Package ‘PWEXP’

March 12, 2024

Type Package
Title Piecewise Exponential Distribution Prediction Model
Imports graphics, grDevices, stats, methods, utils, segmented, foreach, doSNOW, parallel
Depends survival, fastmatch
Suggests knitr, RColorBrewer, rmarkdown
Version 0.5.0
Author Tianchen Xu [aut, cre] (<https://orcid.org/0000-0002-0102-7630>)
Maintainer Tianchen Xu <zjph602xutianchen@gmail.com>
Description Build piecewise exponential survival model for study design (planning) and event/timeline prediction.
URL https://github.com/zjph602xtc/PWEXP
BugReports https://github.com/zjph602xtc/PWEXP/issues
License MIT + file LICENSE
Encoding UTF-8
VignetteBuilder knitr
Repository CRAN
Date/Publication 2024-03-12 16:30:07 UTC

R topics documented:

boot.pwexp.fit .................................................. 2
conditional piecewise exponential ................................ 3
cut_dat ......................................................... 5
cv.pwexp.fit ..................................................... 7
piecewise exponential .......................................... 8
plot_event ....................................................... 10
plot_survival ................................................... 13
predict ......................................................... 14
 Bootstrap a Piecewise Exponential Model

Description

Bootstrap a existing piecewise exponential model or build a piecewise exponential model with bootstrapping.

Usage

## Default S3 method:
boot.pwexp.fit(time, event, nsim=100, breakpoint=NULL, nbreak=0, 
  exclude_int=NULL, min_pt_tail=5, max_set=1000, seed=1818, 
  optimizer='mle', tol=1e-4, parallel=FALSE, mc.core=4, ...)

## S3 method for class 'pwexp.fit'
boot.pwexp.fit(time, nsim=100, max_set=1000, seed=1818, 
  optimizer='mle', tol=1e-4, parallel=FALSE, mc.core=4, ...)

Arguments

time 
observed time from randomization or a pwexp.fit object.

event 
the status indicator. See pwexp.fit.

nsim 
the number of repeated bootstrapping.

breakpoint 
pre-specified breakpoints. See pwexp.fit.

nbreak 
total number of breakpoints. See pwexp.fit.

exclude_int 
an interval that excludes any estimated breakpoints. See pwexp.fit.

min_pt_tail 
the minimum number of events used for estimating the tail (the hazard rate of the last piece). See pwexp.fit.

max_set 
maximum estimated combination of breakpoints. See pwexp.fit.

seed 
a random seed.

optimizer 
one of the optimizers: mle, ols, or hybrid. See pwexp.fit.

tol 
the minimum allowed gap between two breakpoints. The gap is calculated as (max(time)-min(time))*tol. Keep it as default in most cases.

parallel 
logical. If TRUE, use doSNOW package to run in parallel.

mc.core 
number of processes allowed to be run in parallel.

... 
internal function reserved.
conditional piecewise exponential

The Conditional Piecewise Exponential Distribution

Description

Distribution function, quantile function and random generation for the piecewise exponential distribution $t$ with piecewise rate $\lambda$ given $t > qT$.
Usage

```r
ppwexp_conditional(q, qT, rate=1, breakpoint=NULL, lower.tail=TRUE,
log.p=FALSE, one_piece, safety_check=TRUE)
qpwexp_conditional(p, qT, rate=1, breakpoint=NULL, lower.tail=TRUE,
log.p=FALSE, one_piece, safety_check=TRUE)
rpwexp Conditional(n, qT, rate, breakpoint=NULL)
```

Arguments

- `q`: vector of quantiles.
- `p`: vector of probabilities.
- `qT`: the distribution is conditional on \( t > qT \). `qT` can be a scalar or a vector with the same length of `q` or `p`.
- `n`: number of observations. Must be a positive integer with length 1.
- `rate`: a vector of rates in each piece.
- `breakpoint`: a vector of breakpoints. The length is `length(rate)-1`. Can be NULL if `rate` is a single value.
- `log, log.p`: logical; if TRUE, probabilities p are given as log(p).
- `lower.tail`: logical; if TRUE (default), probabilities are \( P[X \leq x] \), otherwise, \( P[X > x] \).
- `one_piece`: (only required when `safety_check=FALSE`) whether the distribution only has one piece (i.e., `rate` is a single value and `breakpoint=NULL`).
- `safety_check`: logical; whether check the input arguments; if FALSE, function has better computing performance by skipping all safety checks.

Details

See webpage https://zjph602xtc.github.io/PWEXP/ for more details for its survival function, cumulative density function, quantile function.

Value

- `ppwexp_conditional` gives the conditional distribution function, `qpwexp_conditional` gives the conditional quantile function, and `rpwexp Conditional` generates conditional random variables.

The length of the result is determined by q, p or n for `ppwexp_conditional`, `qpwexp_conditional` or `rpwexp Conditional`. You can only specify a single piecewise exponential distribution every time you call these functions. This is different from the exponential distribution functions in package `stats`.

When the length of `qT` is 1, then all results are conditional on the same \( t > qT \). In `rpwexp Conditional`, `qT` must be a scalar. When the length of `qT` equals to the length of `q` or `p`, then each value in the result is conditional on \( t > qT \) for each value in `qT`.

Arguments `rate` and `breakpoint` must match. The length of `rate` is the length of `breakpoint` + 1.

