Package: kofn (via r-universe)

June 19, 2026

Title Maximum Likelihood Estimation for k-Out-of-n System Data

Version 0.4.0

Description Maximum likelihood estimation of component lifetime parameters from system-level observations of k-out-of-n systems. Supports exponential and Weibull component distributions under multiple observation schemes: Scheme 0 (system lifetime only), Scheme 1 (periodic inspection), and Scheme 2 (complete monitoring). Provides an EM algorithm for Weibull parallel systems and Fisher information comparison across schemes. The k-out-of-n framework unifies series (k=1) and parallel (k=m) systems as a censoring problem on component lifetimes. Conforms to the 'likelihood.model' generics and returns fitted objects compatible with 'algebraic.mle'. The data-generating process and topology infrastructure (system survival, density, signature, structure function, importance measures) are delegated to the 'dist.structure' package; 'kofn' focuses exclusively on inference for the k-out-of-n family.

License MIT + file LICENSE

URL https://github.com/queelius/kofn, https://queelius.github.io/kofn/

BugReports https://github.com/queelius/kofn/issues

Depends R (>= 4.1.0)

Imports stats, numDeriv, likelihood.model, compositional.mle, generics, dist.structure, algebraic.dist, flexhaz (>= 0.5.2)

Suggests algebraic.mle, maskedcauses, testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

VignetteBuilder knitr

NeedsCompilation no

Author Alexander Towell [aut, cre] (ORCID: <https://orcid.org/0000-0001-6443-9897>)

Maintainer Alexander Towell <lex@metafunctor.com>

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

Date/Publication 2026-06-19 12:00:29 UTC

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

RemoteRef HEAD

RemoteSha 74039558aa6fb5ceb59730184f6183d822c78f25

Contents

`assumptions.exp_kofn` *Assumptions for exponential k-out-of-n model*

Description

Returns a character vector listing the assumptions made by the exponential k-out-of-n likelihood model.

Usage

```
## S3 method for class 'exp_kofn'
assumptions(model, ...)
```

Arguments

`model`	An `exp_kofn` object created by `kofn()`.
`...`	Additional arguments (ignored).

Value

Character vector of assumptions.

Examples

```
model <- kofn(k = 3, m = 3, component = dfr_exponential())
assumptions(model)
```

`assumptions.wei_kofn` *Assumptions for Weibull k-out-of-n system model*

Description

Returns a list of assumptions made by the model.

Usage

```
## S3 method for class 'wei_kofn'
assumptions(model, ...)
```

Arguments

`model`	A `wei_kofn` object.
`...`	Additional arguments (currently unused).

Value

A character vector of assumptions.

Examples

```
model <- kofn(k = 2, m = 2, component = dfr_weibull())
assumptions(model)
```

`compare_fisher_info` *Compare Fisher information across observation schemes*

Description

For a given parameter configuration, computes the observed Fisher information under Scheme 0 (system only), Scheme 1 (periodic inspection), and Scheme 2 (complete component data) via simulation.

Usage

```
compare_fisher_info(
  shapes = NULL,
  scales = NULL,
  rates = NULL,
  n = 200L,
  delta = 1,
  n_rep = 50L,
  component = dfr_exponential()
)
```

Arguments

`shapes`	Numeric vector. Weibull shape parameters (Weibull only, or `NULL` for exponential).
`scales`	Numeric vector. Weibull scale parameters (Weibull only, or `NULL` for exponential).
`rates`	Numeric vector. Exponential rate parameters (exponential only, alternative to `shapes`/`scales`).
`n`	Integer. Sample size per replicate.
`delta`	Numeric scalar. Inspection interval width for Scheme 1.
`n_rep`	Integer. Number of Monte Carlo replicates.
`component`	A `dfr_dist` prototype (e.g. `dfr_exponential()` or `dfr_weibull()`) specifying the component family.

Details

The determinant of the observed information matrix is used as a scalar summary. Efficiency ratios indicate relative information content: a ratio less than 1 means the denominator scheme carries more information.

For Scheme 2 (complete data), the Fisher information is computed analytically:

- Exponential: $I_{jj} = n/\lambda_j^2$ (diagonal).
- Weibull: Block-diagonal with per-component 2x2 Fisher information matrices using the standard Weibull FIM formulas.

For Schemes 0 and 1, the observed information is computed numerically via `numDeriv::hessian` of the negative log-likelihood evaluated at the true parameter values.

Value

A list with components:

`scheme0_det` Numeric vector of Scheme 0 information determinants (length `n_rep`).

`scheme1_det` Numeric vector of Scheme 1 information determinants.

`scheme2_det` Numeric vector of Scheme 2 (complete data) information determinants.

`median_det` Named numeric vector of median determinants across schemes.

`efficiency_01` Ratio of median Scheme 0 to Scheme 1 determinant (< 1 means Scheme 1 is more informative).

