

# Package ‘PwePred’

March 26, 2026

**Type** Package

**Title** Event/Timeline Prediction Model Based on Piecewise Exponential

**Imports** graphics, grDevices, stats, methods, utils, segmented,  
fastmatch, foreach, doSNOW, parallel

**Depends** survival, R (>= 2.10)

**Suggests** knitr, RColorBrewer, rmarkdown

**Version** 1.1.2

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

**Description** Efficient algorithm for estimating piecewise exponential hazard models for right-censored data, and is useful for reliable power calculation, study design, and event/timeline prediction for study monitoring.

**License** MIT + file LICENSE

**Encoding** UTF-8

**VignetteBuilder** knitr

**LazyData** true

**Repository** CRAN

**Date/Publication** 2026-03-26 09:00:02 UTC

**NeedsCompilation** no

**Author** Tianchen Xu [aut, cre] (ORCID: <<https://orcid.org/0000-0002-0102-7630>>)

## Contents

boot.pwexpm . . . . .	2
boot.pwexpm_fit . . . . .	4
coef.pwexpm . . . . .	5
conditional piecewise exponential . . . . .	6
confint.boot.pwexpm . . . . .	8
cut_dat . . . . .	8
cv.pwexpm . . . . .	10
cv.pwexpm_fit . . . . .	11
piecewise exponential . . . . .	13

plot_event . . . . .	15
plot_survival . . . . .	18
predict . . . . .	20
pwexpm . . . . .	22
pwexpm_fit . . . . .	24
pwe_data . . . . .	26
pwe_data_robust_tail . . . . .	27
ruppert_ipd . . . . .	28
simdata . . . . .	29
sim_followup . . . . .	32

<b>Index</b>	<b>36</b>
--------------	-----------

---

boot.pwexpm	<i>Bootstrap a PWE Model with Formula or Existing Model</i>
-------------	---

---

## Description

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

## Usage

```
## Default S3 method:
boot.pwexpm(Surv, data, 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 'pwexpm'
boot.pwexpm(Surv, nsim=100, max_set=1000, seed=1818,
             optimizer='mle', tol=1e-4, parallel=FALSE, mc.core=4, ...)
```

## Arguments

Surv	a <a href="#">Surv</a> object indicating event time and status or a <a href="#">pwexpm</a> object.
data	a data frame in which to interpret the variables named in the Surv argument.
nsim	the number of repeated bootstrapping.
breakpoint	pre-specified breakpoints. See <a href="#">pwexpm</a> .
nbreak	total number of breakpoints. See <a href="#">pwexpm</a> .
exclude_int	an interval that excludes any estimated breakpoints. See <a href="#">pwexpm</a> .
min_pt_tail	the minimum number of events used for estimating the tail (the hazard rate of the last piece). See <a href="#">pwexpm</a> .
max_set	maximum estimated combination of breakpoints. See <a href="#">pwexpm</a> .
seed	a random seed. Do not set seed if seed=NULL.
optimizer	one of the optimizers: mle, ols, or hybrid. See <a href="#">pwexpm</a> .

tol	the minimum allowed gap between two breakpoints. The gap is calculated as $(\max(\text{time}) - \min(\text{time})) * \text{tol}$ . Keep it as default in most cases.
parallel	logical. If TRUE, use <b>doSNOW</b> package to run in parallel.
mc.core	number of processes allowed to be run in parallel.
...	internal function reserved.

### Details

Use bootstrap to repeatedly call [pwexpm](#) to estimate the uncertainty of parameters.

### Value

A object of class "boot.pwexpm" is a list containing the following components:

brk	estimated breakpoints in each row.
lam	estimated piecewise hazard rates in each row.
logLik	the log-likelihood of the original model.
AIC	the Akaike information criterion of the original model.
BIC	the Bayesian information criterion of the original model.
para	the parameters used to estimate the model.

The plot function can be used to make a simple plot for `boot.pwexpm`.

### Author(s)

Tianchen Xu <zjph602xutianchen@gmail.com>

### See Also

[boot.pwexpm\\_fit](#)

### Examples

```
event_dist <- function(n)rpwexpm(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'))

fit_res3 <- pwexpm(Surv(followT, event), data = dat, nbreak = 2)
fit_res_boot <- boot.pwexpm(fit_res3, nsim = 10) # here nsim=10 is for demo purpose,
                                              # pls increase it in practice

plot_survival(dat$followT, dat$event, xlim=c(0,40))
plot_survival(fit_res_boot, col='red', CI_par = list(col='red'))
brk_ci <- confint(fit_res_boot, level = 0.95, parm = 'brk')
abline(v=brk_ci, col='grey', lwd=2)
```

---

boot.pwexpm\_fit      *Bootstrap a PWE Model with Time and Event Vectors*

---

### Description

Build a piecewise exponential model with bootstrapping.

