Package ‘dynamichazard’

April 5, 2019

Type Package
Title Dynamic Hazard Models using State Space Models
Version 0.6.5
Description Contains functions that lets you fit dynamic hazard models using state space models. The first implemented model is described in Fahrmeir (1992) <doi:10.1080/01621459.1992.10475232> and Fahrmeir (1994) <doi:10.1093/biomet/81.2.317>. Extensions hereof are available where the Extended Kalman filter is replaced by an unscented Kalman filter and other options including particle filters. The implemented particle filters support more general state space models.
License GPL-2
LazyData TRUE
LinkingTo Rcpp, RcppArmadillo, nloptr (>= 1.2.0)
Imports parallel, Rcpp (>= 0.12.6), boot, nloptr (>= 1.2.0)
Depends R (>= 3.3.0), stats, graphics, utils, survival
RoxygenNote 6.1.1
Suggests testthat, knitr, rmarkdown, timereg, captioner, biglm, htttr, mgcv, shiny, formatR, R.rsp, speedglm, dichromat, colorspace, plyr, gsl, mvtnorm
VignetteBuilder knitr, R.rsp
BugReports https://github.com/boenecd/dynamichazard/issues
SystemRequirements C++11
URL https://github.com/boenecd/dynamichazard
Encoding UTF-8
NeedsCompilation yes
Author Benjamin Christoffersen [cre, aut],
Alan Miller [cph],
Anthony Williams [cph],
Boost developers [cph],
R-core [cph]
Maintainer  Benjamin Christoffersen <boennecd@gmail.com>
Repository  CRAN
Date/Publication  2019-04-05 07:10:03 UTC

R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddFixed</td>
<td>2</td>
</tr>
<tr>
<td>ddhazard</td>
<td>3</td>
</tr>
<tr>
<td>ddhazard_app</td>
<td>6</td>
</tr>
<tr>
<td>ddhazard_boot</td>
<td>6</td>
</tr>
<tr>
<td>ddhazard_control</td>
<td>7</td>
</tr>
<tr>
<td>get_cloud_means</td>
<td>9</td>
</tr>
<tr>
<td>get_cloud_quantiles</td>
<td>10</td>
</tr>
<tr>
<td>get_Q_0</td>
<td>10</td>
</tr>
<tr>
<td>get_risk_obj</td>
<td>11</td>
</tr>
<tr>
<td>get_survival_case_weights_and_data</td>
<td>12</td>
</tr>
<tr>
<td>hatvalues.ddhazard</td>
<td>14</td>
</tr>
<tr>
<td>hds</td>
<td>15</td>
</tr>
<tr>
<td>logLik.ddhazard</td>
<td>16</td>
</tr>
<tr>
<td>logLik.PF_EM</td>
<td>17</td>
</tr>
<tr>
<td>PF_control</td>
<td>18</td>
</tr>
<tr>
<td>PF_EM</td>
<td>20</td>
</tr>
<tr>
<td>PF_forward_filter</td>
<td>26</td>
</tr>
<tr>
<td>PF_get_score_n_hess</td>
<td>29</td>
</tr>
<tr>
<td>plot.ddhazard</td>
<td>31</td>
</tr>
<tr>
<td>plot.ddhazard_space_errors</td>
<td>32</td>
</tr>
<tr>
<td>plot.ddsurvcurve</td>
<td>33</td>
</tr>
<tr>
<td>plot.PF_clouds</td>
<td>36</td>
</tr>
<tr>
<td>plot.PF_EM</td>
<td>37</td>
</tr>
<tr>
<td>predict.ddhazard</td>
<td>37</td>
</tr>
<tr>
<td>print.ddhazard_boot</td>
<td>39</td>
</tr>
<tr>
<td>print.summary.ddhazard</td>
<td>40</td>
</tr>
<tr>
<td>residuals.ddhazard</td>
<td>40</td>
</tr>
<tr>
<td>static_glm</td>
<td>42</td>
</tr>
</tbody>
</table>

Index  44

ddFixed  Auxiliary Functions for Fixed Effects

Description

Functions used in formula of ddhazard for time-invariant effects. ddFixed_intercept is only used for the intercept.
ddhazard

Usage

ddFixed(object)

ddFixed_intercept(random_intercept = FALSE)

Arguments

object expression that would be used in formula. E.g. x or poly(x, degree = 3).

random_intercept

TRUE if a zero mean time-varying process should be included as an additional term. Only relevant in stationary models. See the type argument in PF_EM.

Examples

# we can get a time-invariant effect of `x1` by
## Not run:
ddhazard(Surv(stop, event) ~ ddFixed(x1), data)

## End(Not run)

# all of the calls below will yield the same result with a time-invariant intercept:
## Not run:
ddhazard(Surv(stop, event) ~ ddFixed_intercept() + x1, data)

ddhazard(Surv(stop, event) ~ -1 + ddFixed_intercept() + x1, data)

## End(Not run)

----------

ddhazard Fitting Dynamic Hazard Models

Description

Function to fit dynamic hazard models using state space models.

Usage

ddhazard(formula, data, model = "logit", by, max_T, id, a_0, Q_0,
Q = Q_0, order = 1, weights, control = ddhazard_control(),
verbose = F)

Arguments

formula coxph like formula with Surv(tstart, tstop, event) on the left hand side of ~

data data.frame or environment containing the outcome and covariates.
model co "logit", "cloglog", or "exponential" for respectively the logistic link function with discrete outcomes, the inverse cloglog link function with discrete outcomes, or for the continuous time model with piecewise constant exponentially distributed arrival times.

by interval length of the bins in which parameters are fixed.

max_T end of the last interval interval.

id vector of ids for each row of the in the design matrix.
a_θ vector for the initial coefficient vector for the first iteration (optional). Default is estimates from static model (see static_glm).

Q_θ covariance matrix for the prior distribution.

Q initial covariance matrix for the state equation.

order order of the random walk.

weights weights to use if e.g. a skewed sample is used.

control list of control variables from dhdhazard_control.

verbose TRUE if you want status messages during execution.

Details

This function can be used to estimate survival models where the regression parameters follows a given order random walk. The order is specified by the order argument. 1. and 2. order random walks is implemented. The regression parameters are updated at time by, 2by, ..., max_T. See the vignette("dhdhazard", "dynamichazard") for details.

All filter methods needs a state covariance matrix Q_θ and state vector a_θ. An estimate from a time-invariant model is used for a_θ if it is not supplied (the same model you would get from static_glm). A diagonal matrix with large entries is recommended for Q_θ. What is large depends on the data set and model. Further, a covariance matrix for the first iteration Q is needed. The Q and a_θ are estimated with an EM-algorithm.

The model is specified through the model argument. The discrete outcome models are where outcomes are binned into the intervals. Be aware that there can be "loss" of information due to binning if outcomes are not discrete to start with. It is key for these models that the id argument is provided if individuals in the data set have time-varying covariates. The the exponential model use a piecewise constant exponential distribution for the arrival times where there is no "loss" information due to binning. Though, one of the assumptions of the model is not satisfied if outcomes are only observed in discrete time intervals.

It is recommended to see the Shiny app demo for this function by calling dhdhazard_app().

Value

A list with class dhdhazard. The list contains

- formula the passed formula.
- call the matched call.
- state_vecs 2D matrix with the estimated state vectors (regression parameters) in each bin.
- state_vars 3D array with smoothed variance estimates for each state vector.
**lag_one_cov** 3D array with lagged correlation matrix for each for each change in the state vector. Only present when the model is logit and the method is EKF.

**n_risk** the number of observations in each interval.

**times** the interval borders.

**risk_set** the object from `get_risk_obj` if saved.

**data** the data argument if saved.

**weights** weights used in estimation if saved.

**id** ids used to match rows in data to individuals.

**order** order of the random walk.

**F_** matrix which map from one state vector to the next.

**method** method used in the E-step.

**est.Q_0** TRUE if Q_0 was estimated in the EM-algorithm.

**family** Rcpp Module with C++ functions used for estimation given the model argument.

**discrete_hazard_func** the hazard function corresponding to the model argument.

**terms** the terms object used.

**has_fixed_intercept** TRUE if the model has a time-invariant intercept.

**xlev** a record of the levels of the factors used in fitting.

**References**


**See Also**

`plot`, `residuals`, `predict`, `static_glm`, `ddhazard_app`, `ddhazard_boot`

**Examples**

```r
# example with first order model
library(dynamichazard)
fit <- ddhazard(
  Surv(time, status == 2) ~ log(bili), pbc, id = pbc$id, max_T = 3600,
  Q_0 = diag(1, 2), Q = diag(1e-4, 2), by = 50,
  control = ddhazard_control(method = "GMA"))
plot(fit)

# example with second order model
fit <- ddhazard(
  Surv(time, status == 2) ~ log(bili), pbc, id = pbc$id, max_T = 3600,
  Q_0 = diag(1, 4), Q = diag(1e-4, 2), by = 50,
```
ddhazard_app

**Description**

ddhazard_app runs a shiny app with demonstration of models.

**Usage**

ddhazard_app(quietly = F, ...)