`efficiency_02` Ratio of median Scheme 0 to Scheme 2 determinant.

`efficiency_12` Ratio of median Scheme 1 to Scheme 2 determinant.

`n_valid` Named integer vector of valid (non-NA) replicate counts per scheme.

Examples

```
set.seed(1)
result <- compare_fisher_info(rates = c(1, 2), n = 50, delta = 1.0, n_rep = 5)
result$median_det
result$efficiency_01  # Scheme 0 vs Scheme 1
```

F_sys_exp *System CDF for exponential parallel systems*

Description

Computes $F_{sys}(t) = \prod_j (1 - e^{-\lambda_j t})$ using the inclusion-exclusion expansion.

Usage

```
F_sys_exp(t, par)
```

Arguments

t	Scalar time point (non-negative numeric).
par	Numeric vector of rates (length m).

Value

Scalar CDF value $P(T_{sys} \leq t)$.

See Also

S_sys_exp() for the survival function.

Examples

```
F_sys_exp(1, c(1, 2))  # P(T_sys <= 1) for 2-component parallel
```

---

fit.exp_kofn *MLE fitting for exponential k-out-of-n model*

Description

Returns a closure function(df, par0, n_starts) that computes the maximum likelihood estimate of the component rate parameters.

Usage

```
## S3 method for class 'exp_kofn'
fit(object, ...)
```

Arguments

object	An exp_kofn object created by kofn().
...	Additional arguments (ignored).

Details

Uses multi-start optimization with L-BFGS-B as primary solver and Nelder-Mead on the log-scale as fallback. Standard errors are computed from the observed Fisher information (negative Hessian) at the MLE.

Value

A function function(df, par0 = NULL, n_starts = 5L) returning a fisher_mle object (from likelihood.model).

Examples

```
model <- kofn(k = 2, m = 2, component = dfr_exponential())
set.seed(42)
df <- rdata(model)(c(1, 2), n = 50)
result <- fit(model)(df)
coef(result)
```

`fit.wei_kofn` *Fit Weibull k-out-of-n system model*

Description

Returns a closure that fits the model to data. Two methods are available:

Usage

```
## S3 method for class 'wei_kofn'
fit(object, ...)
```

Arguments

`object`	A `wei_kofn` object.
`...`	Additional arguments (currently unused).

Details

`"em"` EM algorithm for parallel systems (k = m only). Treats the identity of the last-failing component as latent data. Uses truncated Weibull moments for the E-step and profile optimization over shape with closed-form scale for the M-step.

`"mle"` Direct MLE via L-BFGS-B with Nelder-Mead fallback. Works for any k-out-of-n structure.

The returned solver accepts these additional arguments:

`par0` Initial parameter vector (length 2*m, interleaved shape/scale). If `NULL`, method-of-moments initialization is used.

`n_starts` Number of multi-start attempts (default: 5).

`tol` Convergence tolerance for EM (default: 1e-8).

`maxiter` Maximum EM iterations (default: 2000).

`shape_bounds` Bounds for shape optimization in EM (default: `c(0.05, 15)`).

`verbose` Print iteration progress (default: FALSE).

Value

A function `function(df, par0, ...)` that returns a `fisher_mle` object (from the likelihood.model package).

Examples

```
model <- kofn(k = 2, m = 2, component = dfr_weibull())
set.seed(42)
df <- rdata(model)(c(1.5, 2.0, 2.0, 3.0), n = 50)
result <- fit(model)(df)
coef(result)
```

`fit_scheme1` *Fit k-out-of-n system MLE under Scheme 1 (periodic inspection)*

Description

Returns a closure that fits the model to Scheme 1 data using multi-start optimization with L-BFGS-B and Nelder-Mead fallback.

Usage

```
fit_scheme1(model, ...)
```

Arguments

`model`	A `kofn` model object (exponential or Weibull).
`...`	Additional arguments (currently unused).

Details

The solver uses L-BFGS-B as the primary optimization method with positivity constraints, falling back to Nelder-Mead on the log-parameter scale if L-BFGS-B fails to converge.

Standard errors are computed from the numerical Hessian at the MLE.

Value

A function `function(df, par0 = NULL, n_starts = 5L)` that returns a `fisher_mle` object (from the likelihood.model package).

Examples

```
model <- kofn(k = 2, m = 2, component = dfr_exponential())
set.seed(42)
df <- rdata_scheme1(model)(c(1, 2), n = 50, delta = 1.0)
result <- fit_scheme1(model)(df)
coef(result)
```

hess_loglik.exp_kofn *Hessian of the log-likelihood for exponential k-out-of-n model*

Description

Returns a closure `function(df, par)` that computes the Hessian matrix of the log-likelihood with respect to the rate parameters.

Usage

```
## S3 method for class 'exp_kofn'
hess_loglik(model, ...)
```

Arguments

`model`	An `exp_kofn` object created by `kofn()`.
`...`	Additional arguments (ignored).