### Usage

```
boot.pwexpm_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, ...)
```

### Arguments

time	observed time from randomization.
event	the status indicator. See <a href="#">pwexpm_fit</a> .
nsim	the number of repeated bootstrapping.
breakpoint	pre-specified breakpoints. See <a href="#">pwexpm_fit</a> .
nbreak	total number of breakpoints. See <a href="#">pwexpm_fit</a> .
exclude_int	an interval that excludes any estimated breakpoints. See <a href="#">pwexpm_fit</a> .
min_pt_tail	the minimum number of events used for estimating the tail (the hazard rate of the last piece). See <a href="#">pwexpm_fit</a> .
max_set	maximum estimated combination of breakpoints. See <a href="#">pwexpm_fit</a> .
seed	a random seed. Do not set seed if seed=NULL.
optimizer	one of the optimizers: mle, ols, or hybrid. See <a href="#">pwexpm_fit</a> .
tol	the minimum allowed gap between two breakpoints. The gap is calculated as $(\max(\text{time}) - \min(\text{time})) * \text{tol}$ . Keep it as default in most cases.
parallel	logical. If TRUE, use <b>doSNOW</b> package to run in parallel.
mc.core	number of processes allowed to be run in parallel.
...	internal function reserved.

### Details

Use bootstrap to repeatedly call [pwexpm\\_fit](#) to estimate the uncertainty of parameters.

### Value

A object of class "boot.pwexpm" is a list containing the following components:

brk	estimated breakpoints in each row.
lam	estimated piecewise hazard rates in each row.
logLik	the log-likelihood of the original model.

AIC                    the Akaike information criterion of the original model.  
 BIC                    the Bayesian information criterion of the original model.  
 para                    the parameters used to estimate the model.

The plot function can be used to make a simple plot for boot.pwexpm.

### Author(s)

Tianchen Xu <zjph602xutianchen@gmail.com>

### See Also

[boot.pwexpm](#)

### Examples

```
event_dist <- function(n)rpwexpm(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'))

fit_res3 <- boot.pwexpm_fit(dat$followT, dat$event, nbreak = 2, nsim = 10)
# here nsim=10 is for demo purpose. Pls increase it in practice.

plot_survival(dat$followT, dat$event, xlim=c(0,40))
plot_survival(fit_res3, col='red', CI_par = list(col='red'))
brk_ci <- apply(fit_res3$brk, 2, function(x)quantile(x,c(0.025,0.975)))
abline(v=brk_ci, col='grey', lwd=2)
```

---

coef.pwexpm

*Extract pwexpm Coefficients*

---

### Description

The estimated coefficients for the pwexpm are extracted.

### Usage

```
## S3 method for class 'pwexpm'
coef(object, parm, ...)
```

### Arguments

object                a pwexpm object.  
 parm                 a specification of which parameters are to be extracted (either 'lam' for hazard rates, or 'brk' for change-points). If missing, all parameters are considered.  
 ...                    some methods for this generic require additional arguments. None are used in this method.

**Value**

A data frame containing the requested coefficients.

**Author(s)**

Tianchen Xu <zjph602xutianchen@gmail.com>

---

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 `rate` given  $t > qT$ .

**Usage**

```
ppwexpm_conditional(q, qT, rate=1, breakpoint=NULL, lower.tail=TRUE,
                    log.p=FALSE, one_piece, safety_check=TRUE)
qpwexpm_conditional(p, qT, rate=1, breakpoint=NULL, lower.tail=TRUE,
                    log.p=FALSE, one_piece, safety_check=TRUE)
rpwexpm_conditional(n, qT, rate, breakpoint=NULL)
```

**Arguments**

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

`ppwexpm_conditional` gives the conditional distribution function, `qpwexpm_conditional` gives the conditional quantile function, and `rpwexpm_conditional` generates conditional random variables.

The length of the result is determined by `q`, `p` or `n` for `ppwexpm_conditional`, `qpwexpm_conditional` or `rpwexpm_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 `rpwexpm_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>

**See Also**

[dpwexpm](#), [ppwexpm](#), [qpwexpm](#), [rpwexpm](#)

**Examples**

```
# CDF and quantile function of conditional piecewise exp with rate 2, 1, 3 given t > 0.1
t <- seq(0.1, 1.2, 0.01)
F2_con <- ppwexpm_conditional(t, 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 \nPieewise Exp Dist", xlim=c(0, 1.2), ylim=c(0, 1.2))
lines(F2_con, qpwexpm_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 <- ppwexpm(t, rate=c(2, 1, 3), breakpoint=c(0.3,0.8))
lines(t, F2, lwd=2)
lines(F2, qpwexpm(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 rpwexpm_conditional function to generate piecewise exp samples with rate 2, 1, 3 given t > 0.1
r_sample_con <- rpwexpm_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))

# compare with its CDF
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)
```

---

confint.boot.pwexpm     *Confidence Intervals for Model Parameters in boot.pwexpm*

---

### Description

Computes confidence intervals for one or more parameters in a bootstrapped PWE model.

