
    p=0.1,
    use_cond=0,
    sigma=0.6
)

pso_info <- pso_setting(
    n_swarm=5,
    max_iter=10,
    early_stopping=5,
    tol=0.0001
)

example1_locally <- find_optimal_alt(
    design_type="locally",
    distribution="weibull",
    design_info=design_info,
    pso_info=pso_info,
    coef = c(0.001, 0.9),
    highest_level=TRUE,
    verbose=TRUE,
    n_threads = 1)

opt_design <- extract_design(example1_locally)
```

`find_optimal_alt` *Find Optimal ALT Design Using Hybrid Algorithm*

Description

Runs hybrid algorithm combining PSO and Nelder-Mead to find the optimal design of accelerated life test (ALT).

Usage

```
find_optimal_alt(
  design_type,
  distribution,
  design_info,
  pso_info,
  coef = NULL,
  coef_lower = NULL,
  coef_upper = NULL,
  init_values = NULL,
  highest_level = FALSE,
  n_threads = 1,
  verbose = TRUE,
  seed = 42
)
```

Arguments

`design_type`	Character. One of c("locally", "minimax").
`distribution`	Character. Failure distribution, one of c("weibull, "lognormal").
`design_info`	A 'DesignInfo' object from `set_design_info` containing design specifications.
`pso_info`	A 'PSOInfo' object from `pso_setting` defining PSO hyperparameters.
`coef`	Optional. Fixed model coefficients. Required if `design_type = "locally"`.
`coef_lower`	Optional. Lower bounds for model parameters. Required if `design_type = "minimax"`.
`coef_upper`	Optional. Upper bounds for model parameters. Required if `design_type = "minimax"`.
`init_values`	Optional. An 'InitialValue' object of initial values from `initialize_values`.
`highest_level`	Logical. Whether the highest stress level of the generated design is the upper bound of stress range `x_h`. Default value is `FALSE`.
`n_threads`	Integer. Number of threads for parallel processing.
`verbose`	Logical. If `TRUE`, print optimization progress.
`seed`	Integer. Seed for reproducibility

Value

An object of class OptimalALT

References

1. Chen P (2024). _globpso: Particle Swarm Optimization Algorithms and Differential Evolution for Minimization Problems_. R package version 1.2.1, <https://github.com/PingYangChen/globpso>.
2. Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In Proceedings of the IEEE International Conference on Neural Networks (ICNN) (Vol. 4, pp. 1942–1948).
3. Lee, I. C., Chen, R. B., Wong, W. K., (in press). Optimal Robust Strategies for Accelerated Life Tests and Fatigue Testing of Polymer Composite Materials. Annals of Applied Statistics. <https://imstat.org/journals-and-publications/annals-of-applied-statistics/annals-of-applied-statistics-next-issues/>
4. Meeker, W. Q., & Escobar, L. A. (1998). Statistical methods for reliability data. New York: Wiley-Interscience.
5. Nelder, J. A. and Mead, R. (1965). A simplex algorithm for function minimization. Computer Journal, 7, 308–313. 10.1093/comjnl/7.4.308.

Examples

```
design_info <- set_design_info(
    k_levels=3,
    j_factor=1,
    n_unit=300,
    censor_time=183,
    p=0.1,
    use_cond=0,
    sigma=0.6
)

pso_info <- pso_setting(
    n_swarm=5,
    max_iter=10,
    early_stopping=5,
    tol=0.00001
)

res <- find_optimal_alt(
    design_type="minimax",
    distribution="lognormal",
    design_info=design_info,
    pso_info=pso_info,
    coef_lower=c(10^-6, 0.7),
    coef_upper=c(10^-3, 0.99),
    highest_level = TRUE,
    verbose = FALSE,
    n_threads = 1
)

print(res)
```

```
summary(res)
plot(res, x_l=0, x_h=1)
```

`initialize_values` *Initialize Particle Swarm Optimization and Nelder-Mead Algorithm Values*

Description

Sets initial particles for PSO, initial locally optimal design, and initial parameters for Nelder-Mead algorithm.

Usage

```
initialize_values(init_swarm = NULL, init_local = NULL, init_coef_mat = NULL)
```

Arguments

`init_swarm`	Optional matrix of initial particle positions. If not defined, particle positions are randomly generated using `runif` with pre-determined number of particles `n_swarm` and particle size.
`init_local`	Optional vector of initial locally optimal design for Nelder-Mead optimization. If not defined, the initial vector representing locally optimal design is `c(1, 0.6, 0.3)`.
`init_coef_mat`	Optional matrix of initial parameters to implement multi-start Nelder-Mead algorithm. The number of rows is the number of starts, and each row is the corresponding initial parameters. If not defined, the initial matrix of parameters is generated by sigmoid transformation of `10 * as.matrix(expand.grid(rep(list(c(1, -1)), j_factor + 1)))`.

Value

An 'InitialValue' object of initialized values.

Examples