Author(s)

Tianchen Xu <zjph602xutianchen@gmail.com>
**cut_dat**

See Also
dpwexp, ppwexp, qpwexp, rpwexp

Examples

```r
# CDF and quantile function of conditional piecewise exp with rate 2, 1, 3 given t > 0.1
F2_con <- ppwexp_conditional(t = seq(0.1, 1.2, 0.01), qT = 0.1, rate = c(2, 1, 3), breakpoint = c(0.3, 0.8))
plot(t, F2_con, type = 'l', col = 'red', lwd = 2, main = "CDF and Quantile Function of Conditional \nPiecewise Exp Dist", xlim = c(0, 1.2), ylim = c(0, 1.2))
lines(F2_con, qpwexp_conditional(F2_con, qT = 0.1, rate = c(2, 1, 3), breakpoint = c(0.3, 0.8)), lty = 2, lwd = 2, col = 'red')

# compare with CDF and quantile function of unconditional piecewise exp with rate 2, 1, 3
t <- seq(0, 1.2, 0.01)
F2 <- ppwexp(t, rate = c(2, 1, 3), breakpoint = c(0.3, 0.8))
lines(t, F2, lwd = 2)
lines(F2, qpwexp(F2, rate = c(2, 1, 3), breakpoint = c(0.3, 0.8)), lty = 2, lwd = 2)
abline(v = 0.1, col = 'grey')
abline(h = 0.1, col = 'grey')
legend('topleft', c('CDF of piecewise exp dist given t > 0.1', 'quantile function of piecewise exp dist given t > 0.1', 'CDF of piecewise exp dist', 'quantile function of piecewise exp dist'), col = c('red', 'red', 'black', 'black'), lty = c(1, 2, 1, 2), lwd = 2)

# use rpwexp_conditional function to generate piecewise exp samples with rate 2, 1, 3 given t > 0.1
r_sample_con <- rpwexp_conditional(3000, qT = 0.1, rate = c(2, 1, 3), breakpoint = c(0.3, 0.8))
plot(ecdf(r_sample_con), col = 'red', lwd = 2, main = "Empirical CDF of Conditional Piecewise Exp Dist", xlim = c(0, 1.2), ylim = c(0, 1))
lines(seq(0.1, 1.2, 0.01), F2_con, lwd = 2)
legend('topleft', c('empirical CDF of piecewise exp dist given t > 0.1', 'true CDF of piecewise exp dist given t > 0.1'), col = c('red', 'black'), lty = c(1, 2), lwd = 2)
```

**Description**

Take a subset of a dataset by constraining the randomization time <= cut time. Additionally, it updates the follow-up time, censor/event indicator, censor reason, accordingly.

**Usage**

```r
cut_dat(cut, data, var_randT=NULL, var_followT=NULL, var_followT_abs=NULL, var_censor=NULL, var_event=NULL, var_censor_reason='status_at_end')
```
Arguments

cut 
cut time (from the beginning of the trial); only rows with randomization time <= cut will be kept.
data 
a data frame.
var_randT 
character; the variable name of randomization time. If missing, then the randomization time will be treated as 0 and NO subjects will be filtered by cut time.
var_followT 
character; the variable name of follow-up time (from randomization)
var_followT_abs 
character; the variable name of follow-up time (from the beginning of the trial)
var_censor 
character; the variable name of censoring (drop-out or death) indicator (1=censor, 0=event)
var_event 
character; the variable name of event indicator (1=event, 0=censor)
var_censor_reason 
character; the variable name of censoring reason (character variable). This variable will be created, if data does not contain it.

Details

We first filter rows that randomization time is equal to or less then cut time. Then we modify these columns (if provided):

- var_followT: change values to (cut - randomization time) if (follow-up time + randomization time) > cut
- var_followT_abs: change values to cut if (follow-up time from beginning) > cut
- var_censor: change values to 1 if (follow-up time from beginning) > cut
- var_event: change values to 0 if (follow-up time from beginning) > cut
- var_censor_reason: change values to 'cut' if (follow-up time from beginning) > cut

Value

A subset data frame with the same columns as data.

var_censor_reason is the only variable that is allowed to be absent in data. The function will create this variable in the returned data frame and set values 'cut' to the subjects whose (follow-up time from beginning) > cut.

Note

The original dataset data will NOT be modified.

Author(s)

Tianchen Xu <zjph602xutianchen@gmail.com>
cv.pwexp.fit

Examples

```r
event_dist <- function(n)rwexp(n, rate = c(0.1, 0.01, 0.2), breakpoint = c(5,14))
dat <- simdata(rand_rate = 20, total_sample = 1000, drop_rate = 0.03,
    advanced_dist = list(event_dist=event_dist),
    add_column = c('censor_reason','event','followT','followT_abs'))
cut <- quantile(dat$randT, 0.8)
train <- cut_dat(var_randT = 'randT', cut = cut, data = dat,
    var_followT = 'followT', var_followT_abs = 'followT_abs',
    var_event = 'event', var_censor_reason = 'censor_reason')
```

---

**cv.pwexp.fit**

Cross Validate a Piecewise Exponential Model

**Description**

Cross Validate a existing piecewise exponential model.

**Usage**

```r
## Default S3 method:
CV.pwexp.fit(time, event, nfold=5, nsim=100, breakpoint=NULL,
    nbreak=0, exclude_int=NULL, min_pt_tail=5, max_set=1000, seed=1818,
    optimizer='mle', tol=1e-4, parallel=FALSE, mc.core=4, ...)
## S3 method for class 'pwexp.fit'
CV.pwexp.fit(time, nfold=5, nsim=100, max_set=1000, seed=1818,
    optimizer='mle', tol=1e-4, parallel=FALSE, mc.core=4, ...)
```

**Arguments**

- `time` observed time from randomization or a `pwexp.fit` object.
- `event` the status indicator. See `pwexp.fit`.
- `nfold` the number of folds used in CV.
- `nsim` the number of simulations.
- `breakpoint` pre-specified breakpoints. See `pwexp.fit`.
- `nbreak` total number of breakpoints. See `pwexp.fit`.
- `exclude_int` an interval that excludes any estimated breakpoints. See `pwexp.fit`.
- `min_pt_tail` the minimum number of events used for estimating the tail (the hazard rate of the last piece). See `pwexp.fit`.
- `max_set` maximum estimated combination of breakpoints. See `pwexp.fit`.
- `seed` a random seed.
- `optimizer` one of the optimizers: mle, ols, or hybrid. See `pwexp.fit`.
- `tol` the minimum allowed gap between two breakpoints. The gap is calculated as \((\max(time)-\min(time))\times tol\). Keep it as default in most cases.
- `parallel` logical. If TRUE, use doSNOW package to run in parallel.
- `mc.core` number of processes allowed to be run in parallel.
- `...` internal function reserved.
Details

Use cross validation obtain the prediction log likelihood.