### Usage

```
## S3 method for class 'boot.pwexpm'
confint(object, parm, level = 0.90, ...)
```

### Arguments

object	a boot.pwexpm object.
parm	a specification of which parameters are to be given confidence intervals (either 'lam' for hazard rates, or 'brk' for change-points). If missing, all parameters are considered.
level	the confidence level required.
...	some methods for this generic require additional arguments. None are used in this method.

### Value

A matrix with columns giving lower and upper confidence limits for each parameter. These will be labelled as  $(1-\text{level})/2$  and  $1 - (1-\text{level})/2$  in %.

### Author(s)

Tianchen Xu <zjph602xutianchen@gmail.com>

---

cut\_dat     *Cut Data before a Specified Time*

---

### Description

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

### Usage

```
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 $\leq$ 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 than 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>

**Examples**

```

event_dist <- function(n)rpwexpm(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.pwexpm

*Cross Validate a PWE Model with Formula or Existing Model***Description**

Cross validate an existing piecewise exponential model.

**Usage**

```

## Default S3 method:
cv.pwexpm(Surv, data, 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 'pwexpm'
cv.pwexpm(Surv, nfold=5, nsim=100, max_set=1000, seed=1818,
          optimizer='mle', tol=1e-4, parallel=FALSE, mc.core=4, ...)

```

**Arguments**

Surv	a <a href="#">Surv</a> object indicating event time and status or a <a href="#">pwexpm</a> object.
data	a data frame in which to interpret the variables named in the Surv argument.
nfold	the number of folds used in CV.
nsim	the number of simulations.
breakpoint	pre-specified breakpoints. See <a href="#">pwexpm</a> .
nbreak	total number of breakpoints. See <a href="#">pwexpm</a> .
exclude_int	an interval that excludes any estimated breakpoints. See <a href="#">pwexpm</a> .
min_pt_tail	the minimum number of events used for estimating the tail (the hazard rate of the last piece). See <a href="#">pwexpm</a> .
max_set	maximum estimated combination of breakpoints. See <a href="#">pwexpm</a> .
seed	a random seed. Do not set seed if seed=NULL.
optimizer	one of the optimizers: mle, ols, or hybrid. See <a href="#">pwexpm</a> .
tol	the minimum allowed gap between two breakpoints. The gap is calculated as $(\max(\text{time}) - \min(\text{time})) * \text{tol}$ . Keep it as default in most cases.
parallel	logical. If TRUE, use <b>doSNOW</b> 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 object of class "cv.pwexpm" is a numeric vector containing the CV log likelihood in each round of simulation. The plot function can be used to make a boxplot of the CV log likelihoods from pwexpm.

**Author(s)**

Tianchen Xu <zjph602xutianchen@gmail.com>

**See Also**

[cv.pwexpm\\_fit](#)

**Examples**

```
event_dist <- function(n)rpwexpm(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.pwexpm(Surv(followT, event), data=dat, nsim = 10, nbreak = 0)
cv1 <- cv.pwexpm(Surv(followT, event), data=dat, nsim = 10, nbreak = 1)
cv2 <- cv.pwexpm(Surv(followT, event), data=dat, nsim = 10, nbreak = 2)
sapply(list(cv0,cv1,cv2), median)
```

---

cv.pwexpm\_fit

---

*Cross Validate a PWE Model with Time and Event Vectors*


---

**Description**

Build and cross validate a piecewise exponential model with time and event vectors.

**Usage**

```
cv.pwexpm_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, ...)
```

**Arguments**

time	observed time from randomization.
event	the status indicator. See <a href="#">pwexpm_fit</a> .
ifold	the number of folds used in CV.
nsim	the number of simulations.
breakpoint	pre-specified breakpoints. See <a href="#">pwexpm_fit</a> .
nbreak	total number of breakpoints. See <a href="#">pwexpm_fit</a> .
exclude_int	an interval that excludes any estimated breakpoints. See <a href="#">pwexpm_fit</a> .
min_pt_tail	the minimum number of events used for estimating the tail (the hazard rate of the last piece). See <a href="#">pwexpm_fit</a> .
max_set	maximum estimated combination of breakpoints. See <a href="#">pwexpm_fit</a> .
seed	a random seed. Do not set seed if seed=NULL.
optimizer	one of the optimizers: mle, ols, or hybrid. See <a href="#">pwexpm_fit</a> .
tol	the minimum allowed gap between two breakpoints. The gap is calculated as $(\max(\text{time}) - \min(\text{time})) * \text{tol}$ . Keep it as default in most cases.
parallel	logical. If TRUE, use <b>doSNOW</b> 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 object of class "cv.pwexpm" is a numeric vector containing the CV log likelihood in each round of simulation. The plot function can be used to make a boxplot of the CV log likelihoods from pwexpm.