Details

Uses numerical differentiation via `numDeriv::hessian()` applied to the log-likelihood closure.

Value

A function `function(df, par)` returning an `m x m` Hessian matrix.

Examples

```
model <- kofn(k = 2, m = 2, component = dfr_exponential())
H <- hess_loglik(model)
set.seed(1)
df <- rdata(model)(c(1, 2), n = 30)
H(df, c(1, 2))  # 2x2 Hessian matrix
```

hess_loglik.wei_kofn *Hessian of log-likelihood for Weibull k-out-of-n system*

Description

Returns a closure that computes the Hessian matrix of the log-likelihood via numerical differentiation using `numDeriv::hessian`.

Usage

```
## S3 method for class 'wei_kofn'
hess_loglik(model, ...)
```

Arguments

model	A wei_kofn object.
...	Additional arguments (currently unused).

Value

A function function(df, par) returning a square matrix (the Hessian of the log-likelihood).

Examples

```
model <- kofn(k = 2, m = 2, component = dfr_weibull())
H <- hess_loglik(model)
set.seed(1)
df <- rdata(model)(c(1.5, 2.0, 2.0, 3.0), n = 30)
H(df, c(1.5, 2.0, 2.0, 3.0))  # 4x4 Hessian matrix
```

---

ie_expand *Inclusion-exclusion expansion of a product of CDFs*

Description

For a set of exponential rates lam, computes the inclusion-exclusion expansion of $\prod_i (1 - e^{-\lambda_i t})$.

Usage

```
ie_expand(lam)
```

Arguments

lam	Numeric vector of positive rates (one per component).

Details

Returns sign and rate-sum vectors such that:

$$\prod_i (1 - e^{-\lambda_i t}) = \sum_k \text{sign}_k \cdot e^{-\text{rate\_sum}_k \cdot t}$$

The number of terms is 2^m where m is length(lam).

Value

A list with elements:

sign Integer vector of signs ($+1$ or -1).

rate_sum Numeric vector of summed rates for each term.

Examples

```
ie <- ie_expand(c(1, 2))
# Product: (1 - exp(-t)) * (1 - exp(-2t))
# = 1 - exp(-t) - exp(-2t) + exp(-3t)
ie$sign      # c(1, -1, -1, 1)
ie$rate_sum  # c(0, 1, 2, 3)
```

---

is_kofn *Test if an object is a kofn model*

Description

Test if an object is a kofn model

Usage

```
is_kofn(x)
```

Arguments

x	An object to test.

Value

Logical indicating whether `x` inherits from `"kofn"`.

Examples

```
is_kofn(kofn(k = 3, m = 3, component = dfr_exponential()))
is_kofn(42)
```

---

kofn *Create a k-out-of-n system estimation model*

Description

Constructs a likelihood model for component lifetime estimation from k-out-of-n system data. The system fails when `k` components have failed: `k=1` is a series system (one failure kills it), `k=m` is a parallel system (all must fail).

Usage

```
kofn(
  k = 1L,
  m = 2L,
  component = dfr_exponential(),
  method = "mle",
  lifetime = "t",
  omega = "omega",
  lifetime_upper = "t_upper"
)
```

Arguments

k	System parameter: system fails when k components have failed. k=1 is series, k=m is parallel.
m	Number of components.
component	A dfr_dist prototype from **flexhaz** specifying the shared component distribution family. Currently supported: dfr_exponential and dfr_weibull. Parameter values on the prototype are ignored; they will be estimated from data.
method	Estimation method: "mle" (direct MLE) or "em" (EM algorithm, Weibull only).
lifetime	Column name for system lifetime (default "t").
omega	Column name for observation type (default "omega").
lifetime_upper	Column name for interval upper bound (default "t_upper").

Details

This model satisfies the likelihood_model concept from the likelihood.model package by providing methods for loglik, score, and hess_loglik.

The class hierarchy is:

- "exp_kofn" or "wei_kofn" (distribution-specific dispatch)
- "kofn" (shared methods)
- "likelihood_model" (generic inference infrastructure)

The concrete subclass is determined by the type of component. The current closed-form machinery is homogeneous: all m components share the same distribution family, and component acts as a prototype for that family.

For non-k-of-n topologies (bridges, arbitrary coherent systems), use coherent_dist or one of the topology shortcuts in dist.structure directly. kofn is exclusively for the k-out-of-n family.

Value

An S3 object of class c("exp_kofn"/"wei_kofn", "kofn", "likelihood_model").

Examples

```
# Parallel system with 3 exponential components (k = m)
model <- kofn(k = 3, m = 3, component = dfr_exponential())
print(model)

# Series system (k = 1)
model_series <- kofn(k = 1, m = 4, component = dfr_exponential())

# Weibull parallel system with EM estimation
model_wei <- kofn(k = 2, m = 2, component = dfr_weibull(), method = "em")
```

`loglik.exp_kofn` *Log-likelihood for exponential k-out-of-n model*

Description

Returns a closure `function(df, par)` that computes the log-likelihood of exponential component rates given system-level data.