Value

A vector of length nsim containing the CV log likelihood in each round of simulation.

Author(s)

Tianchen Xu <zjph602xutianchen@gmail.com>

See Also

pwexp.fit

Examples

event_dist <- function(n) rpwexp(n, rate = c(0.1, 0.01, 0.2), breakpoint = c(5, 14))
dat <- simdata(rand_rate = 20, drop_rate = 0.03, total_sample = 1000,
    advanced_dist = list(event_dist = event_dist),
    add_column = c('censor_reason', 'event', 'followT', 'followT_abs'))

# here nsim=10 is for demo purpose, pls increase it in practice!!
cv0 <- cv.pwexp.fit(dat$followT, dat$event, nsim = 10, nbreak = 0)
cv1 <- cv.pwexp.fit(dat$followT, dat$event, nsim = 10, nbreak = 1)
cv2 <- cv.pwexp.fit(dat$followT, dat$event, nsim = 10, nbreak = 2)
sapply(list(cv0, cv1, cv2), median)
piecewise exponential

Arguments

- **x**, **q**: vector of quantiles.
- **p**: vector of probabilities.
- **n**: number of observations. Must be a positive integer with length 1.
- **rate**: a vector of rates in each piece.
- **breakpoint**: a vector of breakpoints. The length is `length(rate)-1`. Can be NULL if rate is a single value.
- **log**, **log.p**: logical; if TRUE, probabilities p are given as log(p).
- **lower.tail**: logical; if TRUE (default), probabilities are \( P[X \leq x] \), otherwise, \( P[X > x] \).
- **one_piece**: (only required when `safety_check=FALSE`) whether the distribution only has one piece (i.e., rate is a single value and breakpoint=NULL).
- **safety_check**: logical; whether check the input arguments; if FALSE, function has better computing performance by skipping all safety checks.

Details

The piecewise distribution function with piecewise rate \( \lambda_1, \ldots, \lambda_r \) is

\[
f(t) = \lambda_{r+1} \exp\left(\sum_{i=1}^{r} (\lambda_{i+1} - \lambda_i) d_i - \lambda_{r+1} t\right)
\]

for \( d_r \leq t < d_{r+1} \).

See webpage [https://zjph602xtc.github.io/PWEXP/](https://zjph602xtc.github.io/PWEXP/) for more details for its hazard function, cumulative hazard function, survival function, cumulative density function, quantile function.

Value

- `dpwexp` gives the density,
- `ppwexp` gives the distribution function,
- `qpwexp` gives the quantile function,
- `rpwexp` generates random deviates.

The length of the result is determined by **x**, **q**, **p** or **n** for `dpwexp`, `ppwexp`, `qpwexp` or `rpwexp`. You can only specify a single piecewise exponential distribution every time you call these functions. This is different from the exponential distribution functions in package `stats`.

Arguments **rate** and **breakpoint** must match. The length of rate is the length of breakpoint + 1.

Note

When **breakpoint=NULL**, the function calls exponential function in `stats`.

Author(s)

Tianchen Xu <zjph602xutianchen@gmail.com>

See Also

- `ppwexp_conditional`, `qpwexp_conditional`, `rpwexp_conditional`
Examples

# use rpwexp function to generate piecewise exp samples with rate 2, 1, 3
r_sample <- rpwexp(50000, rate=c(2, 1, 3), breakpoint=c(0.3, 0.8))
hist(r_sample, freq=FALSE, breaks=200, main="Density of Piecewise Exp Dist",
xlab='t', xlim=c(0, 1.2))

# piecewise exp density with rate 2, 1, 3
t <- seq(0, 1.5, 0.01)
f2 <- dpwexp(t, rate=c(2, 1, 3), breakpoint=c(0.3, 0.8))
points(t, f2, col='red', pch=16)

# exp distribution can be a special case of piecewise exp distribution
f1 <- dpwexp(t, rate=2)
lines(t, f1, lwd=2)

# CDF of piecewise exp with rate 2, 1, 3
F2 <- ppwexp(t, rate=c(2, 1, 3), breakpoint=c(0.3, 0.8), lower.tail=TRUE)
plot(t, F2, type='l', col='red', lwd=2, main="CDF and Quantile Function of Piecewise Exp Dist", ylim=c(0, 1.5))

# CDF of exp dist is compatible with our package
F1 <- ppwexp(t, rate=2, lower.tail=TRUE)
lines(t, F1, lwd=2)

# plot quantile functions of both distributions
lines(F1, qpwexp(F1, rate=2, lower.tail=TRUE), lty=2, lwd=2)
lines(F2, qpwexp(F2, rate=c(2, 1, 3), breakpoint=c(0.3,0.8), lower.tail=TRUE),
col='red', lty=2, lwd=2)
abline(0, 1, col='grey')
legend('topleft', c('CDF of piecewise exp with rate 2, 1, 3', 'quantile function of piecewise exp with rate 2, 1, 3', 'CDF of exp with rate 2', 'quantile function of exp with rate 2'), col=c('red', 'red', 'black', 'black'), lty=c(1, 2, 1, 2), lwd=2)

---

**plot_event**

**Plot Cumulative Event Curve**

**Description**

Plot cumulative event curve with right censoring data.

**Usage**

```r
## Default S3 method: plot_event(time, event, abs_time=TRUE, additional_event=0,)
```
### Arguments

- **time**
  - observed/follow-up time from individual randomization time (abs_time=FALSE) or from the first subject randomization time (abs_time=TRUE); or a predicted object from `predict.pwexp.fit`, or a predicted object with bootstrapping from `predict.boot.pwexp.fit`.

- **abs_time**
  - logical; if TRUE, time is the time from first randomization of the trial. if FALSE, time is the time from the randomization of each subject.

- **event**
  - the status indicator, 0=censor, 1=event. Other choices are TRUE/FALSE (TRUE = event).

- **additional_event**
  - adding the cumulative number of events by a constant number from the beginning.

- **add**
  - logical; if TRUE add lines to current plot.

- **plot**
  - logical; if FALSE, do not plot any lines, but return the line data

- **xyswitch**
  - logical; if TRUE, x-axis will be cumulative number of events and y will be the time.