**Arguments**

- **quietly**
  - TRUE if no messages should be printed when the app is run.
- **...**
  - starting values for the shiny app.

**Details**

Runs a shiny app where you try different model specifications on simulated data.

**Examples**

```r
## Not run:
dynamichazard::ddhazard_app()
dynamichazard::ddhazard_app(seed = 1, more_options = TRUE)
## End(Not run)
```

ddhazard_boot

**Description**

See the vignette vignette("Bootstrap_illustration", "dynamichazard"). The do_stratify_with_event may be useful when either cases or non-cases are very rare to ensure that the model estimation succeeds.

**Usage**

```r
ddhazard_boot(ddhazard_fit, strata, unique_id, R = 100,
  do_stratify_with_event = F, do_sample_weights = F,
  LRs = ddhazard_fit$control$LR * 2^(-4), print_errors = F)
```
Arguments

- `ddhazard_fit` returned object from a `ddhazard` call.
- `strata` strata to sample within. These need to be on an individual by individual basis and not rows in the design matrix.
- `unique_id` unique ids where entries match entries of `strata`.
- `R` number of bootstrap estimates.
- `do_stratify_with_event` TRUE if sampling should be by strata of whether the individual has an event. An interaction factor will be made if `strata` is provided.
- `do_sample_weights` TRUE if weights should be sampled instead of individuals.
- `LRs` learning rates in decreasing order which will be used to estimate the model.
- `print_errors` TRUE if errors should be printed when estimations fails.

Value

An object like from the `boot` function.

See Also

`ddhazard, plot`

Examples

```r
## Not run:
library(dynamichazard)
set.seed(56219373)
fit <- ddhazard(
  Surv(time, status == 2) ~ log(bili), pbc, id = pbc$id, max_T = 3000,
  Q.0 = diag(1, 2), Q = diag(1e-4, 2), by = 100,
  control = ddhazard_control(method = "GMA"))
bt <- ddhazard_boot(fit, R = 999)
plot(fit, ddhazard_boot = bt, level = .9)

## End(Not run)
```
Usage

ddhazard_control(kappa = NULL, alpha = 1, beta = 0, NR_eps = NULL,
LR = 1, n_max = 10^2, eps = 0.001, est_Q_0 = F, method = "EKF",
save_risk_set = T, save_data = T, eps_fixed_parems = 1e-04,
fixed_parems_start = NULL,
q_0_term_for_fixed_e_step = NULL,
permu = if (!is.null(method)) method == "SMA" else F,
posterior_version = "cholesky", GMA_max_rep = 25,
GMA_NR_eps = 1e-04, est_a_0 = TRUE, ...)

Arguments

kappa hyper parameter κ in the unscented Kalman Filter.
alpha hyper parameter α in the unscented Kalman Filter.
beta hyper parameter β in the unscented Kalman Filter.
NR_eps tolerance for the Extended Kalman filter. Default is NULL which means that no extra iteration is made in the correction step.
LR learning rate.
n_max maximum number of iteration in the EM-algorithm.
eps tolerance parameter for the EM-algorithm
est_Q_0 TRUE if you want the EM-algorithm to estimate Q_0. Default is FALSE.
method set to the method to use in the E-step. Either "EKF" for the Extended Kalman Filter, "UKF" for the Unscented Kalman Filter, "SMA" for the sequential posterior mode approximation method or "GMA" for the global mode approximation method. "EKF" is the default.
save_risk_set TRUE if you want to save the list from get_risk_obj used to estimate the model. It may be needed for later calls to e.g., residuals, plot and loglike.
save_data TRUE if you want to keep the data argument. It may be needed for later calls to e.g., residuals, plot and loglike.
eps_fixed_parems tolerance used in the M-step of the Fisher’s scoring algorithm for the fixed effects
fixed_parems_start starting value for fixed terms.
n_threads maximum number of threads to use.
denom_term term added to denominators in either the EKF or UKF.
fixed_terms_method the method used to estimate the fixed effects. Either 'M_step' or 'E_step' for estimation in the M-step or E-step respectively.
Q_0_term_for_fixed_E_step the diagonal value of the initial covariance matrix, Q_0, for the fixed effects if fixed effects are estimated in the E-step.
get_cloud_means

permu  TRUE if the risk sets should be permutated before computation. This is TRUE by default for posterior mode approximation method and FALSE for all other methods.

posterior_version  the implementation version of the posterior approximation method. Either "woodbury" or "cholesky".

GMA_max_rep  maximum number of iterations in the correction step if method = 'GMA'.

GMA_NR_eps  tolerance for the convergence criteria for the relative change in the norm of the coefficients in the correction step if method = 'GMA'.

est_a_0  FALSE if the starting value of the state model should be fixed.

...  additional undocumented arguments.

Value

A list with components named as the arguments.

See Also

ddhazard

get_cloud_means  Compute Mean Estimates from Particle Cloud

Description

Computes the estimated means from a particle cloud.

Usage

get_cloud_means(object, ...)

## S3 method for class 'PF_EM'
get_cloud_means(object, ...)

## S3 method for class 'PF_clouds'
get_cloud_means(object, cov_index = NULL,
    type = c("smoothed_clouds", "forward_clouds", "backward_clouds"), ...)

Arguments

object  object with class PF_EM or PF_clouds.

...  named arguments to pass to the PF_clouds method.

cov_index  integer vector with indices of the random effect to include.

type  character with the type of cloud to compute means for.

Value

A matrix which rows are time indices and columns are random effect indices.
get_cloud_quantiles  

*Compute Quantile Estimates from Particle Cloud*

### Description

Computes the estimated quantiles from a particle cloud.

### Usage

```r
get_cloud_quantiles(object, ...)  
## S3 method for class 'PF_EM'
get_cloud_quantiles(object, ...)  
## S3 method for class 'PF_clouds'
get_cloud_quantiles(object, cov_index = NULL, 
qlvls = c(0.05, 0.5, 0.95), type = c("smoothed_clouds", 
"forward_clouds", "backward_clouds"), ...)
```

### Arguments

- **object**: object with class `PF_EM` or `PF_clouds`.
- **...**: named arguments to pass to the `PF_clouds` method.
- **cov_index**: integer vector with indices of the random effect to include.
- **qlvls**: numeric vector with values in \([0, 1]\) with the quantiles to compute.
- **type**: character with the type of cloud to compute quantiles for.

### Value

A 3 dimensional array where the first dimension is the quantiles, the second dimension is the random effect, and the third dimension is the time.

---

get_Q_0  

*Compute Time-Invariant Covariance Matrix*

### Description

Computes the invariant covariance matrix for a vector autoregression model.

### Usage

```r
get_Q_0(Qmat, Fmat)
```
get_risk_obj

Arguments

Qmat  covariance matrix in transition density.
Fmat  coefficients in transition density.

Value

The invariant covariance matrix.

Examples

Fmat <- matrix(c(.8, .4, .1, .5), 2, 2)
Qmat <- matrix(c(1, .5, .5, 2), 2)
x1 <- get_Q_0(Qmat = Qmat, Fmat = Fmat)
x2 <- Qmat
for(i in 1:101)
  x2 <- tcrossprod(Fmat %*% x2, Fmat) + Qmat
stopifnot(isTRUE(all.equal(x1, x2)))

get_risk_obj  Risk Set on an Equidistant Distant Grid

Description

Get the risk set at each bin over an equidistant distant grid.

Usage

get_risk_obj(Y, by, max_T, id, is_for_discrete_model = T,
             n_threads = 1, min_chunk = 5000)

Arguments

Y  vector of outcome variable returned from Surv.
by  length of each bin.
max_T  last observed time.
id  vector with ids where entries match with outcomes Y.
is_for_discrete_model  TRUE if the model outcome is discrete. For example, a logit model is discrete whereas what is is referred to as the exponential model in this package is a dynamic model.
n_threads  set to a value greater than one to use mclapply to find the risk object.
min_chunk  minimum chunk size of ids to use when parallel version is used.
Value

- a list with the following elements
  - **risk_sets**: list of lists with one for each bin. Each of the sub lists have indices that corresponds to the entries of \( Y \) that are at risk in the bin.
  - **min_start**: start time of the first bin.
  - **I_len**: length of each bin.
  - **d**: number of bins.
  - **is_event_in**: indices for which bin an observation \( Y \) is an event. -1 if the individual does not die in any of the bins.
  - **is_for_discrete_model**: value of is_for_discrete_model argument.

Examples

```r
# small toy example with time-varying covariates
dat <- data.frame(
  id = c(1, 1, 2, 2),
  tstart = c(0, 4, 0, 2),
  tstop = c(4, 6, 2, 4),
  event = c(0, 1, 0, 0))

with(dat, get_risk_obj(Surv(tstart, tstop, event), by = 1, max_T = 6, id = id))
```

---

**get_survival_case_weights_and_data**

*Get data.frame for Discrete Time Survival Models*

**Description**

Function used to get `data.frame` with weights for a static fit for survivals.