Usage

```
## S3 method for class 'exp_kofn'
loglik(model, ...)
```

Arguments

`model`	An `exp_kofn` object created by `kofn()`.
`...`	Additional arguments (ignored).

Details

For parallel systems (`k = m`), the closed-form IE expansion is used, supporting all four observation types: exact, right, left, and interval censored. For general k-out-of-n systems, the per-observation contributions are computed via `dist.structure::exp_kofn(k, par)` and the algebraic.dist generics `density`, `surv`, and `cdf`.

Value

A function `function(df, par)` where `df` is a data frame with columns for lifetime, observation type, and (for interval-censored) upper bounds, and `par` is a numeric vector of `m` component rates.

Examples

```
model <- kofn(k = 3, m = 3, component = dfr_exponential())
ll <- loglik(model)
set.seed(1)
df <- rdata(model)(c(1, 2, 3), n = 30)
ll(df, c(1, 2, 3))
```

`loglik.wei_kofn` *Log-likelihood for Weibull k-out-of-n system*

Description

Returns a closure `function(df, par)` that computes the log-likelihood for a Weibull k-out-of-n system given data and parameters.

Usage

```
## S3 method for class 'wei_kofn'
loglik(model, ...)
```

Arguments

`model`	A `wei_kofn` object.
`...`	Additional arguments (currently unused).

Details

For parallel systems (k = m), uses the direct Weibull system density $f_{sys}(t) = \sum_j f_j(t) \prod_{i \neq j} F_i(t)$. Supports `"exact"` and `"right"` observation types. Left and interval censoring are not supported for Weibull (no IE expansion); use the exponential model or Scheme 1 for those cases.

For general k, delegates per-observation to `dist.structure::wei_kofn(k, shapes, scales)` via the algebraic.dist generics `density`, `surv`, and `cdf`.

Value

A function `function(df, par)` returning a scalar log-likelihood.

Examples

```
model <- kofn(k = 2, m = 2, component = dfr_weibull())
ll <- loglik(model)
set.seed(1)
df <- rdata(model)(c(1.5, 2.0, 2.0, 3.0), n = 30)
ll(df, c(1.5, 2.0, 2.0, 3.0))
```

`loglik_masked` *Masked k-out-of-n log-likelihood*

Description

Computes the log-likelihood for a k-out-of-n system with candidate failed sets (generalized masking). The data frame must contain columns for system lifetime, observation type, and Boolean candidate set indicators (`c1, c2, ..., cm`).

Usage

```
loglik_masked(model, candset = "c", ...)
```

Arguments

`model`	A `kofn` model object.
`candset`	Prefix for candidate set columns (default `"c"`).
`...`	Additional arguments (currently unused).

Details

At system failure, k components have failed. The candidate set `C_i` is a superset of the true failed set. The likelihood marginalizes over all $\binom{|C_i|}{k}$ possible k-subsets.

For series systems (`k = 1`), this reduces to the standard masked cause-of-failure likelihood $\sum_{j \in C_i} w_j(t_i)$. For parallel systems (`k = m`), the candidate set must be `{1, ..., m}` and the likelihood equals the system density.

The function uses the exponential or Weibull distribution functions directly (not the IE expansion), so it works for any family supported by `parse_params`.

Value

A function `function(df, par)` returning a scalar log-likelihood.

Examples

```
# 2-out-of-4 system with candidate failed sets
model <- kofn(k = 2, m = 4, component = dfr_exponential())
ll <- loglik_masked(model)

# Manually construct masked data: k=2 failed, C = {1,2,3}
df <- data.frame(
  t = 1.5, omega = "exact",
  c1 = TRUE, c2 = TRUE, c3 = TRUE, c4 = FALSE
)
ll(df, c(1, 0.8, 0.6, 0.4))
```

`loglik_scheme1` *Log-likelihood for Scheme 1 (periodic inspection) parallel system*

Description

Returns a closure that computes the log-likelihood under Scheme 1 observation. The likelihood combines the system density at the exact system failure time with interval-censored component contributions.

Usage

```
loglik_scheme1(model, ...)
```

Arguments

`model`	A `kofn` model object (exponential or Weibull).
`...`	Additional arguments (currently unused).

Details

The log-likelihood for observation i is:

$$\log L_i(\theta) = \log f_{sys}(t_i) + \sum_j \log[F_j(a_{ij}^+) - F_j(a_{ij}^-)]$$

where f_{sys} is the parallel system density and $[a_{ij}^-, a_{ij}^+)$ is the inspection interval containing component j's failure time.

Value

A function `function(df, par)` returning a scalar log-likelihood.