- **eval_at**
  - a vector of the time (when xyswitch=FALSE) or the number of events (when xyswitch=TRUE) that you want to make prediction on.

- **alpha**
  - the significance level of the confidence interval.

- **type**
  - the type of prediction required. The default confidence returns the confidence interval without random error; the alternative predictive returns the predictive interval.

- **show_CI**
  - logical; if TRUE add confidence interval of the estimated event curve.

- **CI_par**
  - a list of parameters to control the appearance of lines of confidence intervals. The values pass to `lines`.

- **...**
  - other arguments (e.g., lwd, etc.) are passed over to `plot`.

### Details

A convenient function to calculate and plot the cumulative number of events.

Parameters in ... are passed to `plot` function to control the appearance of the event curve; parameters in `CI_par` are passed to `lines` function to control the appearance of confidence intervals. See examples for usage.

By default, `plot_event` plots a data frame in a new figure; and plots a predicted model in existing figure.
Value

A data frame containing these columns:

time             sorted time
n_event          cumulative number of events

Author(s)

Tianchen Xu <zjph602xutianchen@gmail.com>

See Also

plot_survival

Examples

```r
set.seed(1818)
event_dist <- function(n) rpwexp(n, rate = c(0.1, 0.01, 0.2), breakpoint = c(5,14))
dat <- simdata(rand_rate = 20, drop_rate = 0.03, total_sample = 1000,
               advanced_dist = list(event_dist=event_dist),
               add_column = c('censor_reason','event','followT','followT_abs'))
cut <- quantile(dat$randT, 0.8)
train <- cut_dat(var_randT = 'randT', cut = cut, data = dat,
                 var_followT = 'followT', var_followT_abs = 'followT_abs',
                 var_event = 'event', var_censor_reason = 'censor_reason')

fit_res3 <- pwexp.fit(train$followT, train$event, nbreak = 2)
fit_res_boot <- boot.pwexp.fit(fit_res3, nsim = 8) # here nsim=8 is for demo purpose,
               # pls increase it in practice

drop_indicator <- ifelse(train$censor_reason=='drop_out' & !is.na(train$censor_reason),1,0)
fit_res_censor <- pwexp.fit(train$followT, drop_indicator, nbreak = 0)
fit_res_censor_boot <- boot.pwexp.fit(fit_res_censor, nsim = 8)

cut_indicator <- train$censor_reason=='cut'
cut_indicator[is.na(cut_indicator)] <- 0

predicted_boot <- predict(fit_res_boot, cut_indicator = cut_indicator,
                           analysis_time = cut, censor_model_boot=fit_res_censor_boot,
                           future_rand=list(rand_rate=20, total_sample=NROW(dat)-NROW(train)))

plot_event(dat$followT_abs, abs_time = TRUE, event=dat$event, ylim=c(0,800))
plot_event(predicted_boot, eval_at = seq(40,90,5), CI_par = list(lty=3, lwd=2))

plot_event(dat$followT_abs, xlim=c(0,800), event=dat$event, xyswitch = TRUE)
plot_event(predicted_boot, xyswitch = TRUE, eval_at = seq(600,900,50))
```
Description

Plot KM curve with right censoring data or the survival curve of a fitted piecewise exponential model.

Usage

## Default S3 method:
plot_survival(time, event, add=FALSE, conf.int=FALSE, mark.time=TRUE, lwd=2, xlab='Follow-up time', ylab='Survival function', ...)
## S3 method for class 'pwexp.fit'
plot_survival(time, add=TRUE, show_breakpoint=TRUE, breakpoint_par=NULL, ...)
## S3 method for class 'boot.pwexp.fit'
plot_survival(time, add=TRUE, alpha=0.1, show_breakpoint=TRUE, breakpoint_par=NULL, show_CI=TRUE, CI_par=NULL, ...)

Arguments

time observed time from randomization or a pwexp.fit/ boot.pwexp.fit object.
event the status indicator, normally 0=censor, 1=event. Other choices are TRUE/FALSE (TRUE = event).
add logical; if TRUE add lines to current plot.
show_breakpoint logical; if TRUE add vertical dashed lines to indicate breakpoints.
breakpoint_par a list of parameters to control the apperance of vertical lines of breakpoints. The values pass to abline.
alpha the significance level of the confidence interval.
show_CI logical; if TRUE add confidence interval of the estimated curve. For KM estimator, use conf.int=TRUE to show CI band.
CI_par a list of parameters to control the apperance of lines of confidence intervals. The values pass to lines.
conf.int determines whether pointwise confidence intervals will be plotted. Passed over to plot.survfit.
mark.time controls the labeling of the curves. Passed over to plot.survfit.
lwd line width of the KM curve.
xlab x label.
ylab y label.
... other arguments are passed over to plot.survfit (default method) or plot (for class pwexp.fit).
predict

Predict Events for Piecewise Exponential Model

Description

Obtains event prediction and (optionally) confidence interval from a piecewise exponential model.

Details

For the default method, this a wrapper of `plot.survfit` function to plot right censoring data.

For class `pwexp.fit`, parameters in ... are passed to `plot` function to control the appearance of the survival curve; parameters in `breakpoint_par` are passed to `abline` function to control the appearance of vertical lines of breakpoints. See examples for usage.

For class `boot.pwexp.fit`, parameters in ... are passed to `plot` function to control the appearance of the survival curve; parameters in `breakpoint_par` are passed to `abline` function to control the appearance of vertical lines of breakpoints; parameters in `CI_par` are passed to `lines` function to control the appearance of confidence intervals. See examples for usage.

Value

No return value.