**Usage**

```r
get_survival_case_weights_and_data(formula, data, by, max_T, id, 
  init_weights, risk_obj, use_weights = T, is_for_discrete_model = T, 
  c_outcome = "Y", c_weights = "weights", c_end_t = "t")
```

**Arguments**

- **formula**: `coxph` like formula with `Surv(tstart, tstop, event)` on the left hand side of `~`.
- **data**: `data.frame` or environment containing the outcome and covariates.
- **by**: interval length of the bins in which parameters are fixed.
get_survival_case_weights_and_data

max_T  end of the last interval interval.

id  vector of ids for each row of the in the design matrix.

init_weights  weights for the rows in data. Useful e.g., with skewed sampling.

risk_obj  a pre-computed result from a get_risk_obj. Will be used to skip some computations.

use_weights  TRUE if weights should be used. See details.

is_for_discrete_model  TRUE if the model is for a discrete hazard model is used like the logistic model.

c_outcome, c_weights, c_end_t  alternative names to use for the added columns described in the return section. Useful if you already have a column named Y, t or weights.

Details

This function is used to get the data.frame for e.g. a glm fit that is comparable to a ddhazard fit in the sense that it is a static version. For example, say that we bin our time periods into (0,1], (1,2] and (2,3]. Next, consider an individual who dies at time 2.5. He should be a control in the first two bins and should be a case in the last bin. Thus the rows in the final data frame for this individual is c(Y = 1, ..., weights = 1) and c(Y = 0, ..., weights = 2) where Y is the outcome, ... is the covariates and weights is the weights for the regression. Consider another individual who does not die and we observe him for all three periods. Thus, he will yield one row with c(Y = 0, ..., weights = 3).

This function use similar logic as the ddhazard for individuals with time varying covariates (see the vignette vignette("ddhazard", "dynamichazard") for details).

If use_weights = FALSE then the two previously mentioned individuals will yield three rows each. The first individual will have c(Y = 0, t = 1, ..., weights = 1), c(Y = 0, t = 2, ..., weights = 1), c(Y = 1, t = 3, ..., weights = 1) while the latter will have three rows c(Y = 0, t = 1, ..., weights = 1), c(Y = 0, t = 2, ..., weights = 1), c(Y = 0, t = 3, ..., weights = 1). This kind of data frame is useful if you want to make a fit with e.g. gam function in the mgcv package as described en Tutz et. al (2016).

Value

Returns a data.frame where the following is added (column names will differ if you specified them): column Y for the binary outcome, column weights for weights of each row and additional rows if applicable. A column t is added for the stop time of the bin if use_weights = FALSE. An element Y with the used Surv object is added if is_for_discrete_model = FALSE.

References


See Also

ddhazard, static_glm
Examples

```r
library(dynamichazard)
# small toy example with time-varying covariates
dat <- data.frame(
  id = c( 1, 1, 2, 2),
  tstart = c( 0, 4, 0, 2),
  tstop = c( 4, 6, 2, 6),
  event = c( 0, 1, 0, 0),
  x1 = c(1.09, 1.29, 0, -1.16))

get_survival_case_weights_and_data(
  Surv(tstart, tstop, event) ~ x1, dat, by = 1, id = dat$id)$X
get_survival_case_weights_and_data(
  Surv(tstart, tstop, event) ~ x1, dat, by = 1, id = dat$id,
  use_weights = FALSE)$X
```

hatvalues.ddhazard   Hat Values for ddhazard Object

Description

Computes hat-"like" values from usual L2 penalized binary regression.

Usage

```r
## S3 method for class 'ddhazard'
hatvalues(model, ...)
```

Arguments

- `model`: a fit from `ddhazard`.
- `...`: not used.

Details

Computes hat-"like" values in each interval for each individual at risk in the interval. See the vignette("ddhazard", "dynamichazard") vignette for details.

Value

A list of matrices. Each matrix has three columns: the hat values, the row number of the original data point and the id the row belongs to.

See Also

`ddhazard`
Examples

library(dynamichazard)
fit <- ddhazard(
  Surv(time, status == 2) ~ log(bili), pbc, id = pbc$id, max_T = 3000,
  Q_0 = diag(1, 2), Q = diag(1e-4, 2), by = 100,
  control = ddhazard_control(method = "GMA"))
hvs <- hatvalues(fit)
head(hvs[[1]])
head(hvs[[2]])

hds

Hard Drive Failures

Description

A data set containing hard drive failures data from Backblaze in the start-stop format used in the survival package.

Usage

hds

Format

A data frame with the following columns:

- **serial_number**: Serial number for the hard disk which the row belongs to.
- **model**: Hard disk model.
- **manufacturer**: Manufacturer of the hard disk model.
- **tstart**, **tstop**: Start and stop times on the SMART 9 attribute scale.
- **fails**: 1 if the hard disk fails at tstop.
- **size_tb**: Hard disk size in terabytes.
- **smart_x**: The raw SMART attribute x value. E.g., smart_12 is the power cycle count.
- **smart_x_bin**: 1 if the SMART attribute x value is non-zero.
- **..._cumsum**: Cumulative sum of the prefix . . .
- **n_fails**: Number of failures in the original data. Hard disk should only fail once but this is not the case in the raw data.
- **n_records**: Number of records in the original source.
- **min_date**, **max_date**: First and last date in the original source.
- **min_hours**, **max_hours**: Smallest and largest value of the SMART 9 attribute in the original source.
Details

Details about the the SMART attributes can be found on [https://en.wikipedia.org/wiki/S.M.A.R.T.](https://en.wikipedia.org/wiki/S.M.A.R.T.). As stated in the original source:

"Reported stats for the same SMART stat can vary in meaning based on the drive manufacturer and the drive model. Make sure you are comparing apples-to-apples as drive manufacturers don't generally disclose what their specific numbers mean."

There are some notes on [https://en.wikipedia.org/wiki/S.M.A.R.T.](https://en.wikipedia.org/wiki/S.M.A.R.T.) regarding which attributes that have vendor specific raw value. Further,

"The values in the files are the values reported by the drives. Sometimes, those values are out of whack. For example, in a few cases the RAW value of SMART 9 (Drive life in hours) reported a value that would make a drive 10+ years old, which was not possible. In other words, it's a good idea to have bounds checks when you process the data."

See this github page for the processing steps [https://github.com/boennecd/backblaze_survival_analysis_prep](https://github.com/boennecd/backblaze_survival_analysis_prep).

Source

Raw data from [https://www.backblaze.com/b2/hard-drive-test-data.html](https://www.backblaze.com/b2/hard-drive-test-data.html). Data have been processed to get a start-stop data.frame format.

---

**logLik.ddhazard**

*Log Likelihood of Mean Path of ddhazard Object*

**Description**

Computes the log likelihood of (a potentially new) data set given the estimated:

\[ E_\theta(\alpha_1|y_{1:d}), E_\theta(\alpha_2|y_{1:d}), ..., E_\theta(\alpha_d|y_{1:d}) \]

of the ddhazard object. Note that this is not the log likelihood of the observed data given the outcome.

**Usage**

```r
## S3 method for class 'ddhazard'
logLik(object, data = NULL, id, ...)```

**Arguments**

- `object` an object of class ddhazard.
- `data` new data to evaluate the likelihood for.
- `id` the individual identifiers as in ddhazard.
- `...` unused.
Examples

```r
library(dynamichazard)
fit <- ddhazard(
  Surv(time, status == 2) ~ log(bili), pbc, id = pbc$id, max_T = 3600,
  Q.0 = diag(1, 2), Q = diag(1e-4, 2), by = 50,
  control = ddhazard_control(method = "GMA"))
logLik(fit)
```

Description

Computes the approximate log-likelihood using the forward filter clouds. See the vignette("Particle_filtering", "dynamichazard") for details.

Usage

```r
# S3 method for class 'PF_EM'
logLik(object, ...)

# S3 method for class 'PF_clouds'
logLik(object, df = NA_real_, nobs = NA_integer_, ...)
```

Arguments

- `object`: an object of class `PF_clouds` or `PF_EM`.
- `...`: unused.
- `df`: degrees of freedom used in the model.
- `nobs`: integer with number of individuals used to estimate the model.

Value

The approximate log-likelihood value given the observed data and set of parameter used when simulating the clouds. An attribute "$P(y_t | y_{1:(t-1)})$" has the $P(y_t | y_{1:(t-1)})$ terms.
PF_control 

Auxiliary for Controlling Particle Fitting

Description

Auxiliary for additional settings with PF_EM.

Usage