**Author(s)**

Tianchen Xu <zjph602xutianchen@gmail.com>

**See Also**

[cv.pwexpm](#)

**Examples**

```
event_dist <- function(n)rpwexpm(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.pwexpm_fit(dat$followT, dat$event, nsim = 10, nbreak = 0)
cv1 <- cv.pwexpm_fit(dat$followT, dat$event, nsim = 10, nbreak = 1)
cv2 <- cv.pwexpm_fit(dat$followT, dat$event, nsim = 10, nbreak = 2)
sapply(list(cv0,cv1,cv2), median)

```

---

piecewise exponential *The Piecewise Exponential Distribution*

---

### Description

Density, distribution function, quantile function and random generation for the piecewise exponential distribution with piecewise rate rate.

### Usage

```

dpwexpm(x, rate = 1, breakpoint = NULL, log = FALSE, one_piece, safety_check = TRUE)
ppwexpm(q, rate = 1, breakpoint = NULL, lower.tail = TRUE, log.p = FALSE,
        one_piece, safety_check = TRUE)
qpwexpm(p, rate = 1, breakpoint = NULL, lower.tail = TRUE, log.p = FALSE,
        one_piece, safety_check = TRUE)
rpwexpm(n, rate = 1, breakpoint = NULL)

```

### 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  $\lambda_1, \dots, \lambda_r$  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/> for more details for its hazard function, cumulative hazard function, survival function, cumulative density function, quantile function.

### Value

dpwexpm gives the density, ppwexpm gives the distribution function, qpwexpm gives the quantile function, and rpwexpm generates random deviates.

The length of the result is determined by x, q, p or n for dpwexpm, ppwexpm, qpwexpm or rpwexpm. 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

[ppwexpm\\_conditional](#), [qpwexpm\\_conditional](#), [rpwexpm\\_conditional](#)

### Examples

```
# use rpwexpm function to generate piecewise exp samples with rate 2, 1, 3
r_sample <- rpwexpm(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 <- dpwexpm(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 <- dpwexpm(t, rate=2)
lines(t, f1, lwd=2)
legend('topright', c('exp dist with rate 2', 'piecewise exp dist with rate 2, 1,
  3', 'histogram of piecewise exp dist with rate 2, 1, 3'),
     col=c('black', 'red'), fill=c(NA, NA, 'grey'), border=c('white', 'white',
  'black'), lty=c(1, NA, NA), pch=c(NA, 16, NA), lwd=2)

# CDF of piecewise exp with rate 2, 1, 3
F2 <- ppwexpm(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", xlim=c(0, 1.5), ylim=c(0, 1.5))

# CDF of exp dist is compatible with our package
```

```

F1 <- ppwexpm(t, rate=2, lower.tail=TRUE)
lines(t, F1, lwd=2)

# plot quantile functions of both distributions
lines(F1, qpwexpm(F1, rate=2, lower.tail=TRUE), lty=2, lwd=2)
lines(F2, qpwexpm(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	<i>Plot Cumulative Event Curve with Survival Data or Predicted PWE Model</i>
------------	--

---

## Description

Plot cumulative event curve with right censoring data.

## Usage

```

## Default S3 method:
plot_event(time, event, abs_time=TRUE, additional_event=0,
           add=FALSE, plot=TRUE, xyswitch=FALSE, ...)
## S3 method for class 'predict.pwexpm'
plot_event(time, abs_time=TRUE, add=TRUE, plot=TRUE,
           xyswitch=FALSE, eval_at=NULL, ...)
## S3 method for class 'predict.pwexpm'
plot(x, ...)
## S3 method for class 'predict.boot.pwexpm'
plot_event(time, abs_time=TRUE, alpha=0.1, type='confidence',
           add=TRUE, plot=TRUE, xyswitch=FALSE, eval_at=NULL,
           show_CI=TRUE, CI_par=NULL, ...)
## S3 method for class 'predict.boot.pwexpm'
plot(x, ...)

```

## Arguments

time	observed/follow-up time from individual randomization time ( <code>abs_time=FALSE</code> ) or from the first subject randomization time ( <code>abs_time=TRUE</code> ); or a predicted object from <code>predict.pwexpm</code> , or a predicted object with bootstrapping from <code>predict.boot.pwexpm</code> .
x	the same as time.
abs_time	logical; if <code>TRUE</code> , time is the time from first randomization of the trial. if <code>FALSE</code> , 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
xswitch	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 xswitch=FALSE) or the number of events (when xswitch=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 <a href="#">lines</a> .
...	other arguments (e.g., lwd, etc.) are passed over to <a href="#">plot.default</a> .

## Details

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

Parameters in ... are passed to [plot.default](#) 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.

plot method for classes `predict.pwexpm` and `predict.boot.pwexpm` behaves the same as `plot_event`, except that it plots a predicted model in a new figure by default.

## Value

When `xswitch=FALSE`, the function returns a data frame containing these columns:

time	sorted time at eval_at
n_event	predicted cumulative number of events
(alpha/2) n_event	(for <code>predict.boot.pwexpm</code> ) lower alpha level predicted cumulative number of events
1-(alpha/2) n_event	(for <code>predict.boot.pwexpm</code> ) upper alpha level predicted cumulative number of events

When `xswitch=TRUE`, the function returns a data frame containing these columns:

n\_event sorted cumulative number of events at eval\_at  
time predicted required time  
(alpha/2) time (for predict.boot.pwexpm) lower alpha level predicted required time  
1-(alpha/2) time (for predict.boot.pwexpm) upper alpha level predicted required time

**Author(s)**

Tianchen Xu <zjph602xutianchen@gmail.com>

**See Also**

[plot\\_survival](#)

**Examples**