Author(s)

Tianchen Xu <tianchenxu@gmail.com>

See Also

`plot_event`

Examples

```r
# Define event distribution function
event_dist <- function(n) rpwexp(n, rate = c(0.1, 0.01, 0.2), breakpoint = c(5, 14))
# Simulate data
dat <- simdata(rand_rate = 20, drop_rate = 0.03, total_sample = 1000,
               advanced_dist = list(event_dist = event_dist),
               add_column = c('censor_reason', 'event', 'followT', 'followT_abs'))

# Plot survival curve
plot_survival(dat$followT, dat$event, xlim = c(0, 40))
fit_res <- pwexp.fit(dat$followT, dat$event, nbreak = 2)
plot_survival(fit_res, col = 'red', lwd = 3, breakpoint_par = list(col = 'grey', lwd = 2.5))
```
predict

Usage

## S3 method for class 'pwexp.fit'
predict(object, cut_indicator=NULL, analysis_time, censor_model=NULL,
         n_each=100, future_rand=NULL, seed=1818, ...)

## S3 method for class 'boot.pwexp.fit'
predict(object, cut_indicator=NULL, analysis_time,
         censor_model_boot=NULL, n_each=10, future_rand=NULL,
         seed=1818, ...)

Arguments

object
  a `pwexp.fit` or `boot.pwexp.fit` object. It is the event model for the primary endpoint.

cut_indicator
  (optional) A vector indicates which subject is censored due to the end of the trial. The length of
  the vector is the number of rows of the data used in `event_model/event_model_boot`. Value 0 means
  the subject had event or drop-out or death before the end of the trial; 1 means the subject didn’t
  have any of these. See details.

analysis_time
  the analysis time. This is the time length from the start of the trial to the time
  collecting data for the model.

censor_model
  an object of class `pwexp.fit` returned by the `pwexp.fit` function. It is the censoring model for drop-out and death.

censor_model_boot
  an object of class `boot.pwexp.fit` returned by the `boot.pwexp.fit` function. It is the censoring model with bootstrapping for drop-out and death.

n_each
  the number of iterations for each bootstrapping sample to obtain predictive CI. Typically, a value of
  10 to 100 should be enough.

future_rand
  the randomization curve in the following times. Can be `NULL` if all subjects have been randomized. You can specify `future` rand rate and `future` total number of samples to be randomized by `list(rand_rate= ,total_sample= )` or specify the `future` number of randomization each month by `list(rand_n= )`. See details.

seed
  a random seed.

...
  internal function reserved.

Details

The prediction will have a confidence interval only if the event model and censor model are bootstrap models.

cut_indicator indicates the status of each subject in the `event_model/event_model_boot` model at the end of the trial. Value 1 means the subject didn’t have events, drop-out or death at the end of the trial (or say, the subject was censored due to the end of the trial). When `cut_indicator` is NOT provided, we assign value 1 to the subject who didn’t have event (or drop-out, or death) in both `event_model/event_model_boot` and `censor_model/censor_model_boot` models.

future_rand is a list determining the parameter of randomization curve in the following times. For example, you specify `future` rand rate and `future` total number of samples to be randomized by `list(rand_rate= ,total_sample= )` or specify the number of randomization each month (e.g., 10,15,30,30 in four months) by `list(n_rand=c(10,15,30,30))`. 
Value

A list containing:

- `event_fun` number of events vs. time curve function in each bootstrap.
- `time_fun` time vs. number of events curve function in each bootstrap.

This returned list should be used in `plot_event` function for summarizing its result.

Author(s)

Tianchen Xu <zjph602xutianchen@gmail.com>

See Also

`plot_event`

Examples

```r
set.seed(1818)
event_dist <- function(n) rpwexp(n, rate = c(0.1, 0.2), breakpoint = 14)
dat <- simdata(rand_rate = 20, drop_rate = 0.03, total_sample = 500,
                advanced_dist = list(event_dist = event_dist),
                add_column = c('censor_reason', 'event', 'followT', 'followT_abs'))
cut <- quantile(dat$randT, 0.8)
train <- cut_dat(var_randT = 'randT', cut = cut, data = dat,
                 var_followT = 'followT', var_followT_abs = 'followT_abs',
                 var_event = 'event', var_censor_reason = 'censor_reason')

fit_res3 <- pwexp.fit(train$followT, train$event, nbreak = 1)
fit_res_boot <- boot.pwexp.fit(fit_res3, nsim = 8) # here nsim=8 is for demo purpose,
                 # pls increase it in practice

drop_indicator <- ifelse(train$censor_reason == 'drop_out' & !is.na(train$censor_reason), 1, 0)
fit_res_censor <- pwexp.fit(train$followT, drop_indicator, nbreak = 0)
fit_res_censor_boot <- boot.pwexp.fit(fit_res_censor, nsim = 8)

cut_indicator <- train$censor_reason == 'cut'
cut_indicator[is.na(cut_indicator)] <- 0

predicted_boot <- predict(fit_res_boot, cut_indicator = cut_indicator,
                          analysis_time = cut, censor_model_boot = fit_res_censor_boot,
                          future_rand = list(rand_rate = 20, total_sample = NROW(dat) - NROW(train)))
plot_event(train$followT_abs, train$event, xlim = c(0, 69), ylim = c(0, 800))
plot_event(predicted_boot, eval_at = 40:90)
plot_event(train$followT_abs, train$event, xyswitch = TRUE, ylim = c(0, 69), xlim = c(0, 800))
plot_event(predicted_boot, xyswitch = TRUE, eval_at = 600:900)
```
**pwexp.fit**  
*Fit the Piecewise Exponential Distribution*

**Description**

Fit the piecewise exponential distribution with right censoring data. User can specify all breakpoints, some of the breakpoints or let the function estimate the breakpoints.

**Usage**

`pwexp.fit(time, event, breakpoint=NULL, nbreak=0, exclude_int=NULL, min_pt_tail=5, max_set=10000, seed=1818, trace=FALSE, optimizer='mle', tol=1e-4)`

**Arguments**

- **time**: observed time from randomization. For right censored data, this is the follow-up time.
- **event**: the status indicator, normally 0=censor, 1=event. Other choices are TRUE/FALSE (TRUE = event).
- **breakpoint**: fixed breakpoints. Pre-specify some breakpoints. The maximum value must be earlier than the last event time.
- **nbreak**: total number of breakpoints in the model. This number includes the points specified in `breakpoint`. If `nbreak=NULL`, then `nbreak=ceiling(8*(# unique events)^0.2)`.
- **exclude_int**: an interval that excludes any estimated breakpoints (e.g., `exclude_int=c(10,Inf)` will exclude any estimated breakpoints after t=10). See details.
- **min_pt_tail**: the minimum number of events used for estimating the tail (the hazard rate of the last piece). See details.
- **max_set**: maximum estimated combination of breakpoints.
- **seed**: a random seed.
- **trace**: (internal use) logical; if TRUE, the returned data frame contains the log-likelihood of all possible breakpoints instead of the one with maximum likelihood.
- **optimizer**: one of the optimizers: mle, ols, or hybrid.
- **tol**: the minimum allowed gap between two breakpoints. The gap is calculated as `(max(time)-min(time))*tol`. Keep it as default in most cases.