```r
PF_control(N_fw_n_bw = NULL, N_smooth = NULL, N_first = NULL,
eps = 0.01, forward_backward_ESS_threshold = NULL,
method = "AUX_normal_approx_w_cloud_mean", n_max = 25,
n_threads = getOption("ddhazard_max_threads"),
smoother = "fearnhead_0_N", Q_tilde = NULL, est_a_0 = TRUE,
N_smooth_final = N_smooth, nu = 0L, covar_fac = -1,
ftol_rel = 1e-08, averaging_start = -1L)
```

Arguments

- `N_fw_n_bw`: number of particles to use in forward and backward filter.
- `N_smooth`: number of particles to use in particle smoother.
- `N_first`: number of particles to use at time 0 and time \( d + 1 \).
- `eps`: convergence threshold in EM method.
- `forward_backward_ESS_threshold`: required effective sample size to not re-sample in the particle filters.
- `method`: method for forward, backward and smoothing filter.
- `n_max`: maximum number of iterations of the EM algorithm.
- `n_threads`: maximum number of threads to use in the computations.
- `smoother`: smoother to use.
- `Q_tilde`: covariance matrix of additional error term to add to the proposal distributions. NULL implies no additional error term.
- `est_a_0`: FALSE if the starting value of the state model should be fixed. Does not apply for type = "VAR".
- `N_smooth_final`: number of particles to sample with replacement from the smoothed particle cloud with `N_smooth` particles using the particles' weights. This causes additional sampling error but decreases the computation time in the M-step.
- `nu`: integer with degrees of freedom to use in the (multivariate) t-distribution used as the proposal distribution. A (multivariate) normal distribution is used if it is zero.
- `covar_fac`: factor to scale the covariance matrix with. Ignored if the values is less than or equal to zero.
- `ftol_rel`: tolerance passed to nloptr in mode approximation.
- `averaging_start`: index to start averaging. Values less than or equal to zero yields no averaging.
Details

The method argument can take the following values

- `bootstrap_filter` for a bootstrap filter.
- `PF_normal_approx_w_cloud_mean` for a particle filter where a Gaussian approximation is used using a Taylor approximation made at the mean for the current particle given the mean of the parent particles and/or mean of the child particles.
- `AUX_normal_approx_w_cloud_mean` for an auxiliary particle filter version of `PF_normal_approx_w_cloud_mean`.
- `PF_normal_approx_w_particles` for a filter similar to `PF_normal_approx_w_cloud_mean` and differs by making a Taylor approximation at a mean given each sampled parent and/or child particle.
- `AUX_normal_approx_w_particles` for an auxiliary particle filter version of `PF_normal_approx_w_particles`.

The smoother argument can take the following values

- `Fearnhead_O_N` for the smoother in Fearnhead, Wyncoll, and Tawn (2010).
- `Brier_O_N_square` for the smoother in Briers, Doucet, and Maskell (2010).

Value

A list with components named as the arguments.

References


See Also

- `PF_EM`
Description

Method to estimate the hyper parameters with an EM algorithm.

Usage

```
PF_EM(formula, data, model = "logit", by, max_T, id, a_0, Q_0, Q,
order = 1, control = PF_control(...), trace = 0, seed = NULL,
type = "RW", fixed = NULL, random = NULL, Fmat, fixed_effects, G,
theta, J, K, psi, phi, ...)```

Arguments

- **formula**: `coxph` like formula with `Surv(tstart, tstop, event)` on the left hand side of `~`.
- **data**: data.frame or environment containing the outcome and covariates.
- **model**: either 'logit' for binary outcomes with the logistic link function, 'cloglog' for binary outcomes with the inverse cloglog link function, or 'exponential' for piecewise constant exponential distributed arrival times.
- **by**: interval length of the bins in which parameters are fixed.
- **max_T**: end of the last interval interval.
- **id**: vector of ids for each row of the in the design matrix.
- **a_0**: vector $a_0$ for the initial coefficient vector for the first iteration (optional). Default is estimates from static model (see `static_glm`).
- **Q_0**: covariance matrix for the prior distribution.
- **Q**: initial covariance matrix for the state equation.
- **order**: order of the random walk.
- **control**: see `PF_control`.
- **trace**: argument to get progress information. Zero will yield no info and larger integer values will yield incrementally more information.
- **seed**: seed to set at the start of every EM iteration. See `set.seed`.
- **fixed**: two-sided formula to be used with random instead of formula. It is of the form `Surv(tstart, tstop, event) ~ x` or `Surv(tstart, tstop, event) ~ 1` for no fixed effects.
- **random**: one-sided formula to be used with fixed instead of formula. It is of the form `~ z`.
- **Fmat**: starting value for $F$ when `type = "VAR"`. See 'Details' in `PF_EM`.
fixed_effects starting values for fixed effects if any. See ddFixed.

parameters for a restricted type = "VAR" model. See the vignette mentioned in 'Details' of PF_EM and the examples linked to in 'See Also'.

... optional way to pass arguments to control.

Details

Estimates a state model of the form

\[ \alpha_t = F \alpha_t + R \epsilon_t, \quad \epsilon_t \sim N(0, Q) \]

where \( F \in \mathbb{R}^{p \times p} \) has full rank, \( \alpha_t \in \mathbb{R}^p \), \( \epsilon_t \in \mathbb{R}^r \), and \( R = (e_{l_1}, e_{l_2}, \ldots, e_{l_r}) \) where \( e_k \) is column from the \( p \) dimensional identity matrix and \( l_1 < l_2 < \ldots < l_r \). The time zero state is drawn from

\[ \alpha_0 \sim N(a_0, Q_0) \]

with \( Q_0 \in \mathbb{R}^{p \times p} \). The latent states, \( \alpha_t \), are related to the output through the linear predictors

\[ \eta_{it} = X_t^T \alpha_t + Z_t \beta \]

where \( X_t \in \mathbb{R}^{n_t \times r} \) and \( Z_t \in \mathbb{R}^{n_t \times c} \) are design matrices and the outcome for an individual \( i \) at time \( t \) is distributed according to an exponential family member given \( \eta_{it} \). \( \beta \) are constant coefficients.

See vignette("Particle_filtering", "dynamichazard") for details.

Value

An object of class PF_EM.

Warning

The function is still under development so the output and API may change.

See Also

PF_forward_filter to get a more precise estimate of the final log-likelihood.

See the examples at https://github.com/boennecd/dynamichazard/tree/master/examples.

Examples

```r
## Not run:
###
# Fit model with lung data set from survival
# Warning: long-ish computation time

library(dynamichazard)
.lung <- lung[!is.na(lung$ph.ecog), ]
```
# standardize
lung$age <- scale(lung$age)

# fit
set.seed(43588155)
pf_fit <- PF_EM(
  Surv(time, status == 2) ~ ddfixed(ph.ecog) + age,
  data = .lung, by = 50, id = 1:nrow(.lung),
  Q.0 = diag(1, 2), Q = diag(.5^2, 2),
  max.T = 800,
  control = PF_control(
    N_fw_n_bw = 500, N_first = 2500, N_smooth = 5000,
    n_max = 50, eps = .001, Q.tilde = diag(.2^2, 2), est.a.0 = FALSE,
    n_threads = max(parallel::detectCores(logical = FALSE), 1))
)

# Plot state vector estimates
plot(pf_fit, cov_index = 1)
plot(pf_fit, cov_index = 2)

# Plot log-likelihood
plot(pf_fit$log_likes)

## End(Not run)
## Not run:

#####
# example with fixed intercept

# prepare data
temp <- subset(pbc, id <= 312, select=c(id, sex, time, status, edema, age))
pbc2 <- tmerge(temp, temp, id=id, death = event(time, status))
pbc2 <- tmerge(pbc2, pbcseq, id=id, albumin = tdc(day, albumin),
  protime = tdc(day, protime), bili = tdc(day, bili))
pbc2 <- pbc2[, c("id", "tstart", "tstop", "death", "sex", "edema",
  "age", "albumin", "protime", "bili")]
pbc2 <- within(pbc2, {
  log_albumin <- log(albumin)
  log_protime <- log(protime)
  log_bili <- log(bili)
})

# standardize
for(c. in c("age", "log_albumin", "log_protime", "log_bili"))
  pbc2[[c.]] <- drop(scale(pbc2[[c.]]))

# fit model with extended Kalman filter
ddfit <- ddhazard(
  Surv(tstart, tstop, death == 2) ~ ddfixed_intercept() + ddfixed(age) +
  ddfixed(edema) + ddfixed(log_albumin) + ddfixed(log_protime) + log_bili,
  pbc2, Q.0 = 100, Q = 1e-2, by = 100, id = pbc2$id,
  model = "exponential", max.T = 3600,
  control = ddhazard_control(eps = 1e-5, NR_eps = 1e-4, n_max = 1e4))
summary(ddfit)
# fit model with particle filter

```r
set.seed(88235076)

# fit model with particle filter
pf_fit <- PF_EM(
    Surv(tstart, tstop, death == 2) ~ ddfixed_intercept() + ddfixed(age) +
    ddfixed(edema) + ddfixed(log_albumin) + ddfixed(log_protime) + log_bili,
    pbc2, Q_0 = 2^2, Q = ddfit$Q * 100, # use estimate from before
    by = 100, id = pbc2$id,
    model = "exponential", max_T = 3600,
    control = PF_control(
        N_fw_n_bw = 500, N_smooth = 2500, N_first = 1000, eps = 1e-3,
        method = "AUX_normal_approx_w_cloud_mean", est_a_0 = FALSE,
        Q_tilde = as.matrix(.^2),
        n_max = 25, # just take a few iterations as an example
        n_threads = max(parallel::detectCores(logical = FALSE), 1)))
```

# compare results

```r
plot(ddfit)
plot(pf_fit)
sqrt(ddfit$Q * 100)
sqrt(pf_fit$Q)
rbind(ddfit$fixed_effects, pf_fit$fixed_effects)
```

