Package 'survdistr'

May 9, 2026

Title Survival Distribution Container with Flexible Interpolation Methods

Version 0.0.3

Description Efficient containers for storing and managing prediction outputs from survival models, including Cox proportional hazards, random survival forests, and modern machine learning estimators. Provides fast C++ methods to evaluate survival probabilities, hazards, probability densities, and related quantities at arbitrary time points, with support for multiple interpolation methods via 'Rcpp'.

License MIT + file LICENSE

URL https://survdistr.mlr-org.com

Imports checkmate, R6, Rcpp

Suggests testthat (>= 3.0.0)

LinkingTo Rcpp

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation yes

Author John Zobolas [aut, cre], Andreas Bender [ctb]

Maintainer John Zobolas <bblodfon@gmail.com>

Repository CRAN

Date/Publication 2026-04-09 09:30:15 UTC

Contents

`assert_prob` *Assert probability matrix or vector*

Description

Validates that the input is a proper probability matrix or vector representing either a survival function, cumulative distribution function (CDF), cumulative incidence function (CIF), discrete hazard, or discrete density.

Usage

```
assert_prob(x, times = NULL, type = "surv")
```

Arguments

`x`	(`numeric()`	`matrix()`) Survival vector or matrix (rows = observations, columns = time points).
`times`	(`numeric()`	`NULL`) Original time points. If `NULL`, extracted from names/colnames.
`type`	(`character(1)`) Type of probability function: `"surv"` (default), `"cdf"`, `"cif"`, `"haz"`, or `"dens"`.

Details

The following conditions must hold:

1. The input `x` is a numeric matrix with no missing values.
2. Time points (`times`) are numeric, non-negative, unique, and increasing. If not supplied, they are derived from `(col)names(x)` (coerced to `numeric`).
3. All values are valid probabilities, i.e. lie in $[0, 1]$.
4. Each row is monotone:
   - `"surv"`: non-increasing survival curves, i.e. $S(t_i) \geq S(t_{i+1})$.
   - `"cdf"` / `"cif"`: non-decreasing functions, i.e. $F(t_i) \leq F(t_{i+1})$.
   - `"haz"` / `"dens"`: no monotonicity requirement.
5. Boundary condition at `t = 0`:
   - `"surv"`: $S(0) = 1$.
   - `"cdf"` / `"cif"`: $F(0) = 0$.
   - `"haz"` / `"dens"`: $t_1 > 0$ (otherwise, nonzero hazard/density at `t = 0` implies $S(0) \neq 1$)

Value

Invisibly returns the validated numeric time points.

Examples

```
x = matrix(data = c(1, 0.6, 0.4,
                    0.8, 0.8, 0.7),
           nrow = 2, ncol = 3, byrow = TRUE)

# Explicitly provide time points
assert_prob(x, times = c(12, 34, 42), type = "surv")

# Or use column names as time points
colnames(x) = c(12, 34, 42)
assert_prob(x)

# check CDF
assert_prob(1 - x, type = "cdf")

# check discrete hazards
assert_prob(c(0.2, 0.01, 0.3), times = c(1, 2, 3), type = "haz")

# check discrete densities
assert_prob(c(0.2, 0.01, 0.3), times = c(1, 2, 3), type = "dens")
```

---

as_survDistr *Coerce Object to survDistr*

Description

S3 generic to coerce supported objects into a survDistr object.

Usage

```
as_survDistr(x, ...)

## S3 method for class 'matrix'
as_survDistr(
  x,
  times = NULL,
  method = "const_surv",
  check = TRUE,
  trim_dups = FALSE,
  ...
)

## S3 method for class 'survDistr'
```

```
as_survDistr(x, ...)

## Default S3 method:
as_survDistr(x, ...)
```

Arguments

x	Object to convert.
...	Additional arguments passed to methods.
times	(numeric) Numeric vector of time points corresponding to columns of x. If NULL, column names of x are used.
method	(character(1)) Interpolation method passed to survDistr constructor.
check	(logical(1)) Whether to validate x and times.
trim_dups	(logical(1)) Whether to remove duplicate S(t) values and corresponding time points.

Value

A survDistr object.

---

convert_to_surv *Convert density/hazard to survival*

Description

Converts density or hazards from one of four input representations to survival probabilities at the same anchor time points (no interpolation).