**Details**

See webpage [https://zjph602xtc.github.io/PWEXP/] for a detailed description of the model and optimizers.

If user specifies `breakpoint`, we will check the values to make the model identifiable. Any breakpoints after the last event time will be removed; Any breakpoints before the first event time will be removed; a mid-point will be used if there are NO events between any two consecutive breakpoints. A warning will be given.
If user sets nbreak=NULL, then the function will automatically apply nbreak=ceiling(8*(# unique events)^0.2). This empirical number of breakpoints is for the reference below, and it may be too large in many cases.

Argument exclude_int is a vector of two values such as exclude_int=c(a, b) (b can be Inf). It defines an interval that excludes any estimated breakpoints. It is helpful when excluding breakpoints that are too close to the tail.

In order to obtain a more robust hazard rate estimation of the tail, user can set min_pt_tail to the minimum number of events for estimating the tail (last piece of the piecewise exponential). It only works for optimizer='mle'.

Value

A data frame (res) containing these columns:

- brk1, ..., brkx estimated breakpoints. The attr(res,'brk') can extract the vector of breakpoint from the model (res is the returned model from pwexp.fit).
- lam1, ..., lamx estimated piecewise hazard rates. The attr(res,'lam') can extract the vector of hazard rates from the model (res is the returned model from pwexp.fit).
- likelihood the log-likelihood of the model.
- AIC the Akaike information criterion of the model.
- BIC the Bayesian information criterion of the model.

Author(s)

Tianchen Xu <zjph602xutianchen@gmail.com>

References


See Also

boot.pwexp.fit, cv.pwexp.fit

Examples

event_dist <- function(n)rpwexp(n, rate=c(0.1, 0.01, 0.2), breakpoint=c(5,14))
dat <- simdata(rand_rate = 20, total_sample = 1000, drop_rate = 0.03,
  advanced_dist = list(event_dist=event_dist),
  add_column = c('censor_reason','event','followT','followT_abs'))
cut <- quantile(dat$randT, 0.8)
train <- cut_dat(var_randT = 'randT', cut = cut, data = dat,
  var_followT = 'followT', var_followT_abs = 'followT_abs',
  var_event = 'event', var_censor_reason = 'censor_reason')
fit_a0 <- pwexp.fit(train$followT, train$event, breakpoint = c(5,14))
fit_a1 <- pwexp.fit(train$followT, train$event, nbreak = 2, breakpoint = c(14))
fit_b0 <- pwexp.fit(train$followT, train$event, nbreak = 0)
fit_b1 <- pwexp.fit(train$followT, train$event, nbreak = 1)
fitted <- pwexp.fit(train$followT, train$event, nbreak = 2)

---

**Simulate Survival Data**

**Description**

`simdata` is used to simulate a clinical trial data with time-to-event endpoints.

**Usage**

```r
simdata(group="Group 1", strata="Strata 1", allocation=1,
 event_lambda=NA, drop_rate=NA, death_lambda=NA, n_rand=NULL,
 rand_rate=NULL, total_sample=NULL, add_column=c("followT"),
 simplify=TRUE, advanced_dist=NULL)
```

**Arguments**

- **group**: a character vector of the names of each group (e.g., `c('treatment','control')`).
- **strata**: a character vector of the names of strata in groups (e.g., `c('young','old')`).
- **allocation**: the relative ratio of sample size in each subgroup (`group*strata`). See details. The value will be recycled if the length is less than needed.
- **event_lambda**: the hazard rate of the primary endpoint (event). See details. The value will be recycled if the length is less than needed.
- **drop_rate** (optional): the drop-out rate (patients/month). Not hazard rate. See details. The value will be recycled if the length is less than needed.
- **death_lambda** (optional): the hazard rate of death. The value will be recycled if the length is less than needed.
- **n_rand** (required when `rand_rate=NULL`): a vector of the number of randomization each month; can be non-integers.
- **rand_rate** (required when `n_rand=NULL`): the randomization rate (patients/month; can be non-integer).
- **total_sample** (required when `n_rand=NULL`): total scheduled sample size.
- **add_column**: request additional columns of the returned data frame. Valid options are:
  - 'eventT_abs': absolute event time from the beginning of the trial (=eventT+randT)
  - 'dropT_abs': absolute drop-out time from the beginning of the trial (=dropT+randT)
  - 'deathT_abs': absolute death time from the beginning of the trial (=deathT+randT)
  - 'censor': censoring (drop-out or death) indicator
  - 'event': event indicator
  - 'censor_reason': censoring reason ('drop_out','death','never_event'(eventT=inf))
  - 'followT': follow-up time (true observed time) from randT
`'followT.abs'`: absolute follow-up time from the beginning of the trial

`simplify`  
whether drop unused columns (e.g., the group variable when there is only one
group). See details.

`advanced_dist`  
use user-specified distributions for event, drop-out and death. A list containing
random generation functions. See details and examples.

**Details**


The total number of subgroups will be `# treatment groups` * `# strata`. The strata variable will be distributed into each treatment group. For example, if `group = c('trt', 'placebo')`, `strata=c('A', 'B', 'C')`, then there will be 6 subgroups: trt+A, trt+B, trt+C, placebo+A, placebo+B, placebo+C. The lengths of allocation, event_lambda, drop_rate, death_lambda should be 6 as well. Note that the values will be recycled for these variables. For example, if `allocation=c(1,2,3)`, then the proportion of 6 subgroups is actually 1:2:3:1:2:3, which means 1:1 ratio for groups, 1:2:3 ratio in each stratum.