```
init_local <- c(1, 0.6, 0.3)

init_coef_mat <- rbind(
  c(1e-6, 0.99),
  c(1e-2, 1),
  c(1.01e-6, 0.9999)
)

j_factor <- 1
k_levels <- 3
n_swarm <- 32
```

```
d_swarm <- (j_factor + 1) * k_levels - 1
init_swarm <- matrix(runif(n_swarm*d_swarm), nrow=n_swarm, byrow=TRUE)

init_values <- initialize_values(
    init_swarm=init_swarm,
    init_local=init_local,
    init_coef_mat=init_coef_mat
)
```

`plot.OptimalALT` *Plot an OptimalALT Object*

Description

Plot the verification of design optimality.

Usage

```
## S3 method for class 'OptimalALT'
plot(x, x_l = 0, x_h = 1, nlevels = 10, ...)
```

Arguments

`x`	An object of class 'OptimalALT', typically returned by `minimax_alt()`.
`x_l`	Numeric. Lower bound of the stress range. Default is `0`.
`x_h`	Numeric. Upper bound of the stress range. Default is `1`.
`nlevels`	Integer. Number of grid levels used for plotting the optimality check for a two-factor design. Default is `10`.
`...`	Additional arguments.

Value

Invisibly returns the original 'OptimalALT' object.

Examples

```
## Not run:
# Suppose `result` is an OptimalALT object returned by find_optimal_alt().
plot(result)

# Specify the stress range explicitly.
plot(result, x_l = 0, x_h = 1)

# For a two-factor design, increase the plotting grid resolution.
plot(result, x_l = 0, x_h = 1, nlevels = 20)

## End(Not run)
```

print.DesignInfo *Print a DesignInfo Object*

Description

Prints the design specifications stored in a `DesignInfo` object.

Usage

```
## S3 method for class 'DesignInfo'
print(x, ...)
```

Arguments

`x`	An object of class `DesignInfo`, typically returned by `set_design_info`.
`...`	Additional arguments.

Value

Invisibly returns the original `DesignInfo` object.

Examples

```
## Not run:
design_info <- set_design_info(...)
print(design_info)

## End(Not run)
```

print.InitialValue *Print an InitialValue Object*

Description

Prints a concise summary of the initial values used for the PSO and Nelder-Mead optimization procedures.

Usage

```
## S3 method for class 'InitialValue'
print(x, ...)
```

Arguments

`x`	An object of class `InitialValue`, typically created by `initialize_values`.
`...`	Additional arguments.

Value

Invisibly returns the `InitialValue` object.

Examples

```
## Not run:
init_values <- initialize_values(
    init_swarm = init_swarm,
    init_local = init_local,
    init_coef = init_coef
)

print(init_values)

## End(Not run)
```

`print.OptimalALT` *Print an OptimalALT Object*

Description

Prints a summary of the optimization results and optimality check for an `OptimalALT` object.

Usage

```
## S3 method for class 'OptimalALT'
print(x, ...)
```

Arguments

`x`	An object of class `OptimalALT`, typically returned by `find_optimal_alt`.
`...`	Additional arguments.

Value

Invisibly returns the original `OptimalALT` object.

Examples

```
## Not run:
result <- find_optimal_alt(...)
print(result)

## End(Not run)
```

print.OptimalityCheck *Print an OptimalityCheck Object*

Description

Prints the results of an optimality check for an accelerated life test (ALT) design.

Usage

```
## S3 method for class 'OptimalityCheck'
print(x, show_candidates = FALSE, ...)
```

Arguments

x	An object of class OptimalityCheck, typically returned by check_optimality.
show_candidates	Logical. If TRUE, the candidate model set used in the optimality check is printed. Default is FALSE.
...	Additional arguments.

Value

Invisibly returns the original OptimalityCheck object.

Examples

```
## Not run:
optimality <- check_optimality(...)
print(optimality)

## End(Not run)
```

print.PSOInfo *Print a PSOInfo Object*

Description

Prints the particle swarm optimization (PSO) hyperparameters stored in a PSOInfo object.

Usage

```
## S3 method for class 'PSOInfo'
print(x, ...)
```

Arguments

x	An object of class PSOInfo, typically returned by pso_setting.
...	Additional arguments.

Value

Invisibly returns the original PSOInfo object.

Examples

```
## Not run:
pso_info <- pso_setting(...)
print(pso_info)

## End(Not run)
```

---

pso_setting *Set PSO Optimization Settings*

Description

Define hyperparameters for particle swarm optimization (PSO).

Usage