## End(Not run)

## Not run:

```r
# simulation example with `random` and `fixed` argument and a restricted model

# g groups with k individuals in each
g <- 3L
k <- 400L

# matrices for state equation
p <- g + 1L
G <- matrix(0, p^2, 2L)
for(i in 1:p)
    G[i + (i - 1L) * p, 1L + (i == p)] <- 1L

theta <- c(.9, .8)
# coefficients in transition density
(F <- matrix(as.vector(G %*% theta), 4L, 4L))

J <- matrix(0, ncol = 2L, nrow = p)
J[-p, 1L] <- J[p, 2L] <- 1
psi <- c(log(c(.3, .1)))

K <- matrix(0, p * (p - 1L) / 2L, 2L)
j <- 0L
for(i in (p - 1L):1L){
    j <- j + 1
    K[j, 2L] <- 1
}
K[K[,] < 1, 1L] <- 1
```
phi <- log(-(c(.8, .3) + 1) / (c(.8, .3) - 1))

V <- diag(exp(dop(I %*% psi)))
C <- diag(1, ncol(V))
C[lower.tri(C)] <- 2/(1 + exp(-drop(K %*% phi))) - 1
C[upper.tri(C)] <- t(C)[upper.tri(C)]
(Q <- V %*% C %*% V)  # covariance matrix in transition density

cov2cor(Q)

Q.0 <- get_Q.0(Q, F.)  # time-invariant covariance matrix
beta <- c(rep(-6, g), 0)  # all groups have the same long run mean intercept

# simulate state variables
set.seed(56219373)
n_periods <- 300L
alphas <- matrix(nrow = n_periods + 1L, ncol = p)
alphas[1L, ] <- rnorm(p) %*% chol(Q.0)
for(i in 1:n_periods + 1L)
  alphas[i, ] <- F. %*% alphas[i - 1L, ] + drop(rnorm(p) %*% chol(Q))
alphas <- t(t(alphas) + beta)

# plot state variables
matplot(alphas, type = "l", lty = 1)

# simulate individuals’ outcome
n_obs <- g * k
df <- lapply(1:n_obs, function(i){
  # find the group
  grp <- (i - 1L) %/% (n_obs / g) + 1L
  # left-censoring
  tstart <- max(0L, sample.int((n_periods - 1L) * 2L, 1) - n_periods + 1L)
  # covariates
  x <- c(1, rnorm(1))
  # outcome (stop time and event indicator)
  osa <- NULL
  oso <- NULL
  osx <- NULL
  y <- FALSE
  for(tstop in (tstart + 1L):n_periods){
    sigmoid <- 1 / (1 + exp(- drop(x %*% alphas[tstop + 1L, c(grp, p)])))
    if(sigmoid > runif(1)){
      y <- TRUE
      break
    }
  }
  if(.01 > runif(1L) & tstop < n_periods){
    # sample new covariate
    osa <- c(osa, tstart)
    tstart <- tstop
    oso <- c(oso, tstop)
  }
})
osx <- c(osx, x[2])
x[2] <- rnorm(1)
}
)
cbind(
  tstart = c(osa, tstart), tstop = c(oso, tstop),
  x = c(osx, x[2]), y = c(rep(FALSE, length(osa)), y), grp = grp,
  id = i)
)
df <- data.frame(do.call(rbind, df))
df$grp <- factor(df$grp)

# fit model. Start with "cheap" iterations
fit <- PF_EM(
  fixed = Surv(tstart, tstop, y) ~ x, random = ~ grp + x - 1,
  data = df, model = "logit", by = 1L, max_T = max(df$tstop),
  Q_0 = diag(1.5^2, p), id = df$id, type = "VAR",
  G = G, theta = c(.5, .5), J = J, psi = log(c(.1, .1)),
  K = K, phi = log(-(c(.4, 0) / (c(.4, 0) - 1))},
  control = PF_control(            
    N_fw_n_bw = 100L, N_smooth = 100L, N_first = 500L,
    method = "AUX_normal_approx_w_cloud_mean",
    nu = 5L, # sample from multivariate t-distribution
    n_max = 100L, averaging_start = 50L,
    smoother = "Fearnhead_O_N", eps = 1e-4, covar_fac = 1.2,
    n_threads = 4L # depends on your cpu(s)
  ),
  trace = 1L)
plot(fit$log_likes) # log-likelihood approximation at each iterations

# take more iterations with more particles
cl <- fit$call
ctrl <- cl["control"]
ctrl[c("N_fw_n_bw", "N_smooth", "N_first", "n_max", 
  "averaging_start")]<- list(500L, 2000L, 5000L, 200L, 30L)
cl["control"] <- ctrl
cl[c("phi", "psi", "theta")]<- list(fit$phi, fit$psi, fit$theta)
fit_extra <- eval(cl)
plot(fit_extra$log_likes) # log-likelihood approximation at each iteration

# check estimates
sqrt(diag(fit_extra$Q))

sqrt(diag(Q))
cov2cor(fit_extra$Q)
cov2cor(Q)
fit_extra$F
F.

# plot predicted state variables
for(i in 1:p){
  plot(fit_extra, cov_index = i)
PF_forward_filter

Forward Particle Filter

Description

Functions to only use the forward particle filter. Useful for log-likelihood evaluation though there is an $O(d^2)$ variance of the estimate where $d$ is the number of time periods. The number of particles specified in the control argument has no effect.

The function does not alter the .Random.seed to make sure the same rng.kind is kept after the call. See PF_EM for model details.

Usage

PF_forward_filter(x, N_fw, N_first, ...)  

## S3 method for class 'PF_EM'  
PF_forward_filter(x, N_fw, N_first, seed, ...)  

## S3 method for class 'formula'  
PF_forward_filter(x, N_fw, N_first, data,  
model = "logit", by, max_T, id, a_0, Q_0, Q, fixed_effects,  
control = PF_control(...), seed = NULL, trace = 0, G, theta, J, K,  
psi, phi, type = "RW", Fmat, ...)  

## S3 method for class 'data.frame'  
PF_forward_filter(x, N_fw, N_first, formula,  
model = "logit", by, max_T, id, a_0, Q_0, Q, fixed_effects,  
control = PF_control(...), seed = NULL, trace = 0, fixed = NULL,  
random = NULL, G, theta, J, K, psi, phi, type = "RW", Fmat,  
order = 1, ...)  

Arguments

x

an PF_EM or formula object.

N_fw

number of particles.

N_first

number of time zero particles to draw.

...  

optional way to pass arguments to control.

seed

.globalEnv$.Random.seed to set. Not seed as in set.seed function. Can be used with the .Random.seed returned by PF_EM.

data

data.frame or environment containing the outcome and covariates.
model either 'logit' for binary outcomes with the logistic link function, 'cloglog' for binary outcomes with the inverse clog link function, or 'exponential' for piecewise constant exponential distributed arrival times.

by interval length of the bins in which parameters are fixed.

max_T end of the last interval interval.

id vector of ids for each row of the in the design matrix.

a_0 vector a₀ for the initial coefficient vector for the first iteration (optional). Default is estimates from static model (see static_glm).

Q_0 covariance matrix for the prior distribution.

Q initial covariance matrix for the state equation.

fixed_effects values for the fixed parameters.

control see PF_control.

trace argument to get progress information. Zero will yield no info and larger integer values will yield incrementally more information.

G, theta, J, K, psi, phi parameters for a restricted type = "VAR" model. See the vignette mentioned in 'Details' of PF_EM and the examples linked to in 'See Also'.


Fmat starting value for F when type = "VAR". See 'Details' in PF_EM.

formula coxph like formula with Surv(tstart, tstop, event) on the left hand site of ~.

fixed two-sided formula to be used with random instead of formula. It is of the form Surv(tstart, tstop, event) ~ x or Surv(tstart, tstop, event) ~ - 1 for no fixed effects.

random one-sided formula to be used with fixed instead of formula. It is of the form ~ z.

order order of the random walk.

Value

An object of class PF_clouds.

Methods (by class)

• PF_EM: Forward particle filter using the estimates of an PF_EM call.

• formula: Forward particle filter with formula input.

• data.frame: Forward particle filter with data.frame data input as x instead of data. Can be used with fixed and random argument.