The `event_lambda` ($\lambda$) is the hazard rate of the interested events. The density function of events is $f(t) = \lambda e^{-\lambda t}$. Similarly, the `death_lambda` is the hazard rate of death.

The `drop_rate` is the probability of drop-out at $t = 1$, which means the hazard rate of drop-out is $-\log(1 - drop\_rate)$ (or say, `drop_rate=1 - e^{-hazardrate}`).

When `simplify=TRUE`, these columns will NOT be included:

- group when only one group is specified
- strata when only one stratum is specified
- eventT when event_lambda=NA
- dropT when drop_rate=NA
- deathT when death_lambda=NA

`advanced_dist` is used to define non-exponential distributions for event, drop-out or death. It is a list containing at least one of the elements: `event_dist`, `drop_dist`, `death_dist`. Each element has random generation functions for each subgroup. For example, `advanced_dist=list(event_dist=c(function1, function2), drop_dist=c(function3, function4))`. Here `function1`, `function3` are the event, drop-out generation function for the first subgroup; `function2`, `function4` for the second. If there is a third subgroup, `function1`, `function3` will be reused. Each data generation function (`functionX`) is a function with only one input argument `n` (sample size). If any of the `event_dist`, `drop_dist`, `death_dist` is missing, then we search for `event_lambda`, `drop_rate`, `death_lambda` to generate a exp distribution; if they are also missing, then corresponding variable will not be generated.

**Value**

A data frame containing the some of these columns:

<table>
<thead>
<tr>
<th>ID</th>
<th>subject ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>group</td>
<td>group indicator</td>
</tr>
</tbody>
</table>
strata  stratum indicator
randT  randomization time (from the beginning of the trial)
eventT event time (from randT)
eventT_abs event time (from the beginning of the trial)
dropT  drop-out time (from randT)
dropT_abs drop-out time (from the beginning of the trial)
deathT death time (from randT)
deathT_abs death time (from the beginning of the trial)
censor censoring (drop-out or death) indicator
censor_reason censoring reason (’drop_out’,’death’,’never_event’(followT=inf))
event event indicator
followT follow-up time / observed time (from randT)
followT_abs follow-up time / observed time (from the beginning of the trial)

Note
event_lambda, drop_rate, death_lambda can be 0, which means the corresponding subgroup will have an Inf value for each variable.

Author(s)
Tianchen Xu <zjph602xutianchen@gmail.com>

See Also
rpwexp, rpwexp_conditional

Examples

# Two groups with two strata. In the treatment group, there is a treatment
# sensitive stratum and a non-sensitive stratum. In the placebo group, all
# subjects are the same. Treatment:place=1:2. Drop rate=1% only in treatment group.
dat <- simdata(group=c(’trt’, ’place’), strata=c(’sensitive’,’non-sensitive’),
allocation = c(1,1,2,2), rand_rate = 20, total_sample = 1000,
event_lambda = c(0.1, 0.2, 0.01, 0.01),
drop_rate = c(0.01, 0.01, 0, 0))

# randomized subjects
table(dat$group,dat$strata)
# randomization curve
plot(sort(dat$randT), 1:1000, xlab=’time’, ylab=’randomized subjects’)
# event time in treatment group
plot(ecdf(dat$eventT[dat$group==’trt’ & dat$strata==’sensitive’]))
lines(ecdf(dat$eventT[dat$group==’trt’ & dat$strata==’non-sensitive’]), col=’red’)
```
dist_trt <- function(n) rpwexp(n, rate=c(0.01, 0.05, 0.01), breakpoint = c(30,60))
dist_placebo <- function(n) rpwexp(n, rate=c(0.01, 0.005), breakpoint = c(50))
dat <- simdata(group = c('trt', 'placebo'), n_rand = c(rep(10,50),rep(20,10)),
              death_lambda = 0.01,
              advanced_dist = list(event_dist=c(dist_trt, dist_placebo),
                                    drop_dist=function(n)rweibull(n,3,40)))

# randomized subjects
table(dat$group)
# randomization curve
plot(sort(dat$randT), 1:700, xlab='time', ylab='randomized subjects')
# event time in both groups
plot(ecdf(dat$eventT[dat$group=='trt']), xlim=c(0,100))
lines(ecdf(dat$eventT[dat$group=='placebo']), col='red')
# drop-out time
plot(ecdf(dat$dropT), xlim=c(0,100))

# mixture cure distribution, 20% of the subject are cured and will not have events
dat <- simdata(strata=c('cure', 'non-cure'), allocation=c(20,80),
                event_lambda=c(0, 0.38), n_rand = rep(20,30),
                add_column = c('eventT_abs', 'censor', 'event',
                                'censor_reason', 'followT', 'followT_abs'))
```

---

**sim_followup**

*Estimate follow up time and number of events by simulation*

**Description**

`sim_followup` is used to estimate follow-up time and number of events (given calendar time, or number of randomized samples, or number of events).

**Usage**

```
sim_followup(at, type = "calander", group="Group 1", strata="Strata 1",
             allocation=1, event_lambda=NA, drop_rate=NA, death_lambda=NA,
             n_rand=NULL, rand_rate=NULL, total_sample=NULL, extra_follow=0,
             by_group=FALSE, by_strata=FALSE, advanced_dist=NULL,
             stat=c(mean, median, sum), follow_up_endpoint=c('death', 'drop_out',
                                                            'cut'), count_in_extra_follow=FALSE, count_insufficient_event=FALSE,
             start_date=NULL, rep=300, seed=1818)
```

**Arguments**

- **at** specify a vector of occasions. When type='calander', at is the time from first randomization; when type='event', at is the number of accumulated events; when type='sample', at is the number of randomized samples.
- **type** specify the type of at. Must be 'calander', event or sample.
- **group** a character vector of the names of each group (e.g., c('treatment', 'control')). See `simdata`. 

---
**strata**

a character vector of the names of strata in groups (e.g., c('young', 'old')). See simdata.