Usage

```
convert_to_surv(
  x,
  times = NULL,
  input = "cont_haz",
  check = TRUE,
  integration = "trapezoid",
  clamp_surv = FALSE,
  eps = 1e-12
)
```

Arguments

`x`	(`numeric()` | `matrix()`) Input vector or matrix (rows = observations, columns = time points).
`times`	(`numeric()` | `NULL`) Anchor time points. If `NULL`, extracted from names/colnames of `x`.
`input`	(`character(1)`) Input type. One of `"disc_haz"`, `"disc_dens"`, `"cont_haz"` or `"cont_dens"`.
`check`	(`logical(1)`) If `TRUE` (default), run *input* validation checks. Disable only if you know the input is valid and want to skip checks for speed.
`integration`	(`character(1)`) Numerical integration rule for continuous inputs: `"trapezoid"` (default) uses the trapezoidal rule, while `"riemann"` is the left Riemann sum. Only used for `"cont_dens"` and `"cont_haz"`.
`clamp_surv`	(`logical(1)`) If `TRUE`, clamp survival probabilities to `[eps, 1]` to avoid numerical issues.
`eps`	(`numeric(1)`) Small value used to clamp near-zero survival probabilities if `clamp_surv = TRUE`.

Details

Let $t_1, \ldots, t_B$ denote the anchor time points, $\Delta_j = t_j - t_{j-1}$, and $S_j = S(t_j)$ the survival probabilities at the anchors. The conversion depends on the value of **input** as follows:

- Discrete densities $\tilde{f}_k$ (`"disc_dens"`):

$$S_j = 1 - \sum_{k=1}^{j} \tilde{f}_k$$

- Discrete hazards $\tilde{h}_k$ (`"disc_haz"`):

$$S_j = \prod_{k=1}^{j} (1 - \tilde{h}_k)$$

- Continuous densities f_k (`"cont_dens"`):
  - Trapezoidal rule:

$$S_j = 1 - \sum_{k=1}^{j} \frac{f_{k-1} + f_k}{2} \Delta_k$$

    (with $f_0 = f_1$)
  - Left Riemann sum:

$$S_j = 1 - \sum_{k=1}^{j} f_k \Delta_k$$

- Continuous hazards λ_k (`"cont_haz"`):

- Trapezoidal rule:

$$S_j = \exp\left(- \sum_{k=1}^{j} \frac{\lambda_{k-1} + \lambda_k}{2} \Delta_k \right)$$

  (with $\lambda_0 = \lambda_1$)
- Left Riemann sum:

$$S_j = \exp\left(- \sum_{k=1}^{j} \lambda_k \Delta_k \right)$$

For continuous inputs (`"cont_dens"` / `"cont_haz"`), numerical integration can be done either with the trapezoidal rule (`integration = "trapezoid"`, default) or with a left Riemann sum (`integration = "riemann"`). Trapezoidal rule is more accurate (lower approximation error in the order of Δ^2 while the Riemann sum has an approximation error in the order of Δ). At the first anchor both rules are identical, because no previous anchor value is available; therefore both use $x_1 \Delta_1$.

Value

A numeric vector or matrix of survival probabilities with the same dimensions as `x`.

Validation

If `check = TRUE`, we validate that the input is a proper discrete density/hazard matrix or vector using `assert_prob()`. For continuous hazards/densities, we only check that the input is a non-negative numeric matrix/vector.

Examples

```
# Continuous hazard => survival
haz_cont = c(0.02, 0.1, 0.2, 0.15)
times = c(0, 1, 2, 3)
convert_to_surv(haz_cont, times = times, input = "cont_haz")

# Discrete hazard => survival
haz_disc = c(0.1, 0.2, 0.15)
times = c(1, 2, 3)
convert_to_surv(haz_disc, times = times, input = "disc_haz")
```

`extract_times` *Extract time points from a probability matrix or vector*

Description

Helper function to consistently obtain and validate the time points associated with a probability matrix or vector.

Usage

```
extract_times(x, times = NULL)
```

Arguments

x	(numeric() | matrix()) Probability vector (length = time points) or matrix (rows = observations, columns = time points).
times	(numeric() | NULL) Optional vector of time points corresponding to x.

Value

A validated numeric vector of time points.

---

interp *Interpolate Survival Curves*

Description

Interpolates survival curves (vector or matrix) at new time points using internal C++ interpolation functions. Output can be survival, cumulative distribution, density, hazard or cumulative hazard functions.