Examples

```
model <- kofn(k = 2, m = 2, component = dfr_exponential())
ll <- loglik_scheme1(model)
set.seed(1)
df <- rdata_scheme1(model)(c(1, 2), n = 30, delta = 1.0)
ll(df, c(1, 2))
```

ncomponents.kofn *Number of components in a kofn model*

Description

Returns the number of components `m` in the k-out-of-n system.

Usage

```
## S3 method for class 'kofn'
ncomponents(x, ...)
```

Arguments

`x`	A `kofn` model object.
`...`	Additional arguments (ignored).

Value

Integer number of components.

Examples

```
ncomponents(kofn(k = 5, m = 5, component = dfr_exponential()))
```

observe_exact *Exact observation scheme (no censoring)*

Description

Creates an observation functor that records the exact failure time with no censoring. This is the trivial case — included for completeness and for explicit use in Fisher information comparisons.

Usage

```
observe_exact()
```

Value

Observation functor: `function(t_true)` returning a list with `t` (exact time), `omega` (`"exact"`), and `t_upper` (`NA`).

Examples

```
obs <- observe_exact()
obs(42.5)  # list(t = 42.5, omega = "exact", t_upper = NA)
```

observe_interval_censor *Interval-censoring observation scheme*

Description

Creates an observation functor that places all failure times into a fixed window `[a, b)`. Systems failing within the window are interval-censored; systems failing outside are observed exactly.

Usage

```
observe_interval_censor(a, b)
```

Arguments

a	Lower bound of censoring window (non-negative numeric).
b	Upper bound of censoring window (positive numeric, b > a).

Details

For regular grids, use `observe_periodic` instead.

Value

Observation functor: `function(t_true)` returning a list with `t`, `omega` (`"interval"` or `"exact"`), and `t_upper`.

Examples

```
obs <- observe_interval_censor(a = 5, b = 10)
obs(7)   # interval: list(t = 5, omega = "interval", t_upper = 10)
obs(3)   # exact:    list(t = 3, omega = "exact", t_upper = NA)
obs(12)  # exact:    list(t = 12, omega = "exact", t_upper = NA)
```

observe_left_censor *Left-censoring observation scheme*

Description

Creates an observation functor that applies left-censoring at time `tau`. Systems that fail after `tau` are observed exactly; systems that fail before `tau` are left-censored (we only know T < tau).

Usage

```
observe_left_censor(tau)
```

Arguments

tau	Censoring time (positive numeric).

Value

Observation functor: function(t_true) returning a list with t, omega ("left" or "exact"), and t_upper (NA).

Examples

```
obs <- observe_left_censor(tau = 10)
obs(5)   # left:  list(t = 10, omega = "left", t_upper = NA)
obs(15)  # exact: list(t = 15, omega = "exact", t_upper = NA)
```

---

observe_mixture *Mixture of observation schemes*

Description

Creates an observation functor that randomly selects from a set of observation schemes for each observation. This models heterogeneous monitoring environments where different units may be observed under different protocols.

Usage

```
observe_mixture(..., weights = NULL)
```

Arguments

...	Observation functors (created by observe_* functions).
weights	Mixing probabilities (numeric vector). If NULL, uniform weights are used. Weights are normalized to sum to 1.

Value

Observation functor: function(t_true) that randomly selects a constituent scheme and returns its output.

Examples

```
obs <- observe_mixture(
  observe_right_censor(tau = 100),
  observe_periodic(delta = 10, tau = 100),
  weights = c(0.7, 0.3)
)
set.seed(42)
obs(30)
```

observe_periodic *Periodic inspection observation scheme*

Description

Creates an observation functor for periodic inspections at intervals of width `delta`. The failure time is known only to lie within the inspection interval containing it (interval-censored). Systems surviving past `tau` are right-censored.

Usage

```
observe_periodic(delta, tau = Inf)
```

Arguments

`delta`	Inspection interval width (positive numeric).
`tau`	Study end time / right-censoring time (positive numeric or `Inf` for no right-censoring).

Value

Observation functor: `function(t_true)` returning a list with `t` (interval lower bound or `tau`), `omega` (`"interval"` or `"right"`), and `t_upper` (interval upper bound or `NA`).

Examples

```
obs <- observe_periodic(delta = 10, tau = 100)
obs(25)   # interval: list(t = 20, omega = "interval", t_upper = 30)
obs(150)  # right:    list(t = 100, omega = "right", t_upper = NA)
```

observe_right_censor *Right-censoring observation scheme*

Description

Creates an observation functor that applies right-censoring at time `tau`. Systems that fail before `tau` are observed exactly; systems surviving past `tau` are right-censored.

Usage

```
observe_right_censor(tau = Inf)
```

Arguments

`tau`	Censoring time (positive numeric, or `Inf` for no censoring).

Value

Observation functor: `function(t_true)` returning a list with `t`, `omega` (`"exact"` or `"right"`), and `t_upper` (`NA`).