```
set.seed(1818)
event_dist <- function(n)rpwexpm(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 <- pwexpm(Surv(followT, event), data=train, nbreak = 1)
fit_res_boot <- boot.pwexpm(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 <- pwexpm_fit(train$followT, drop_indicator, nbreak = 0)
fit_res_censor_boot <- boot.pwexpm(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=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(0,500))
plot_event(predicted_boot, eval_at = seq(40,90,5), CI_par = list(lty=3, lwd=2))

plot_event(train$followT_abs, train$event, xyswitch = TRUE, ylim=c(0,69), xlim=c(0,400))
plot_event(predicted_boot, xyswitch = TRUE, eval_at = seq(250,400,50))
```

plot\_survival

*Plot Survival Curve for Survival Data or PWE Model***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 'pwexpm'
plot_survival(time, add=TRUE, show_breakpoint=TRUE,
              breakpoint_par=NULL, ...)
## S3 method for class 'pwexpm'
plot(x, ...)
## S3 method for class 'boot.pwexpm'
plot_survival(time, add=TRUE, alpha=0.1, show_breakpoint=TRUE,
              breakpoint_par=NULL, show_CI=TRUE, CI_par=NULL, ...)
## S3 method for class 'boot.pwexpm'
plot(x, ...)
```

**Arguments**

time	observed time from randomization or a <a href="#">pwexpm/ boot.pwexpm</a> object.
x	the same as time.
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 appearance of vertical lines of breakpoints. The values pass to <a href="#">abline</a> .
alpha	the significance level of the confidence interval.
show_CI	logical; if TRUE add confidence interval of the estimated curve. For KM estimator, use <code>conf.int=TRUE</code> to show CI band.
CI_par	a list of parameters to control the appearance of lines of confidence intervals. The values pass to <a href="#">lines</a> .
conf.int	determines whether pointwise confidence intervals will be plotted. Passed over to <a href="#">plot.survfit</a> .
mark.time	controls the labeling of the curves. Passed over to <a href="#">plot.survfit</a> .
lwd	line width of the KM curve.

xlab	x label.
ylab	y label.
...	other arguments are passed over to <a href="#">plot.survfit</a> (default method) or <a href="#">plot.default</a> (for classes <code>pwexpm</code> , <code>boot.pwexpm</code> ).

### Details

For the default method, this a wrapper of [plot.survfit](#) function to plot right censoring data.

For class `pwexpm`, parameters in `...` are passed to [plot.default](#) 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.pwexpm`, parameters in `...` are passed to [plot.default](#) 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.

`plot` method for classes `pwexpm` and `boot.pwexpm` behaves the same as `plot_survival`, except that it also plots the raw event data (by KM curve) by default.

### Value

No return value.

### Author(s)

Tianchen Xu <zjph602xutianchen@gmail.com>

### See Also

[plot\\_event](#)

### Examples

```
event_dist <- function(n)rpwexpm(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'))

plot_survival(dat$followT, dat$event, xlim=c(0,40))

fit_res <- pwexpm(Surv(followT, event), data = dat, nbreak = 2)
plot_survival(fit_res, col='red', lwd=3, breakpoint_par = list(col='grey', lwd=2.5))
```

---

predict *Predict Events for Piecewise Exponential (PWE) Model*

---

### Description

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

### Usage

```
## S3 method for class 'pwexpm'
predict(object, cut_indicator=NULL, analysis_time, censor_model=NULL,
        n_each=100, future_rand=NULL, seed=1818, ...)
## S3 method for class 'boot.pwexpm'
predict(object, cut_indicator=NULL, analysis_time, censor_model=NULL,
        n_each=10, future_rand=NULL, seed=1818, ...)
```

### Arguments

object	a <a href="#">pwexpm</a> or <a href="#">boot.pwexpm</a> 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 <code>event_model/event_model_boot</code> . 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 <a href="#">pwexpm</a> (or <a href="#">boot.pwexpm</a> ). It is the censoring model 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 <b>future</b> rand rate and <b>future</b> total number of samples to be randomized by <code>list(rand_rate= , total_sample= )</code> or specify the <b>future</b> number of randomization each month by <code>list(n_rand= )</code> . See details.
seed	a random seed. Do not set seed if <code>seed=NULL</code> .
...	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 randomization rate=10pt/month and total sample size=1000 by list(rand\_rate=10, total\_sample=1000) 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 object of class "predict.pwexpm" or "predict.boot.pwexpm" is a list containing the following components:

event_fun	number of events vs. time curve function in each bootstrap.
event_model	the event model for the primary endpoint.
censor_model	the censoring model for drop-out and death.
nsim	the number of repeated bootstrapping. nsim=1 for non-bootstrapped model.
bootstrap	a logical value indicating if the event_model is a bootstrapped model.
para	the parameters used to conduct the prediction procedure.