**allocation**

the relative ratio of sample size in each subgroup (group*strata). The value will be recycled if the length is less than needed. See simdata.

**event_lambda**

the hazard rate of the primary endpoint (event). The value will be recycled if the length is less than needed. See simdata.

**drop_rate**

(optional) the drop-out rate (patients/month). Not hazard rate. The value will be recycled if the length is less than needed. See simdata.

**death_lambda**

(optional) the hazard rate of death. The value will be recycled if the length is less than needed. See simdata.

**n_rand**

(required when rand_rate=NULL) a vector of the number of randomization each month; can be non-integers. See simdata.

**rand_rate**

(required when n_rand=NULL) the randomization rate (patients/month; can be non-integer). See simdata.

**total_sample**

(required when n_rand=NULL) total scheduled sample size. See simdata.

**extra_follow**

delay the analysis time by extra time (extra_follow) after the time specified by at. See details.

**by_group**

logical; if TRUE, also return results by each group.

**by_strata**

logical; if TRUE, also return results by each stratum.

**advanced_dist**

use user-specified distributions for event, drop-out and death. A list containing random generation functions. See details and examples in simdata.

**stat**

a vector of functions to summarize the follow-up time. See example.

**follow_up_endpoint**

Which endpoints can be regarded as the end of follow-up. Choose from 'death', 'drop_out', 'cut' (censored at the end of the trial) or 'event'.

**count_in_extra_follow**

logical; whether to count subjects who are randomized after the time specified by at but before the time specified by at + extra_follow.

**count_insufficient_event**

logical; only affects the result when type='event'. If TRUE, for samples that cannot achieve required number of events, the last follow-up time is the analysis time. If FALSE, these samples will be dropped.

**start_date**

the start date of the first randomization; in the format: "2000-01-30"

**rep**

number simulated iterations.

**seed**

a random seed.

**Details**

See the help document of simdata for most arguments details.

When type='calander', the function estimates the follow-up time and number of events at time at plus extra_follow; when type='event', the function estimates these at the time when total number of events is at plus time extra_follow; when type='sample', the function estimates these at the time when total number of randomized subjects is at plus time extra_follow.

The stat specifies a vector of user defined functions. Each of them must take a vector of individual follow-up time as input and return a single summary value. See example.
Value

A data frame containing the some of these columns:

- **ID**: subject ID
- **group**: group indicator
- **strata**: stratum indicator
- **randT**: randomization time (from the beginning of the trial)
- **eventT**: event time (from randT)
- **eventT_abs**: event time (from the beginning of the trial)
- **dropT**: drop-out time (from randT)
- **dropT_abs**: drop-out time (from the beginning of the trial)
- **deathT**: death time (from randT)
- **deathT_abs**: death time (from the beginning of the trial)
- **censor**: censoring (drop-out or death) indicator
- **censor_reason**: censoring reason (‘drop_out’, ‘death’, ‘never_event’(followT=inf))
- **event**: event indicator
- **followT**: follow-up time / observed time (from randT)
- **followT_abs**: follow-up time / observed time (from the beginning of the trial)

Note

event_lambda, drop_rate, death_lambda can be 0, which means the corresponding subgroup will have an Inf value for each variable.

Author(s)

Tianchen Xu <zjph602xutianchen@gmail.com>

See Also

`simdata`

Examples

# Two groups. Treatment:place=1:2. Drop rate=3%/month. Hazard ratio=0.7.

# define the piecewiese exponential event generation function
myevent_dist_trt <- function(n)rpwexp(n, rate=c(0.1, 0.01, 0.2)*0.7, breakpoint=c(5,14))
myevent_dist_con <- function(n)rpwexp(n, rate=c(0.1, 0.01, 0.2), breakpoint=c(5,14))

# user defined summary function, the proportion of subjects that follow more than 12 month
prop_12 <- function(x)mean(x >= 12)

# estimate the event curve or timeline:
# (here rep=60 is for demo purpose only, please increase this value in practice!)
event_curve <- sim_followup(at=seq(20,90,10), type = 'calendar', group = c('trt','con'),
rand_rate = 20, total_sample = 1000, drop_rate = 0.03, allocation = 1:2,
advanced_dist = list(event_dist=c(myevent_dist_trt, myevent_dist_con)),
by_group = TRUE, stat = c(median, mean, prop_12), start_date = "2020-01-01",
rep=60)
time_curve <- sim_followup(at=seq(200,600,100), type = 'event', group = c('trt','con'),
rand_rate = 20, total_sample = 1000, drop_rate = 0.03, allocation = 1:2,
advanced_dist = list(event_dist=c(myevent_dist_trt, myevent_dist_con)),
stat = c(median, mean, prop_12), start_date = "2020-01-01", rep=60)
# plot event curve or timeline
plot(event_curve$T_all$analysis_time_c, event_curve$T_all$event, xlab='Time',
ylab='Number of events', type='b')
plot(time_curve$T_all$event, time_curve$T_all$analysis_time_c, xlab='Number of events', ylab='Time', type='b')
Index

abline, 13, 14
boot.pwexp.fit, 2, 13, 15, 18
conditional piecewise exponential, 3
cut_dat, 5
cv.pwexp.fit, 7, 18
dpwexp, 5
dpwexp (piecewise exponential), 8
lines, 11, 13, 14
piecewise exponential, 8
plot, 11, 13, 14
plot.survfit, 13, 14
plot_event, 10, 14, 16
plot_survival, 12, 13
ppwexp, 5
ppwexp (piecewise exponential), 8
ppwexp conditional, 9
ppwexp conditional (conditional piecewise exponential), 3
predict, 14
predict.boot.pwexp.fit, 11
predict.pwexp.fit, 11
pwexp.fit, 2, 3, 7, 8, 13, 15, 17
qpwexp, 5
qpwexp (piecewise exponential), 8
qpwexp conditional, 9
qpwexp conditional (conditional piecewise exponential), 3
rpwexp, 5, 21
rpwexp (piecewise exponential), 8
rpwexp conditional, 9, 21
rpwexp conditional (conditional piecewise exponential), 3
sim_followup, 22
simdata, 19, 22–24