Warning

The function is still under development so the output and API may change.
Examples

```r
## Not run:
is_censored <- c(
  6, 27, 34, 36, 42, 46, 48:51, 51 + c(15, 30:28, 33, 35:37, 39, 40, 42:45))
head_neck_cancer <- data.frame(
  id = 1:96,
  stop = c(
    1, 2, 2, rep(3, 6), 4, 4, rep(5, 8),
    rep(6, 7), 7, 8, 8, 9, 10, 10, 11, 14, 14, 15, 18, 18, 20,
    20, 37, 37, 38, 41, 45, 47, 47,
    2, 2, 3, rep(4, 4), rep(5, 5), rep(6, 5),
    7, 7, 7, 9, 10, 11, 12, 15, 16, 18, 18, 21,
    21, 24, 25, 27, 36, 41, 44, 52, 54, 59, 59, 63, 67, 71, 76),
  event = !(1:96 %in% is_censored),
  group = factor(c(rep(1, 45 + 6), rep(2, 45))))

# fit model
set.seed(61364778)
ctrl <- PF_control(
  N_fw_n_bw = 500, N_smooth = 2500, N_first = 2000,
  n_max = 1, n_threads = max(parallel::detectCores(logical = FALSE), 1),
  eps = 0.001, Q_tilde = as.matrix(.3^2), est_a_0 = FALSE)
pf_fit <- suppressWarnings(
  PF_EM(
    survival::Surv(stop, event) ~ ddFixed(group),
    data = head_neck_cancer, by = 1, Q_0 = 1, Q = 0.1^2, control = ctrl,
    max_T = 30))

# the log-likelihood in the final iteration
(end_log_like <- logLik(pf_fit))

# gives the same
fw_ps <- PF_forward_filter(
  survival::Surv(stop, event) ~ ddFixed(group), N_fw = 500, N_first = 2000,
  data = head_neck_cancer, by = 1, Q_0 = 1, Q = 0.1^2,
  a_0 = pf_fit$a_0, fixed_effects = -0.5370051,
  control = ctrl, max_T = 30, seed = pf_fit$seed)
all.equal(c(end_log_like), c(logLik(fw_ps)))

# will differ since we use different number of particles
fw_ps <- PF_forward_filter(
  survival::Surv(stop, event) ~ ddFixed(group), N_fw = 1000, N_first = 3000,
  data = head_neck_cancer, by = 1, Q_0 = 1, Q = 0.1^2,
  a_0 = pf_fit$a_0, fixed_effects = -0.5370051,
  control = ctrl, max_T = 30, seed = pf_fit$seed)
all.equal(c(end_log_like), c(logLik(fw_ps)))

# will differ since we use the final estimates
fw_ps <- PF_forward_filter(pf_fit, N_fw = 500, N_first = 2000)
all.equal(c(end_log_like), c(logLik(fw_ps)))
```
PF_get_score_n_hess

## Description

Returns a list of functions to approximate the negative observation matrix and score vector.

## Usage

```r
PF_get_score_n_hess(object, debug = FALSE, use_0_n_sq = FALSE)
```

## Arguments

- **object**: object of class `PF_EM`.
- **debug**: TRUE if debug information should be printed to the console.
- **use_0_n_sq**: TRUE if the method from Poyiadjis et al. (2011) should be used.

## Details

The score vector and negative observed information matrix are computed with the (forward) particle filter. This comes at an $O(d^2)$ variance where $d$ is the number of periods. Thus, the approximation may be poor for long series. The score vector can be used to perform stochastic gradient descent. If `use_0_n_sq` is TRUE then the method in Poyiadjis et al. (2011) is used. This may only have a variance which is linear in the number of time periods. However, the present implementation is $O(N^2)$ where $N$ is the number of particles. The method uses a particle filter as in Section 3.1 of Lin et al. (2005). There is no need to call `run_particle_filter` unless one wants a new approximation of the log-likelihood as a separate filter is run with `get_get_score_n_hess` when `use_0_n_sq` is TRUE.

## Value

A list with the following functions as elements

- `run_particle_filter` function to run particle filter as with `PF_forward_filter`.
- `set_parameters` function to set the parameters in the model. The first argument is a vectorized version of $F$ matrix and $Q$ matrix. The second argument is the fixed effect coefficients.
- `set_n_particles` sets the number of particles to use in `run_particle_filter` and `get_get_score_n_hess` when `use_0_n_sq` is TRUE.
- `get_get_score_n_hess` computes the approximate negative observation matrix and score vector. The argument toggles whether the approximate negative observation matrix should be computed. The last particle cloud from `run_particle_filter` is used when `use_0_n_sq` is FALSE.
**Warning**

The function is still under development so the output and API may change.

**References**


**See Also**

See the examples at https://github.com/boennecd/dynamichazard/tree/master/examples.

**Examples**

```r
## Not run:
library(dynamichazard)
.lung <- lung[!is.na(lung$ph.ecog), ]
# standardize
.lung$age <- scale(.lung$age)

set.seed(43588155)
pf_fit <- PF_EM(
  fixed = Surv(time, status == 2) ~ ph.ecog + age,
  random = ~ age, model = "exponential",
  data = .lung, by = 50, id = 1:nrow(.lung),
  Q_0 = diag(1, 2), Q = diag(.5^2, 2), type = "VAR",
  max_T = 800,
  control = PF_control(
    N_fw_n_bw = 250, N_first = 2000, N_smooth = 500, covar_fac = 1.1,
    mu = 6, n_max = 1000L, eps = 1e-5, est_a_0 = FALSE, averaging_start = 100L,
    n_threads = max(parallel::detectCores(logical = FALSE), 1)))

comp_obj <- PF_get_score_n_hess(pf_fit)
comp_obj$set_n_particles(N_fw = 10000L, N_first = 10000L)
comp_obj$run_particle_filter()
(o1 <- comp_obj$get_get_score_n_hess())

# O(N^2) method with lower variance
comp_obj <- PF_get_score_n_hess(pf_fit, use_O_n_sq = TRUE)
comp_obj$set_n_particles(N_fw = 2500L, N_first = 2500L)
(o2 <- comp_obj$get_get_score_n_hess())
```
# approximations may have large variance
o3 <- replicate(10L, {
  runif(1)
  pf_fit$seed <- .Random.seed
  comp_obj <- PF_get_score_n_hess(pf_fit)
  comp_obj$set_n_particles(N_fw = 10000L, N_first = 10000L)
  comp_obj$run_particle_filter()
  comp_obj$get_get_score_n_hess()
}, simplify = FALSE)
sapply(o3, function(x) x$observation$score)
sapply(o3, function(x) sqrt(diag(solve(x$observation$neg_obs_info))))

## End(Not run)

---

**Description**

Plot of estimated state space variables from a `ddhazard` fit.

**Usage**

```r
## S3 method for class 'ddhazard'
plot(x, xlab = "Time", ylab = "Hazard",
     type = "cov", plot_type = "l", cov_index, ylim, col = "black",
     add = F, do_alternate_mfcol = T, level = 0.95, ddhazard_boot, ...)
```

**Arguments**

- `x`: result of `ddhazard` call.
- `xlab`, `ylab`, `ylim`, `col`: arguments to override defaults set in the function.
- `type`: type of plot. Currently, only "cov" is available for plot of the state space parameters.
- `plot_type`: the type argument passed to `plot`.
- `cov_index`: the index (indices) of the state space parameter(s) to plot.
- `add`: FALSE if you want to make a new plot.
- `do_alternate_mfcol`: TRUE if the function should alter `par(mfcol)` in case that `cov_index` has more than one element.
- `level`: level (fraction) for confidence bounds.
- `ddhazard_boot`: object from a `ddhazard_boot` call which confidence bounds will be based on and where bootstrap samples will be printed with a transparent color.
- `...`: arguments passed to `plot.default` or `lines` depending on the value of `add`. 
plot.ddhazard_space_errors

Details

Creates a plot of state variables or adds state variables to a plot with indices cov_index. Pointwise 1.96 std. confidence intervals are provided with the smoothed co-variance matrices from the fit.

Examples

```r
library(dynamichazard)
fit = ddhazard(
  Surv(time, status == 2) ~ log(bili), pbc, id = pbc$id, max_T = 3600,
  Q.0 = diag(1, 2), Q = diag(1e-4, 2), by = 50,
  control = ddhazard_control(method = "GMA")
)
plot(fit)
plot(fit, cov_index = 2)
```

---

**plot.ddhazard_space_errors**

*State Space Error Plot*

Description

Plot function for state space errors from *ddhazard* fit.

Usage

```r
## S3 method for class 'ddhazard_space_errors'
plot(x, mod, cov_index = NA,
     t_index = NA, p_cex = par()$cex * 0.2, pch = 16,
     ylab = "Std. state space error", x_tick_loc = NA, x_tick_mark = NA,
     xlab = "Time", ...)
```

Arguments

- **x**: result of `residuals` with a 'type' argument which yields state space errors.
- **mod**: the *ddhazard* result used in the `residuals` call.
- **cov_index**: the indices of state vector errors to plot. Default is to use all.
- **t_index**: the bin indices to plot. Default is to use all bins.
- **p_cex**: cex argument for the points
- **pch, ylab, xlab**: arguments to override defaults set in the function.
- **x_tick_loc, x_tick_mark**: at and labels arguments passed to `axis`.
- **...**: arguments passed to `plot.default`. 

---
plot.ddsurvcurve

Create and plot survival curves

Description

The function creates a predicted survival curve for a new observation using a estimated ddhazard model from ddhazard. The predicted curve is based on the predicted mean path of the state vector. Thus, the survival curve will not be a "mean" curve due to the non-linear relation between the probability of an event and the state vector.

Usage

## S3 method for class 'ddsurvcurve'
plot(x, y, xlab = "Time", ylab = "Survival",
     ylim, xaxs = "i", yaxs = "i", ...)