This returned object should be used in [plot\\_event](#) function for summarizing its result.

### Author(s)

Tianchen Xu <zjph602xutianchen@gmail.com>

### See Also

[plot\\_event](#)

### Examples

```
set.seed(1818)
event_dist <- function(n)rpwexpm(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 <- pwexpm(Surv(followT, event), data=train, nbreak = 1)
fit_res_boot <- boot.pwexpm(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 <- pwexpm_fit(train$followT, drop_indicator, nbreak = 0)
fit_res_censor_boot <- boot.pwexpm(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=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(0,500))
plot_event(predicted_boot, eval_at = seq(40,90,5), CI_par = list(lty=3, lwd=2))

plot_event(train$followT_abs, train$event, xswitch = TRUE, ylim=c(0,69), xlim=c(0,400))
plot_event(predicted_boot, xswitch = TRUE, eval_at = seq(250,400,50))

```

pwexpm

*Fit a PWE Model with Formula***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**

```
pwexpm(Surv, data, breakpoint=NULL, nbreak=0, exclude_int=NULL, min_pt_tail=5,
       max_set=10000, seed=1818, trace=FALSE, optimizer='mle', tol=1e-4)
```

**Arguments**

Surv	a <a href="#">Surv</a> object indicating event time and status.
data	a data frame in which to interpret the variables named in the Surv argument.
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. Do not set seed if seed=NULL.
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 object of class "pwexpm" is a list containing the following components:

<code>brk</code>	estimated breakpoints.
<code>lam</code>	estimated piecewise hazard rates.
<code>logLik</code>	the log-likelihood of the model.
<code>AIC</code>	the Akaike information criterion of the model.
<code>BIC</code>	the Bayesian information criterion of the model.
<code>para</code>	the parameters used to estimate the model.

The generic accessor functions `AIC`, `BIC`, `logLik` can be used to extract various useful statistics from `pwexpm`.

## Author(s)

Tianchen Xu <[zjph602xutianchen@gmail.com](mailto:zjph602xutianchen@gmail.com)>

## References

Muller, H. G., & Wang, J. L. (1994). Hazard rate estimation under random censoring with varying kernels and bandwidths. *Biometrics*, 61-76.

## See Also

[pwexpm\\_fit](#), [boot.pwexpm](#), [cv.pwexpm](#)

**Examples**

```

event_dist <- function(n)rpwexpm(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 <- pwexpm(Surv(followT, event), data=train, breakpoint = c(5,14))
fit_a1 <- pwexpm(Surv(followT, event), data=train, nbreak = 2, breakpoint = 14)
fit_b0 <- pwexpm(Surv(followT, event), data=train, nbreak = 0)
fit_b1 <- pwexpm(Surv(followT, event), data=train, nbreak = 1)
fit_b2 <- pwexpm(Surv(followT, event), data=train, nbreak = 2)

```

---

pwexpm\_fit

*Fit a PWE model with Time and Event Vectors*

---

**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**

```

pwexpm_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) <sup>0.2</sup> ).
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. Do not set seed if seed=NULL.

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 $(\max(\text{time}) - \min(\text{time})) * \text{tol}$ . 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  $\text{nbreak} = \text{ceiling}(8 * (\# \text{ unique events})^{0.2})$ . 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  $\text{exclude\_int} = c(a, b)$  (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 object of class "pwexpm" is a list containing the following components:

brk	estimated breakpoints.
lam	estimated piecewise hazard rates.
logLik	the log-likelihood of the model.
AIC	the Akaike information criterion of the model.
BIC	the Bayesian information criterion of the model.
para	the parameters used to estimate the model.

The generic accessor functions AIC, BIC, logLik can be used to extract various useful statistics from pwexpm. The plot function can be used to make a simple plot for pwexpm.

### Author(s)

Tianchen Xu <zjph602xutianchen@gmail.com>

### References

Muller, H. G., & Wang, J. L. (1994). Hazard rate estimation under random censoring with varying kernels and bandwidths. *Biometrics*, 61-76.

**See Also**

[pwexpm](#), [boot.pwexpm](#), [cv.pwexpm](#)

**Examples**