Examples

```
obs <- observe_right_censor(tau = 100)
obs(50)   # exact:  list(t = 50, omega = "exact", t_upper = NA)
obs(150)  # right:  list(t = 100, omega = "right", t_upper = NA)

# No censoring
obs_all <- observe_right_censor(tau = Inf)
obs_all(999)  # exact: list(t = 999, omega = "exact", t_upper = NA)
```

---

print.kofn *Print method for kofn models*

Description

Displays a human-readable summary of the k-out-of-n system model, including system type, component distribution, estimation method, and column name conventions.

Usage

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

Arguments

`x`	A `kofn` model object.
`...`	Additional arguments (ignored).

Value

The model object, invisibly.

Examples

```
print(kofn(k = 3, m = 3, component = dfr_exponential()))
```

`rdata.exp_kofn` *Random data generation for exponential k-out-of-n model*

Description

Returns a closure that generates random system-level data from the exponential k-out-of-n data-generating process.

Usage

```
## S3 method for class 'exp_kofn'
rdata(model, ...)
```

Arguments

`model`	An `exp_kofn` object created by `kofn()`.
`...`	Additional arguments (ignored).

Details

Workflow:

1. Generate i.i.d. exponential component lifetimes.
2. Compute system lifetime as the (m - k + 1)-th order statistic.
3. Apply the observation functor (exact observation by default).

Value

A function `function(theta, n, observe = NULL)` returning a data frame with columns `t` (system lifetime) and `omega` (observation type). Latent component lifetimes, true critical component, and the true parameters are stored as attributes.

Examples

```
model <- kofn(k = 3, m = 3, component = dfr_exponential())
gen <- rdata(model)
set.seed(1)
df <- gen(theta = c(1, 2, 3), n = 20)
head(df)

# With right-censoring
df2 <- gen(c(1, 2, 3), n = 20, observe = observe_right_censor(tau = 2))
table(df2$omega)
```

`rdata.wei_kofn` *Generate random data from a Weibull k-out-of-n system*

Description

Returns a closure that generates system lifetime data with observation scheme support. Component lifetimes are Weibull-distributed with parameters specified by `theta` (interleaved shape/scale).

Usage

```
## S3 method for class 'wei_kofn'
rdata(model, ...)
```

Arguments

`model`	A `wei_kofn` object.
`...`	Additional arguments (currently unused).

Details

The interface matches `rdata.exp_kofn`: the returned closure accepts an `observe` functor for observation schemes (censoring, periodic inspection, etc.). The output includes an `omega` column.

Value

A function `function(theta, n, observe = NULL)` returning a data frame with columns `t` (system lifetimes) and `omega` (observation type). Attributes:

`comp_times` n x m matrix of component lifetimes.

`par` The parameter vector used for generation.

Examples

```
model <- kofn(k = 2, m = 2, component = dfr_weibull())
gen <- rdata(model)
set.seed(42)
df <- gen(theta = c(1.5, 2.0, 2.0, 3.0), n = 50)
head(df)

# With right-censoring
df_cens <- gen(c(1.5, 2.0, 2.0, 3.0), 50,
               observe = observe_right_censor(tau = 3))
table(df_cens$omega)
```

rdata_masked *Generate masked k-out-of-n data*

Description

Returns a closure that generates system lifetime data with candidate failed sets. The true failed set (the k components that failed by T_sys) is always included in the candidate set. Additional non-failed components are included independently with probability p_mask.

Usage

```
rdata_masked(model, candset = "c", ...)
```

Arguments

model	A kofn model object.
candset	Prefix for candidate set columns (default "c").
...	Additional arguments (currently unused).

Value

A function function(theta, n, p_mask = 0, observe = NULL) returning a data frame with columns for system lifetime, observation type, and Boolean candidate set indicators.

Examples

```
model <- kofn(k = 2, m = 4, component = dfr_exponential())
gen <- rdata_masked(model)
set.seed(42)
df <- gen(theta = c(1, 0.8, 0.6, 0.4), n = 10, p_mask = 0.3)
head(df)
```

rdata_scheme1 *Generate Scheme 1 (periodic inspection) data*

Description

Returns a closure that generates k-out-of-n system data under periodic inspection. Each observation yields the exact system failure time and interval-censored component failure times.

Usage

```
rdata_scheme1(model, ...)
```

Arguments

model	A kofn model object (exponential or Weibull).
...	Additional arguments (currently unused).

Details

Note: The Scheme 1 likelihood uses a composite approximation that treats the system density and component interval contributions as independent. This works well for k >= 2 but is unreliable for series systems (k = 1), where surviving components' intervals should be conditioned on survival past T_sys. For series systems, use the maskedcauses package instead.

Value

A function function(theta, n, delta) returning a data frame with columns:

t System failure time (exact).

comp_lower_j Lower bound of inspection interval for component j.

comp_upper_j Upper bound of inspection interval for component j.