## S3 method for class 'ddsurvcurve'
lines(x, col = "Black", lty = 1,
      lwd = par()$lwd, ...)

ddsurvcurve(object, new_data, tstart = ",", tstop = ")

Arguments

- \(x\) a ddsurvcurve object.
- \(y\) not used.
- \(xlab\) xlab passed to plot.
- \(ylab\) ylab passed to plot.
- \(ylim\) ylim passed to plot.
- \(xaxs\) xaxs passed to plot.
- \(yaxs\) yaxs passed to plot.
- \(\ldots\) not used.
- \(col\) col passed to lines.
- \(lty\) lty passed to lines.
- \(lwd\) lwd passed to lines.
- \(object\) a ddhazard object.
- \(new\_data\) a data.frame with the new data for the observation who the survival curve should be for. It can have more rows if tstart and tstop is supplied. The rows need to be consecutive and non-overlapping time intervals.
- \(tstart\) name of the start time column in new_data. It must be on the same time scale as the tstart used in the Surv(tstart, tstop, event) in the formula passed to ddhazard.
- \(tstop\) same as tstart for the stop argument.
Value

`ddsurvcurve` returns an object of class `ddsurvcurve`. It elements are the predicted discrete survival curve, time points for the survival curve, point of the first time period, the call, the discrete probabilities of an event in each interval conditional on survival up to that point, and the name of the distribution family. It should be seen as a plug-in estimate.

Methods (by generic)

- `plot`: method for plotting survival curve.
- `lines`: Method for adding survival curve to a plot.

`plot.ddsurvcurve`

Returns the same as `lines.ddsurvcurve`.

`lines.ddsurvcurve`

Either returns the objects used in the call to `segments` for discrete time hazard models, or the time points and survival function used to draw the survival curve.

See Also

`ddhazard`, and `predict.ddhazard`.

Examples

```R
###
# example with continuous time model
# setup data set. See `vignette("timedep", "survival")`
library(dynamichazard)
temp <- subset(pbc, id <= 312, select=c(id,sex, stage))
pbc2 <- tmerge(temp, temp, id=id, death = event(time, status))
pbc2 <- tmerge(pbc2, pbcseq, id = id, bili = tdc(day, bili))

# fit model
f1 <- ddhazard(Surv(tstart, tstop, death == 2) ~ ddfixed(log(bili)), pbc2, id = pbc2$id, max_T = 3600, Q_0 = 1, Q = 1e-2, by = 100, model = "exponential", control = ddhazard_control(method = "EKF", eps = 1e-4, n_max = 1000, fixed_terms_method = "M_step"))

# predict with default which is all covariates set to zero
ddcurve <- ddsurvcurve(f1)
par(mar = c(4.5, 4, .5, .5))
plot(ddcurve, col = "DarkBlue", lwd = 2)

# compare with cox model
f2 <- coxph(Surv(tstart, tstop, death == 2) ~ log(bili), data = pbc2)
nw <- data.frame(bili = 1, tstart = 0, tstop = 3000)
lines(survfit(f2, newdata = nw))
```
# same as above but with bil = 3
nw <- data.frame(bil = 3)
lines(ddsurvcurve(f1, new_data = mw), col = "DarkBlue")
lines(survfit(f2, newdata = mw))

# change to time-varying slope
f3 <- ddhazard(
  Surv(tstart, tstop, death == 2) ~ log(bili), pbc2, id = pbc$bid,
  max_T = 3600, Q.0 = diag(1, 2), Q = diag(1e-2, 2), by = 100, model = "exponential",
  control = ddhazard_control(method = "EKF", eps = 1e-4, n_max = 1000))

# example with time-varying coefficient
nw <- data.frame(
  bil = c(2, 1, 3, 3, 3, 8, 3, 6, 4, 4, 9, 4, 2, 5, 1, 0, 2),
  tstart = c(0L, 225L, 407L, 750L, 1122L, 1479L, 1849L, 2193L, 2564L, 2913L, 3284L),
ddcurve <- ddsurvcurve(f3, new_data = mw, tstart = "tstart", tstop = "tstop")
lines(ddcurve, "darkorange", lwd = 2)

# can condition on survival up to some time
ddcurve <- ddsurvcurve(f3, new_data = mw[-c(1:5), ], tstart = "tstart",
  tstop = "tstop")
lines(ddcurve, lty = 2, lwd = 2)

###
# example with discrete time model
is_censored <- c(6, 27, 34, 36, 42, 46, 48:51, 51 + c(15, 30:28, 33, 35:37, 39, 40, 42:45))
head_neck_cancer <- data.frame(
  id = 1:96,
  stop = c(1, 2, 2, rep(3, 6), 4, 4, rep(5, 8),
   rep(6, 7), 7, 8, 8, 8, 9, 9, 10, 10, 11, 14, 14, 14, 15, 18, 18, 20,
   20, 37, 37, 38, 41, 45, 47, 47,
   2, 2, 3, rep(4, 4), rep(5, 5), rep(6, 5),
   7, 7, 7, 9, 10, 11, 12, 15, 16, 18, 18, 18, 21,
   21, 24, 25, 27, 36, 41, 44, 44, 52, 54, 59, 59, 63, 63, 67, 71, 76),
  event = !(1:96 * !is_censored),
  group = factor(c(rep(1, 45 + 6), rep(2, 45))))

# fit model
h1 <- ddhazard(
  Surv(stop, event) ~ group, head_neck_cancer, by = 1, max_T = 45,
  Q.0 = diag(2^2, 2), Q = diag(.01^2, 2), control = ddhazard_control( method = "GMA", eps = 1e-4, n_max = 200))

# plot predicted survival curve. Notice the steps since the model is discrete
nw <- data.frame(group = factor(1, levels = 1:2), tstart = 0, tstop = 30)
ddcurve <- ddsurvcurve(h1, new_data = mw, tstart = "tstart",
  tstop = "tstop")
plot(ddcurve, col = "Darkblue")

nw$group <- factor(2, levels = 1:2)
ddcurve <- ddsurvcurve(h1, new_data = nw, tstart = "tstart",
                       tstop = "tstop")
lines(ddcurve, col = "DarkOrange")

# compare with KM
lines(survfit(Surv(stop, event) ~ 1, head_neck_cancer, subset = group == 1),
      col = "DarkBlue")
lines(survfit(Surv(stop, event) ~ 1, head_neck_cancer, subset = group == 2),
      col = "DarkOrange")

### S3 method for class 'PF_clouds'
plot(x, y, type = c("smoothed_clouds",
                    "forward_clouds", "backward_clouds"), ylim, add = FALSE,
     qlvls = c(0.05, 0.5, 0.95), pch = 4, lty = 1, col, ..., cov_index,
     qtype = c("points", "lines"))

**Arguments**

- **x**: an object of class PF_clouds.
- **y**: unused.
- **type**: parameter to specify which cloud to plot.
- **ylim**: ylim passed to matplot.
- **add**: TRUE if a new plot should not be made.
- **qlvls**: vector of quantile levels to be plotted.
- **pch**: pch argument for the quantile points.
- **lty**: lty argument for the mean curves.
- **col**: col argument to matplot and matpoints or matlines.
- **...**: unused.
- **cov_index**: indices of the state vector to plot. All are plotted if this argument is omitted.
- **qtype**: character specifying how to show quantiles. Either "points" for crosses or "lines" for dashed lines.
plot.PF_EM

Value

List with quantile levels and mean curve.

plot.PF_EM

Plot for a PF_EM Object

Description

Short hand to call plot.PF_clouds.

Usage

## S3 method for class 'PF_EM'
plot(x, y, ...)

Arguments

x an object of class PF_EM.
y unused.
... arguments to plot.PF_clouds.

Value

See plot.PF_clouds

predict.ddhazard

Predict Method for ddhazard Object

Description

Predict method for ddhazard.

Usage

## S3 method for class 'ddhazard'
predict(object, new_data, type = c("response", "term"), tstart = "start", tstop = "stop", use_parallel, sds = F, max_threads, ...)


Arguments

- **object**: result of a `ddhazard` call.
- **new_data**: new data to base predictions on.
- **type**: either "response" for predicted probability of an event or "term" for predicted terms in the linear predictor.
- **tstart**: name of the start time column in `new_data`. It must be on the same time scale as the `tstart` used in the `Surv(tstart, tstop, event)` in the formula passed to `ddhazard`.
- **tstop**: same as `tstart` for the stop argument.
- **use_parallel**: not longer supported.
- **sds**: TRUE if point wise standard deviation should be computed. Convenient if you use functions like `ns` and you only want one term per term in the right hand side of the formula used in `ddhazard`.
- **max_threads**: not longer supported.
- **...**: not used.

Details

The function checks if there are columns in `new_data` which names match `tstart` and `tstop`. If matched, then the bins are found which the start time to the stop time are in. If `tstart` and `tstop` are not matched then all the bins used in the estimation method will be used.

Term

The result with **type** = "term" is a lists of list each having length equal to `nrow(new_data)`. The lists are

- **terms**: It's elements are matrices where the first dimension is time and the second dimension is the terms.
- **sds**: similar to terms for the point-wise confidence intervals using the smoothed co-variance matrices. Only added if `sds` = TRUE.
- **fixed_terms**: contains the fixed (non-time-varying) effect.
- **varcov**: similar to sds but differs by containing the whole covariance matrix for the terms. It is a 3D array where the third dimension is time. Only added if `sds` = TRUE.
- **start**: numeric vector with start time for each time-varying term.
- **tstop**: numeric vector with stop time for each time-varying term.