```

event_dist <- function(n)rpwexpm(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 <- pwexpm_fit(train$followT, train$event, breakpoint = c(5,14))
fit_a1 <- pwexpm_fit(train$followT, train$event, nbreak = 2, breakpoint = c(14))
fit_b0 <- pwexpm_fit(train$followT, train$event, nbreak = 0)
fit_b1 <- pwexpm_fit(train$followT, train$event, nbreak = 1)
fit_b2 <- pwexpm_fit(train$followT, train$event, nbreak = 2)

```

---

pwe\_data

*Piece-wise Exponential Example Data*

---

**Description**

This is a simulated clinical data with piece-wise exponential distributed endpoint.

**Usage**

```

pwe_data
  data(pwe_data, package="PwePred")

```

**Format**

ID:	id
randT:	the randomization time for each subject
eventT:	the event time
dropT:	the drop-out time
censor_reason:	If a subject is censored, censor_reason shows the type of censoring (i.e., drop_out, death)
event:	indicates whether the primary event occurred at the end of follow-up with \$0\$ for censoring and \$1\$ for occ
followT:	the follow-up time, which is the minimum value of eventT, dropT. In real-world datasets, this is the observa
followT_abs:	the sum of randT and followT

**Note**

Here we utilize the `simdata` function to create a simple example dataset with the following characteristics:

- randomization rate is defined as 20 subjects per month (`rand_rate = 20`), and total sample size is 1000 (`total_sample = 1000`);
- the primary endpoint (event) follows a PWE distribution with monthly hazard rates of 0.1 for  $t < 5$  months, 0.01 for  $5 \leq t < 14$  months, 0.2 for  $t \geq 14$  months (defined by `myevent_dist` function);
- the drop-out follows an exponential distribution with a drop-out probability of 3%/month (`drop_rate = 0.03`) (equivalently, a monthly drop-out hazard rate is  $-\log(1-0.03)=0.0304$ );
- the `add_column` argument requests additional variables to be included in the dataset.

```
set.seed(1818)
myevent_dist <- function(n)rpwexpm(n, rate = c(0.1, 0.01, 0.2), breakpoint = c(5,14))
pwe_data <- simdata(rand_rate = 20, total_sample = 1000, drop_rate = 0.03, advanced_dist
= list(event_dist = myevent_dist), add_column = c('censor_reason', 'event', 'followT',
'followT_abs'))
```

---

`pwe_data_robust_tail` *Piece-wise Exponential Example Data with 'Abnormal' Tail*

---

**Description**

This is a simulated clinical data.

**Usage**

```
pwe_data_robust_tail
  data(pwe_data_robust_tail, package="PwePred")
```

**Format**

```
Survival.time: time
Status:        1 for events, 0 for censoring
```

---

ruppert_ipd	<i>OS Curves from Ruppert (2020)</i>
-------------	--------------------------------------

---

## Description

OS Curves for IPI 4-5 (red curve) from Figure 2A in Ruppert (2020)

## Usage

```
ruppert_ipd
  data(ruppert_ipd, package="PwePred")
```

## Format

```
Survival.time: time
Status:        1 for events, 0 for censoring
```

## Note

We digitize the OS curves for IPI 4-5 (red curve) from Figure 2A in Ruppert (2020), using Web-PlotDigitizer (Rohatgi A (2024)). Subsequently, we reconstruct individual subject data (IPD) using the **IPDfromKM** package (Liu (2021)).

## References

Ruppert AS, Dixon JG, Salles G, Wall A, Cunningham D, Poeschel V, Haioun C, Tilly H, Ghesquieres H, Ziepert M, et al. (2020). "International Prognostic Indices in Diffuse Large B-cell Lymphoma: A Comparison of IPI, R-IPI, and NCCN-IPI." *Blood, The Journal of the American Society of Hematology*, 135(23), 2041-2048.

Rohatgi A (2024). "Webplotdigitizer: Version 4.7." URL: <https://automeris.io/WebPlotDigitizer>.

Liu N, Zhou Y, Lee JJ (2021). "IPDfromKM: Reconstruct Individual Patient Data from Published Kaplan-Meier Survival Curves." *BMC medical Research Methodology*, 21(1), 111.

---

simdata	<i>Simulate Survival Data</i>
---------	-------------------------------

---

**Description**

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

**Usage**

```
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','event'),
        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: <ul style="list-style-type: none"> <li>• 'eventT_abs': absolute event time from the beginning of the trial (=eventT+randT)</li> <li>• 'dropT_abs': absolute drop-out time from the beginning of the trial (=dropT+randT)</li> <li>• 'deathT_abs': absolute death time from the beginning of the trial (=deathT+randT)</li> <li>• 'censor': censoring (drop-out or death) indicator</li> <li>• 'event': event indicator</li> <li>• 'censor_reason': censoring reason ('drop_out','death','never_event'(eventT=inf))</li> <li>• 'followT': follow-up time (true observed time) from randT</li> <li>• 'followT_abs': absolute follow-up time from the beginning of the trial (=followT+randT)</li> </ul>

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

See webpage <https://zjph602xtc.github.io/PWEXP/> for a diagram illustration of the relationship between returned variables.