```
pso_setting(
  n_swarm = 32,
  max_iter = 128,
  early_stopping = 10,
  tol = 0.01,
  c1 = 2.05,
  c2 = 2.05,
  w0 = 1.2,
  w1 = 0.2,
  w_var = 0.8,
  vk = 4
)
```

Arguments

n_swarm	Integer. Number of particles in the swarm.
max_iter	Integer. Maximum number of iterations.
early_stopping	Integer. The frequency, i.e. number of iterations, of validating the design optimality using equivalence theorem. The optimization process stops once maximum directional derivative is approximately 1.

tol	Numeric. Convergence tolerance. The algorithm stops if abs(max_directional_derivative - 1) < tol.
c1	Numeric. Cognitive acceleration coefficient. Default value is 2.05.
c2	Numeric. Social acceleration coefficient. Default value is 2.05.
w0	Numeric. Starting inertia weight. Default value is 1.2.
w1	Numeric. Ending inertia weight. Default value is 0.2.
w_var	Numeric. A number between $[0, 1]$ that controls the percentage of iterations during which PSO linearly decrease inertia weight from w0 to w1. Default value is 0.8.
vk	Numeric. Velocity clamping factor. Default value is 4.

Value

A 'PSOInfo' object of PSO hyperparameters.

Examples

```
pso_info <- pso_setting(n_swarm=32, max_iter=128, early_stopping=10, tol=0.01)
```

---

set_design_info *Set ALT Design Information*

Description

Configures the settings for an accelerated life test.

Usage

```
set_design_info(
  k_levels,
  j_factor,
  n_unit,
  censor_time,
  p,
  use_cond,
  sigma,
  x_l = 0,
  x_h = 1,
  reparam = TRUE
)
```

Arguments

k_levels	Integer. Number of stress levels.
j_factor	Integer. Number of stress factors.
n_unit	Integer. Total number of test units.
censor_time	Numeric. Test duration or censoring time.
p	Numeric. $0 < p < 1$. Lifetime percentile to be estimated at the use condition, i.e. stress levels are 0.
use_cond	Vector. Stress levels at the use condition.
sigma	Numeric. Scale parameter of the lifetime distribution.
x_l	Numeric. Lower bound of stress range. Default is 0.
x_h	Numeric. Upper bound of stress range. Default is 1.
reparam	Logical. Whether reparameterization is applied to model parameters. Reparameterization is supported for all design types, while non-reparameterization is only available for locally optimal design `design_type = "locally"`. Default is TRUE.

Value

A 'DesignInfo' object of design specifications

Examples

```
design_info <- set_design_info(k_levels=3, j_factor=1, n_unit=300,
                               censor_time=183, p=0.1, use_cond=c(0), sigma=0.6)
```

---

summary.OptimalALT *Summarize an OptimalALT Object*

Description

Provides a concise summary of the generated optimal accelerated life test (ALT) design and its optimality-check results.

Usage

```
## S3 method for class 'OptimalALT'
summary(object, ...)
```

Arguments

object	An object of class `OptimalALT`, typically returned by `find_optimal_alt`.
...	Additional arguments.

Value

Invisibly returns the original `OptimalALT` object.

Examples

```
## Not run:
result <- find_optimal_alt(...)
summary(result)

## End(Not run)
```

`update_optimality_check` *Update Optimality Check for Optimal Design*

Description

Update optimality check for an OptimalALT object

Usage

```
update_optimality_check(design, check)
```

Arguments

`design`	An object of class 'OptimalALT'
`check`	An object of class 'OptimalityCheck'

Value

An 'OptimalityALT' object with updated optimality check information

Examples

```
design_info <- set_design_info(
    k_levels=3,
    j_factor=1,
    n_unit=300,
    censor_time=183,
    p=0.1,
    use_cond=0,
    sigma=0.6,
    x_l = 0.1,
    x_h = 1
)

pso_info <- pso_setting(
    n_swarm=5,
```

```
    max_iter=10,
    early_stopping=5,
    tol=0.0001
)

example1_lnorm_minimax <- find_optimal_alt(
    design_type="minimax",
    distribution="lognormal",
    design_info=design_info,
    pso_info=pso_info,
    coef_lower=c(10^-6, 0.7),
    coef_upper=c(10^-4, 0.9),
    highest_level = TRUE,
    verbose=TRUE,
    n_threads = 1)

summary(example1_lnorm_minimax)

opt_design <- extract_design(example1_lnorm_minimax)

model_set <- rbind(
    c(10^-6, 0.99, "lognormal"),
    c(10^-4, 0.99, "lognormal")
)

equi <- check_optimality (
    best_design=opt_design,
    model_set=model_set,
    design_info=design_info
)

opt_check_update_design <- update_optimality_check(example1_lnorm_minimax, equi)

summary(opt_check_update_design)
```

Index

check_optimality, 2, *11*

extract_design, 4

find_optimal_alt, 5, *10*, *14*

initialize_values, *5*, 7, *9*

plot.OptimalALT, 8
print.DesignInfo, 9
print.InitialValue, 9
print.OptimalALT, 10
print.OptimalityCheck, 11
print.PSOInfo, 11
pso_setting, *5*, 12, *12*

set_design_info, *3*, *5*, *9*, 13
summary.OptimalALT, 14

update_optimality_check, 15