Usage

```
interp(
  x,
  times = NULL,
  eval_times = NULL,
  method = "const_surv",
  output = "surv",
  add_times = TRUE,
  check = TRUE,
  eps = 1e-12,
  trim_dups = FALSE
)
```

Arguments

x	(numeric() | matrix()) Survival vector or matrix (rows = observations, columns = time points).
times	(numeric() | NULL) Anchor time points. If NULL, extracted from names/colnames.
eval_times	(numeric() | NULL) New time points at which to interpolate. Values need to be unique, increasing non-negative numbers. If NULL, the anchor times are used.
method	(character(1)) Interpolation method; one of "const_surv" (default), "const_dens" (alias: "linear_surv") and "const_haz" (alias: "exp_surv").

output (character(1))
Output type: "surv", "cdf", "cumhaz", "density" or "hazard".

add_times (logical(1))
If TRUE, attach eval_times as names/colnames.

check (logical(1))
If TRUE, run input matrix validation checks using assert_prob(); set to FALSE to skip checks (NOT recommended for external use).

eps (numeric(1))
Small positive value used to replace extremely low survival probabilities when computing cumulative hazard, preventing numerical instability in $-\log S(t)$ calculations.

trim_dups (logical(1))
If TRUE, removes adjacent duplicate values from the input using trim_duplicates(). This eliminates flat segments in survival curves and improves interpolation efficiency. Default is FALSE.

Details

Input must always be *survival probabilities*. We currently provide three interpolation options:

- "const_surv": left-continuous constant interpolation of S(t) (default).
- "const_dens"/"linear_surv": linear interpolation of S(t) (equivalent to piecewise constant interpolation of the density function).
- "const_haz"/"exp_surv": exponential interpolation of S(t) (equivalent to piecewise constant interpolation of the hazard function).

We will provide tables with the interpolation formulas for each method in an upcoming arXiv preprint and link it here.

For constant hazard interpolation ("const_haz"), any right-anchor S(t) values equal to 0 are internally floored at min(1e-12, S_left) within each interval. This keeps hazards/densities finite without allowing a local increase in S(t).

Value

A numeric vector or matrix of interpolated values.

Examples

```
x = matrix(c(1, 0.8, 0.6,
             1, 0.7, 0.4), nrow = 2, byrow = TRUE)
times = c(0, 8, 13)
eval_times = c(5, 10, 14)

# constant S(t) interpolation
interp(x, times, eval_times)

# linear S(t) interpolation
interp(x, times, eval_times, method = "linear_surv")
```

```
# exponential S(t) interpolation (same as `method = "const_haz"`)
interp(x, times, eval_times, method = "exp_surv")

# Cumulative distribution with linear S(t) interpolation
interp(x, times, eval_times, method = "linear_surv", output = "cdf")

# H(t) with linear S(t) interpolation
interp(x, times, eval_times, method = "linear_surv", output = "cumhaz")

# f(t) with constant hazard interpolation
interp(x, times, eval_times, method = "const_haz", output = "density")

# h(t) with constant hazard interpolation
interp(x, times, eval_times, method = "const_haz", output = "hazard")
```

`interp_cif` *Interpolate CIF matrix*

Description

Interpolates cumulative incidence (CIF) functions (corresponding to one competing event only) using left-continuous constant interpolation.

Usage

```
interp_cif(x, times = NULL, eval_times = NULL, add_times = TRUE, check = TRUE)
```

Arguments

`x`	(`matrix()`) CIF matrix (rows = observations, columns = time points).
`times`	(`numeric()` | `NULL`) Anchor time points. If `NULL`, extracted from `colnames(x)`.
`eval_times`	(`numeric()` | `NULL`) New time points at which to interpolate. Values need to be unique, increasing non-negative numbers. If `NULL`, the anchor `times` are used.
`add_times`	(`logical(1)`) If `TRUE`, attach `eval_times` as colnames in the output matrix.
`check`	(`logical(1)`) If `TRUE`, run input matrix validation checks using `assert_prob()`; set to `FALSE` to skip checks (NOT recommended for external use).

Value

Interpolated CIF matrix.

`survDistr` *Survival Distribution Container*

Description

survDistr is an R6 specialized container designed for storing and managing prediction outputs from survival models in single-event settings. This includes models such as Cox proportional hazards, random survival forests, and other classical or machine learning-based survival estimators.

The main prediction data type is survival matrix, where **rows represent observations and columns represent time points**.

Details

Key design features:

- The survival matrix is stored internally and can be accessed using the `$data()` method.
- The `$times` active field provides the time points corresponding to the matrix columns.
- The interpolation method is controlled via the `$method` active field.
- Survival-related quantities (e.g., distribution, density, hazard functions) are interpolated using the `interp()` function.
- The `assert_prob()` function validates the input data matrix during construction if `check` is set to `TRUE`.
- Use the `$filter()` method to subset observations in-place by filtering rows of the stored matrix.
- Use `trim_dups = TRUE` in the constructor to remove flat survival segments (repeated values across time points) with a pre-specified tolerance (for a more controlled pre-processing, see `trim_duplicates()`).