Response

The result with **type** = "response" is a list with the elements below. If `tstart` and `tstop` are matched in columns in `new_data`, then the probability will be for having an event between `tstart` and `tstop` conditional on no events before `tstart`.

- **fits**: fitted probability of an event.
- **istart**: numeric vector with start time for each element in fits.
- **istop**: numeric vector with stop time for each element in fits.
Example

```r
fit <- ddhazard(
  Surv(time, status == 2) ~ log(bili), pbc, id = pbc$id, max_T = 3600,
  Q_0 = diag(1, 2), Q = diag(1e-4, 2), by = 50,
  control = ddhazard_control(method = "GMA"))
predict(fit, type = "response", new_data =
  data.frame(time = 0, status = 2, bili = 3))
predict(fit, type = "term", new_data =
  data.frame(time = 0, status = 2, bili = 3))

# probability of an event between time 0 and 2000 with bili = 3
predict(fit, type = "response", new_data =
  data.frame(time = 0, status = 2, bili = 3, tstart = 0, tstop = 2000),
  tstart = "tstart", tstop = "tstop")
```

---

### `print.ddhazard_boot`

**Summary Statistics for a ddhazard_boot Object**

**Description**

Arguments have the same effects as for an object from a `boot` call. See `print`.

**Usage**

```r
## S3 method for class 'ddhazard_boot'
print(x, digits = getOption("digits"),
  index = 1L:ncol(boot.out$t), ...)```

**Arguments**

- `x` returned object from a `ddhazard_boot` call.
- `digits` the number of digits to be printed in the summary statistics.
- `index` indices indicating for which elements of the bootstrap output summary statistics are required.
- `...` not used.

**See Also**

`ddhazard_boot`
### print.summary.ddhazard

*Summarizing Dynamic Hazard Models Fits*

**Description**

The sd printed for time-varying effects are point-wise standard deviations from the smoothed covariance matrices.

**Usage**

```r
## S3 method for class 'summary.ddhazard'
print(x, digits =getOption("digits"), ...)

## S3 method for class 'ddhazard'
summary(object,
    var_indices = 1:ncol(object$state_vecs), max_print = 10, ...)
```

**Arguments**

- x: object returned from `summary.ddhazard`
- digits: number of digits to print.
- ...: not used.
- object: object returned from `ddhazard`.
- var_indices: variable indices to print for time-varying effects.
- max_print: maximum number of time points to print coefficients at.

### residuals.ddhazard

*Residuals Method for ddhazard Object*

**Description**

Residuals method for the result of a `ddhazard` call.

**Usage**

```r
## S3 method for class 'ddhazard'
residuals(object, type = c("std_space_error",
    "space_error", "pearson", "raw"), data = NULL, ...)
```
Arguments

- **object**: result of ddhazard call.
- **type**: type of residuals. Four possible values: "std_space_error", "space_error", "pearson" and "raw". See the sections below for details.
- **data**: data.frame with data for the Pearson or raw residuals. This is only needed if the data set is not saved with the object. Must be the same data set used in the initial call to ddhazard.

Pearson and raw residuals

Is the result of a call with a type argument of either "pearson" or "raw" for Pearson residuals or raw residuals. Returns a list with class "ddhazard_residual" with the following elements:

- **residuals**: list of residuals for each bin. Each element of the list contains a 2D array where the rows corresponds to the passed data and columns are the residuals (residuals), estimated probability of death (p_est), outcome (Y) and row number in the initial data set (row_num). The data rows will only have a residuals in a given risk list if they are at risk in that risk set.
- **type**: the type of residual.

State space errors

Is the result of a call with a type argument of either "std_space_error" or "space_error". The former is for standardized residuals while the latter is non-standardized. Returns a list with class "ddhazard_space_errors" with the following elements:

- **residuals**: 2D array with either standardized or non-standardized state space errors. The row are bins and the columns are the parameters in the regression.
- **standardize**: TRUE if standardized state space errors.
- **Covariances**: 3D array with the smoothed co-variance matrix for each set of the state space errors.

Examples

```r
library(dynamichazard)
fit <- ddhazard(
  Surv(time, status == 2) ~ log(bili), pbc, id = pbc$id, max_T = 3600,
  Q_0 = diag(1, 2), Q = diag(1e-4, 2), by = 50,
  control = ddhazard_control(method = "GMA"))
resids <- residuals(fit, type = "pearson")$residuals
head(resids[[1]])
head(resids[[2]])
```
**static_glm**  

*Static glm Fit*

**Description**

Method to fit a static model corresponding to a ddhazard fit. The method uses weights to ease the memory requirements. See `get_survival_case_weights_and_data` for details on weights.

The parallelglm_quick and parallelglm_QR methods are similar to two methods used in bam function in the mgcv package (see the `use.chol` argument or Wood et al. 2015). parallelglm_QR is more stable but slower. See Golub (2013) section 5.3 for a comparison of the Cholesky decomposition method and the QR method.

**Usage**

```r
static_glm(formula, data, by, max_T, ..., id, family = "logit", model = F, weights, risk_obj = NULL, speedglm = F, only_coef = FALSE, mf, method_use = c("glm", "speedglm", "parallelglm_quick", "parallelglm_QR"), n_threads = getOption("ddhazard_max_threads"))
```

**Arguments**

- `formula`: coxph like formula with `Surv(tstart, tstop, event)` on the left hand site of `~`
- `data`: data.frame or environment containing the outcome and covariates.
- `by`: interval length of the bins in which parameters are fixed.
- `max_T`: end of the last interval interval.
- `...`: arguments passed to glm or speedglm. If only_coef = TRUE then the arguments are passed to glm.control if glm is used.
- `id`: vector of ids for each row of the in the design matrix.
- `family`: "logit", "cloglog", or "exponential" for a static equivalent model of ddhazard.
- `model`: TRUE if you want to save the design matrix used in glm.
- `weights`: weights to use if e.g. a skewed sample is used.
- `risk_obj`: a pre-computed result from a `get_risk_obj`. Will be used to skip some computations.
- `speedglm`: depreciated.
- `only_coef`: TRUE if only coefficients should be returned. This will only call the speedglm::speedglm.wfit or glm.fit which will be faster.
- `mf`: model matrix for regression. Needed when only_coef = TRUE
- `method_use`: method to use for estimation. glm uses glm.fit, speedglm::speedglm uses speedglm::speedglm.wfit and parallelglm_quick and parallelglm_QR uses a parallel C++ estimation method.
- `n_threads`: number of threads to use when method_use is "parallelglm".
Value

The returned list from the `glm` call or just coefficients depending on the value of `only_coef`.

References


Examples

```r
library(dynamichazard)
fit <- static_glm(
  Surv(time, status == 2) ~ log(bili), pbc, id = pbc$id, max_T = 3600,
  by = 50)
fit$coefficients
```
Index

*Topic datasets
  hds, 15
  .Random.seed, 26
boot, 7, 39
coxph, 3, 12, 20, 27, 42
ddFixed, 2, 21
ddFixed_intercept (ddFixed), 2
ddhazard, 2, 3, 7, 9, 13, 14, 16, 31–34, 37, 38, 40–42
ddhazard_app, 4, 5, 6
ddhazard_boot, 5, 6, 31, 39
ddhazard_control, 4, 7
ddsurvcurve (plot.ddsurvcurve), 33
  formula, 20, 27
gam, 13
get_cloud_means, 9
get_cloud_quantiles, 10
get_Q_0, 10
get_risk_obj, 5, 8, 11, 13, 42
get_survival_case_weights_and_data, 12, 42
  glm, 42, 43
  glm.control, 42
  glm.fit, 42
  hatvalues.ddhazard, 14
  hds, 15
lines.ddsurvcurve (plot.ddsurvcurve), 33
logLik.ddhazard, 16
logLik.PF_clouds (logLik.PF_EM), 17
logLik.PF_EM, 17
matlines, 36
matplot, 36
matpoints, 36
mclapply, 11
Module, 5
nloptr, 18
ns, 38
  PF_control, 18, 20, 27
  PF_EM, 3, 18–20, 20, 21, 26, 27, 29
  PF_forward_filter, 21, 26, 29
  PF_get_score_n_hess, 29
  plot, 5, 7
  plot.ddhazard, 31
  plot.ddhazard_space_errors, 32
  plot.ddsurvcurve, 33
  plot.default, 31, 32
  plot.PF_clouds, 36, 37
  plot.PF_EM, 37
  predict, 5
  predict.ddhazard, 34, 37
  print, 39
  print.ddhazard.boot, 39
  print.summary.ddhazard, 40
residuals, 5, 32
residuals.ddhazard, 40
segments, 34
set.seed, 20, 26
speedglm, 42
static_glm, 4, 5, 13, 20, 27, 42
summary.ddhazard
  (print.summary.ddhazard), 40
Surv, 3, 11, 12, 20, 27, 33, 35, 38, 42
terms, 5