The data frame has attributes comp_times (true component times), delta (inspection interval), and par (true parameters).

Examples

```
model <- kofn(k = 2, m = 2, component = dfr_exponential())
gen <- rdata_scheme1(model)
set.seed(1)
df <- gen(theta = c(1, 2), n = 20, delta = 1.0)
head(df)
```

S_sys_exp *System survival function for exponential parallel systems*

Description

Computes $S_{sys}(t) = 1 - F_{sys}(t)$ for a parallel system with exponential components.

Usage

```
S_sys_exp(t, par)
```

Arguments

t	Scalar time point (non-negative numeric).
par	Numeric vector of rates (length m).

Value

Scalar survival probability $P(T_{sys} > t)$.

See Also

`F_sys_exp()` for the CDF.

Examples

```
S_sys_exp(1, c(1, 2))  # P(T_sys > 1) for 2-component parallel
```

`score.exp_kofn` *Score function for exponential k-out-of-n model*

Description

Returns a closure `function(df, par)` that computes the gradient of the log-likelihood with respect to the rate parameters.

Usage

```
## S3 method for class 'exp_kofn'
score(model, ...)
```

Arguments

`model`	An `exp_kofn` object created by `kofn()`.
`...`	Additional arguments (ignored).

Details

Uses numerical differentiation via `numDeriv::grad()` applied to the log-likelihood closure.

Value

A function `function(df, par)` returning a numeric vector of length `m` (the score vector).

Examples

```
model <- kofn(k = 2, m = 2, component = dfr_exponential())
sc <- score(model)
set.seed(1)
df <- rdata(model)(c(1, 2), n = 30)
sc(df, c(1, 2))  # gradient at true parameters
```

score.wei_kofn *Score function for Weibull k-out-of-n system*

Description

Returns a closure that computes the score (gradient of log-likelihood) via numerical differentiation using `numDeriv::grad`.

Usage

```
## S3 method for class 'wei_kofn'
score(model, ...)
```

Arguments

`model`	A `wei_kofn` object.
`...`	Additional arguments (currently unused).

Value

A function `function(df, par)` returning a numeric vector (the gradient of the log-likelihood).

Examples

```
model <- kofn(k = 2, m = 2, component = dfr_weibull())
sc <- score(model)
set.seed(1)
df <- rdata(model)(c(1.5, 2.0, 2.0, 3.0), n = 30)
sc(df, c(1.5, 2.0, 2.0, 3.0))
```

solve_mle *Default MLE solver for positive parameters*

Description

Multi-start optimization via **compositional.mle**: L-BFGS-B with positivity constraints as the primary solver, Nelder-Mead on the log-parameter scale as fallback. Runs from `n_starts` random perturbations of `par0` and returns the best result.

Usage

```
solve_mle(neg_ll, par0, n_par, n_starts = 5L, nobs = NULL)
```

Arguments

neg_ll	Function of par only: guarded negative log-likelihood (returns .Machine$double.xmax / 2 on failure).
par0	Numeric vector of initial parameter values.
n_par	Integer number of parameters.
n_starts	Integer number of random restarts (default 5).
nobs	Integer number of observations.

Value

An mle_numerical result object (from algebraic.mle) with coef(), vcov(), logLik(), etc. Returns NULL if all starts fail.

Examples

```
# Fit a simple exponential rate from positive data
x <- rexp(100, rate = 0.5)
neg_ll <- function(rate) -sum(dexp(x, rate, log = TRUE))
fit <- solve_mle(neg_ll, par0 = 1, n_par = 1, nobs = length(x))
coef(fit)
```

---

trunc_log_moment_vec *Vectorized truncated log-moment of the Weibull distribution*

Description

Computes $E[\log T|T < t_i]$ for $T \sim \text{Weibull}(\alpha, \beta)$ simultaneously for a vector of truncation points.

Usage

```
trunc_log_moment_vec(v_max_vec, alpha, beta)
```

Arguments

v_max_vec	Numeric vector. Precomputed values of $(t_i/\beta)^\alpha$.
alpha	Numeric scalar. Weibull shape parameter.
beta	Numeric scalar. Weibull scale parameter.

Details

Uses the identity:

$$E[\log T|T < t] = \log \beta + \frac{1}{\alpha} E[\log U|U < v_{\max}]$$

where $U \sim \text{Exp}(1)$ and $v_{\max} = (t/\beta)^\alpha$.

The inner expectation $E[\log U|U < v_{\max}]$ is computed via central finite difference of the lower incomplete gamma function at $s = 1$.

For very small $v_{\max}$ (below `1e-10`), the asymptotic approximation $E[\log T|T < t] \approx \log t - 1/\alpha$ is used. For large $v_{\max}$ where numerical issues arise, the untruncated moment $E[\log T] = \log \beta + \psi(1)/\alpha$ is used as fallback, where ψ is the digamma function.