Active bindings

`times (numeric)`
: Numeric vector of time points corresponding to columns of `data`. Read-only.

`method (character(1))`
: Interpolation method; one of `"const_surv"` (default), `"const_dens"` (alias: `"linear_surv"`) and `"const_haz"` (alias: `"exp_surv"`).

Methods

Public methods:

- `survDistr$new()`
- `survDistr$print()`
- `survDistr$data()`
- `survDistr$filter()`
- `survDistr$survival()`

- survDistr$distribution()
- survDistr$density()
- survDistr$cumhazard()
- survDistr$hazard()
- survDistr$clone()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
survDistr$new(
  x,
  times = NULL,
  method = "const_surv",
  check = TRUE,
  trim_dups = FALSE
)
```

Arguments:

`x` (`matrix`)
: A numeric matrix of survival probabilities (values between 0 and 1). Column names must correspond to time points if `times` is `NULL`.

`times` (`numeric()`)
: Numeric vector of time points for matrix `x`, must match the number of columns.

`method` (`character(1)`)
: Interpolation method; one of `"const_surv"` (default), `"const_dens"` (alias: `"linear_surv"`) and `"const_haz"` (alias: `"exp_surv"`).

`check` (`logical(1)`)
: If `TRUE`, run input matrix validation checks using `assert_prob()`; set to `FALSE` to skip checks (NOT recommended for external use).

`trim_dups` (`logical(1)`)
: If `TRUE`, removes adjacent duplicate values from the input using `trim_duplicates()`. This eliminates flat segments in survival curves and improves interpolation efficiency. Default is `FALSE`.

Method `print()`**:** Displays summary information about a survDistr object, including the number of observations and time points.

Usage:

```
survDistr$print()
```

Method `data()`**:** Return the stored data matrix.

Usage:

```
survDistr$data(rows = NULL, add_times = TRUE)
```

Arguments:

`rows` (`integer()` | `logical()` | `NULL`)
: Row indices or a logical vector used to filter observations. Logical vectors must have length equal to the number of observations. Integer indices must be positive and within range. If `NULL`, no filtering is applied.

`add_times` (`logical(1)`)
If `TRUE`, attach `eval_times` to the output.

Returns: (`matrix`)

Method `filter()`**:** Filters observations **in-place** by subsetting rows of the stored matrix.

Usage:

```
survDistr$filter(rows = NULL)
```

Arguments:

`rows` (`integer()` | `logical()` | `NULL`)
Row indices or a logical vector used to filter observations. Logical vectors must have length equal to the number of observations. Integer indices must be positive and within range. If `NULL`, no filtering is applied.

Returns: Invisibly returns the `survDistr` object itself.

Method `survival()`**:** Computes survival probabilities $S(t)$ at the specified time points.

Usage:

```
survDistr$survival(rows = NULL, times = NULL, add_times = TRUE)
```

Arguments:

`rows` (`integer()` | `logical()` | `NULL`)
Row indices or a logical vector used to filter observations. Logical vectors must have length equal to the number of observations. Integer indices must be positive and within range. If `NULL`, no filtering is applied.

`times` (`numeric`)
New time points at which to interpolate. Values need to be unique, increasing non-negative numbers. If `NULL`, the object's stored time points are used.

`add_times` (`logical(1)`)
If `TRUE`, attach `eval_times` to the output.

Returns: a `matrix` of survival probabilities (rows = observations, columns = time points).

Method `distribution()`**:** Computes the cumulative distribution function $F(t) = 1 - S(t)$ or CDF at the specified time points. $F(t)$ is the probability that the event has occurred up until time t.

Usage:

```
survDistr$distribution(rows = NULL, times = NULL, add_times = TRUE)
```

Arguments:

`rows` (`integer()` | `logical()` | `NULL`)
Row indices or a logical vector used to filter observations. Logical vectors must have length equal to the number of observations. Integer indices must be positive and within range. If `NULL`, no filtering is applied.

`times` (`numeric`)
New time points at which to interpolate. Values need to be unique, increasing non-negative numbers. If `NULL`, the object's stored time points are used.

`add_times` (`logical(1)`)
If `TRUE`, attach `eval_times` to the output.

Returns: a `matrix` of CDF values (rows = observations, columns = time points).

Method `density()`**:** Computes the probability density function $f(t)$ or PDF at the specified time points. $f(t) = \frac{d}{dt}F(t)$ is the probability of the event occurring at the specific time t.

Usage:

```
survDistr$density(rows = NULL, times = NULL, add_times = TRUE)
```

Arguments:

`rows (integer() | logical() | NULL)`
: Row indices or a logical vector used to filter observations. Logical vectors must have length equal to the number of observations. Integer indices must be positive and within range. If `NULL`, no filtering is applied.

`times (numeric)`
: New time points at which to interpolate. Values need to be unique, increasing non-negative numbers. If `NULL`, the object's stored time points are used.

`add_times (logical(1))`
: If `TRUE`, attach `eval_times` to the output.

Returns: a `matrix` of PDF values (rows = observations, columns = time points).

Method `cumhazard()`**:** Computes the cumulative hazard (accumulated risk up to time t) at the specified time points as $H(t) = -log(S(t))$.

Usage:

```
survDistr$cumhazard(rows = NULL, times = NULL, add_times = TRUE, eps = 1e-12)
```

Arguments:

`rows (integer() | logical() | NULL)`
: Row indices or a logical vector used to filter observations. Logical vectors must have length equal to the number of observations. Integer indices must be positive and within range. If `NULL`, no filtering is applied.

`times (numeric)`
: New time points at which to interpolate. Values need to be unique, increasing non-negative numbers. If `NULL`, the object's stored time points are used.

`add_times (logical(1))`
: If `TRUE`, attach `eval_times` to the output.

`eps (numeric(1))`
: Small positive value used to replace extremely low survival probabilities when computing cumulative hazard, preventing numerical instability in $-\log S(t)$ calculations.

Returns: a `matrix` of cumulative hazards (rows = observations, columns = time points).

Method `hazard()`**:** Computes the hazard $h(t) = \frac{f(t)}{S(t)}$ at the specified time points. Hazard is the conditional instantaneous event rate at time t given survival up to time t.

Usage:

```
survDistr$hazard(rows = NULL, times = NULL, add_times = TRUE)
```

Arguments:

`rows (integer() | logical() | NULL)`
: Row indices or a logical vector used to filter observations. Logical vectors must have length equal to the number of observations. Integer indices must be positive and within range. If `NULL`, no filtering is applied.

`times` (`numeric`)
: New time points at which to interpolate. Values need to be unique, increasing non-negative numbers. If `NULL`, the object's stored time points are used.

`add_times` (`logical(1)`)
: If `TRUE`, attach `eval_times` to the output.

Returns: a `matrix` of hazard values (rows = observations, columns = time points).

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
survDistr$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# generate survival matrix
mat = matrix(data = c(1,0.6,0.4,0.8,0.8,0.7), nrow = 2,
             ncol = 3, byrow = TRUE)
times = c(12, 34, 42)
x = survDistr$new(mat, times)
x

# stored survival matrix
x$data()

# interpolation method
x$method

# time points
x$times

eval_times = c(10, 30, 42, 50)
# S(t) at given time points (constant interpolation)
x$survival(times = eval_times)
# same but with linear interpolation
x$method = "linear_surv"
x$survival(times = eval_times)

# Cumulative hazard
x$cumhazard(times = eval_times)

# Density
x$density(times = eval_times)

# Hazard
x$hazard(times = eval_times)
```

`trim_duplicates` *Remove adjacent duplicate values*

Description

Removes adjacent duplicate values over the time axis, possibly from a probability vector or matrix (e.g. survival curves). Equality is determined with a numeric tolerance.

For matrices, duplicate detection is done column-wise across all rows. Only the earliest time point in each run of (near-)equal values is kept.

Usage

```
trim_duplicates(x, times = NULL, tol = 1e-10)
```

Arguments

`x`	(`numeric()` | `matrix()`) Vector (length = time points) or matrix (rows = observations, columns = time points).
`times`	(`numeric()` | `NULL`) Optional time points corresponding to `x`. If `NULL`, extracted from names/colnames.
`tol`	(`numeric(1)`) Absolute tolerance used to detect equality between adjacent time points.

Value

A named list with:

- `x`: numeric vector or matrix with duplicate adjacent time points removed.
- `times`: numeric vector of retained time points.

Examples

```
# remove adjacent duplicates from a survival probability vector
surv = c(1, 1, 0.8, 0.8, 0.5, 0.5, 0.2)
trim_duplicates(surv, times = 1:7)
```

Index