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  $-\log(1 - \text{drop\_rate})$  (or say,  $\text{drop\_rate} = 1 - e^{-\text{hazardrate}}$ ).

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 subgroups. 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 search for `event_lambda`, `drop_rate`, `death_lambda` to generate a exp distribution; if they are also missing, then the corresponding variable will not be generated.

## 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**

[rpwexpm](#), [rpwexpm\\_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')

# One group. Event follows a piecewise exponential distribution; drop-out follows
# a Weibull; death follows a exponential.
dist_trt <- function(n)rpwexpm(n, rate=c(0.01, 0.05, 0.01), breakpoint = c(30,60))
dist_placebo <- function(n)rpwexpm(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 calander 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 <a href="#">simdata</a> .

strata	a character vector of the names of strata in groups (e.g., c('young', 'old')). See <a href="#">simdata</a> .
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 <a href="#">simdata</a> .
event_lambda	the hazard rate of the primary endpoint (event). The value will be recycled if the length is less than needed. See <a href="#">simdata</a> .
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 <a href="#">simdata</a> .
death_lambda	(optional) the hazard rate of death. The value will be recycled if the length is less than needed. See <a href="#">simdata</a> .
n_rand	(required when rand_rate=NULL) a vector of the number of randomization each month; can be non-integers. See <a href="#">simdata</a> .
rand_rate	(required when n_rand=NULL) the randomization rate (patients/month; can be non-integer). See <a href="#">simdata</a> .
total_sample	(required when n_rand=NULL) total scheduled sample size. See <a href="#">simdata</a> .
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 <a href="#">simdata</a> .
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. Do not set seed if seed=NULL.

## 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 list contains some of the data frames:

T_all	Simulation result for the whole population
T_by_group	Simulation result by arms
T_by_strata	Simulation result by strata
T_by_group_strata	Simulation result by arms and strata

Each data frame contains some of these columns:

at	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
group	group indicator
strata	stratum indicator
analysis_time	time from the first randomization
analysis_time_c	calendar time from the first randomization
event	number of events
subjects	number of enrolled subjects
functions in stat	function names specified in the stat argument

**Note**

event\_lambda, drop\_rate, death\_lambda can be 0, which means the corresponding subgroup will have no events, drop-out, or deaths.

**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 piecewise exponential event generation function
myevent_dist_trt <- function(n)rpwexpm(n, rate=c(0.1, 0.01, 0.2)*0.7, breakpoint=c(5,14))
myevent_dist_con <- function(n)rpwexpm(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

## \* PwePred

- pwe\_data, 26
  - pwe\_data\_robust\_tail, 27
  - ruppert\_ipd, 28
- abline, 18, 19
- boot.pwexpm, 2, 5, 18, 20, 23, 26
- boot.pwexpm\_fit, 3, 4
- coef.pwexpm, 5
- conditional piecewise exponential, 6
- confint.boot.pwexpm, 8
- cut\_dat, 8
- cv.pwexpm, 10, 12, 23, 26
- cv.pwexpm\_fit, 11, 11
- dpwexpm, 7
- dpwexpm (piecewise exponential), 13
- lines, 16, 18, 19
- piecewise exponential, 13
- plot.boot.pwexpm (plot\_survival), 18
- plot.cv.pwexpm (cv.pwexpm), 10
- plot.default, 16, 19
- plot.predict.boot.pwexpm (plot\_event), 15
- plot.predict.pwexpm (plot\_event), 15
- plot.pwexpm (plot\_survival), 18
- plot.survfit, 18, 19
- plot\_event, 15, 19, 21
- plot\_survival, 17, 18
- ppwexpm, 7
- ppwexpm (piecewise exponential), 13
- ppwexpm\_conditional, 14
- ppwexpm\_conditional (conditional piecewise exponential), 6
- predict, 20
- predict.boot.pwexpm, 15
- predict.pwexpm, 15
- pwe\_data, 26
- pwe\_data\_robust\_tail, 27
- pwexpm, 2, 3, 10, 18, 20, 22, 26
- pwexpm\_fit, 4, 12, 23, 24
- qpwexpm, 7
- qpwexpm (piecewise exponential), 13
- qpwexpm\_conditional, 14
- qpwexpm\_conditional (conditional piecewise exponential), 6
- rpwexpm, 7, 31
- rpwexpm (piecewise exponential), 13
- rpwexpm\_conditional, 14, 31
- rpwexpm\_conditional (conditional piecewise exponential), 6
- ruppert\_ipd, 28
- sim\_followup, 32
- simdata, 29, 32–34
- summary.boot.pwexpm (boot.pwexpm), 2
- summary.predict.boot.pwexpm (predict), 20
- summary.predict.pwexpm (predict), 20
- summary.pwexpm (pwexpm), 22
- Surv, 2, 10, 22