Value

Numeric vector of $E[\log T|T < t_i]$ values, same length as `v_max_vec`.

Examples

```
# E[log T | T < t] for Weibull(shape=2, scale=1) at t = 0.5, 1.0, 2.0
v_max <- (c(0.5, 1.0, 2.0) / 1)^2
trunc_log_moment_vec(v_max_vec = v_max, alpha = 2, beta = 1)
```

`trunc_pow_moment` *Scalar truncated power moment of the Weibull distribution*

Description

Convenience wrapper around `trunc_pow_moment_vec()` for a single truncation point. Computes $E[T^k|T < t]$ for $T \sim \text{Weibull}(\alpha, \beta)$.

Usage

```
trunc_pow_moment(k, t, alpha, beta)
```

Arguments

`k`	Numeric scalar. Power parameter (can be non-integer).
`t`	Numeric scalar. Truncation point (positive).
`alpha`	Numeric scalar. Weibull shape parameter.
`beta`	Numeric scalar. Weibull scale parameter.

Value

Numeric scalar $E[T^k|T < t]$.

See Also

`trunc_pow_moment_vec()` for the vectorized version.

Examples

```
# E[T^1 | T < 2] for Weibull(shape=1.5, scale=1)
trunc_pow_moment(k = 1, t = 2, alpha = 1.5, beta = 1)
```

`trunc_pow_moment_vec` *Vectorized truncated power moment of the Weibull distribution*

Description

Computes $E[T^k|T < t_i]$ for $T \sim \text{Weibull}(\alpha, \beta)$ simultaneously for a vector of truncation points.

Usage

```
trunc_pow_moment_vec(k, v_max_vec, alpha, beta)
```

Arguments

`k`	Numeric scalar. Power parameter (can be non-integer).
`v_max_vec`	Numeric vector. Precomputed values of $(t_i/\beta)^\alpha$.
`alpha`	Numeric scalar. Weibull shape parameter.
`beta`	Numeric scalar. Weibull scale parameter.

Details

Uses the substitution $U = (T/\beta)^\alpha \sim \text{Exp}(1)$, so that $T^k = \beta^k U^{k/\alpha}$ and the truncation $T < t$ becomes $U < v_{\max}$. The result is:

$$E[T^k|T < t] = \beta^k \frac{\gamma(k/\alpha + 1, v_{\max})}{1 - e^{-v_{\max}}}$$

where $\gamma(s, x)$ is the lower incomplete gamma function.

Computation is done in log-space for numerical stability.

For very small $v_{\max}$ (below `1e-12`), the asymptotic approximation $E[T^k|T < t] \approx t^k/(k/\alpha + 1)$ is used to avoid numerical issues.

Value

Numeric vector of $E[T^k|T < t_i]$ values, same length as `v_max_vec`.

Examples

```
# E[T^2 | T < t] for Weibull(shape=2, scale=1) at t = 0.5, 1.0, 2.0
v_max <- (c(0.5, 1.0, 2.0) / 1)^2
trunc_pow_moment_vec(k = 2, v_max_vec = v_max, alpha = 2, beta = 1)
```

`w_j_exact` *Component weight for exponential parallel systems*

Description

Computes the contribution of component j to the parallel system density at time t. For component j with rate λ_j and other rates λ_{-j}:

$$w_j(t) = \lambda_j e^{-\lambda_j t} \prod_{i \neq j} (1 - e^{-\lambda_i t})$$

Usage

```
w_j_exact(t, par, j)
```

Arguments

`t`	Scalar time point (positive numeric).
`par`	Numeric vector of rates (length m), one per component.
`j`	Component index (integer, 1-based).

Details

Uses the inclusion-exclusion expansion to express this as a finite sum of exponentials.

Value

Scalar value of $w_j(t)$.

See Also

`w_j_integral()` for the closed-form integral of w_j.

Examples

```
# Component 1 contribution at t = 0.5, rates = c(1, 2, 3)
w_j_exact(0.5, c(1, 2, 3), j = 1)
```

`w_j_integral` *Closed-form integral of w_j(t) over an interval*

Description

Computes $\int_a^b w_j(t)\,dt$ in closed form using the inclusion-exclusion expansion. Each term integrates as:

$$\int_a^b e^{-rt}\,dt = \frac{e^{-ra} - e^{-rb}}{r}$$

Usage

```
w_j_integral(a, b, par, j)
```

Arguments

`a`	Lower bound of integration (non-negative numeric scalar).
`b`	Upper bound of integration (positive numeric scalar, `b > a`).
`par`	Numeric vector of rates (length m).
`j`	Component index (integer, 1-based).

Value

Scalar integral value.

See Also

`w_j_exact()` for the pointwise evaluation.

Examples

```
# Integral of w_1(t) from 0 to 1, rates = c(1, 2)
w_j_integral(0, 1, c(1, 2), j = 1)
```

Index