

Package ‘VAJointSurv’

January 20, 2025

Type Package

Title Variational Approximation for Joint Survival and Marker Models

Version 0.1.1

Maintainer Benjamin Christoffersen <boennecd@gmail.com>

Description Estimates joint marker (longitudinal) and survival (time-to-event) outcomes using variational approximations. The package supports multivariate markers allowing for correlated error terms and multiple types of survival outcomes which may be left-truncated, right-censored, and recurrent. Time-varying fixed and random covariate effects are supported along with non-proportional hazards.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

URL <https://github.com/boennecd/VAJointSurv>

BugReports <https://github.com/boennecd/VAJointSurv/issues>

LinkingTo Rcpp, RcppArmadillo, RcppEigen, testthat, psqn (>= 0.3.0)

Imports Rcpp, splines, utils, stats, SimSurvNMarker, psqn, Matrix, methods, lme4

Suggests rmarkdown, knitr, testthat (>= 3.0.0), xml2, R.rsp

Depends R (>= 3.5.0), survival

VignetteBuilder R.rsp

SystemRequirements C++17

Config/testthat/edition 3

NeedsCompilation yes

Author Benjamin Christoffersen [cre, aut]
(<<https://orcid.org/0000-0002-7182-1346>>),
Mark Clements [aut],
Birzhan Akynkozhayev [aut],
Antoine Savine [cph]

Repository CRAN

Date/Publication 2025-01-15 18:10:09 UTC

Contents

bs_term	2
joint_ms_format	3
joint_ms_hess	4
joint_ms_lb	6
joint_ms_opt	8
joint_ms_profile	11
joint_ms_ptr	13
joint_ms_set_vcov	15
joint_ms_start_val	17
joint_ms_va_par	19
marker_term	20
ns_term	21
plot_marker	22
plot_surv	24
poly_term	26
stacked_term	27
surv_term	28
VAJointSurv-terms	29
weighted_term	30

Index	31
--------------	-----------

bs_term	<i>Term for a B-Spline Basis for Polynomial Splines</i>
---------	---

Description

Term for a B-Spline Basis for Polynomial Splines

Usage

```
bs_term(
  x = numeric(),
  df = NULL,
  knots = NULL,
  degree = 3,
  intercept = FALSE,
  Boundary.knots = range(if (use_log) log(x) else x),
  use_log = FALSE
)
```

Arguments

x, df, knots, degree, intercept, Boundary.knots
 same as [bs](#).

use_log TRUE if the polynomials should be in the log of the argument.

Value

A list like `bs` with an additional element called `eval` to evaluate the basis. See [VAJointSurv-terms](#).

See Also

[poly_term](#), [ns_term](#), [weighted_term](#), and [stacked_term](#).

Examples

```
vals <- c(0.41, 0.29, 0.44, 0.1, 0.18, 0.65, 0.29, 0.85, 0.36, 0.47)
spline_basis <- bs_term(vals,df = 3)
# evaluate spline basis at 0.5
spline_basis$eval(0.5)
# evaluate first derivative of spline basis at 0.5
spline_basis$eval(0.5, der = 1)
```

joint_ms_format	<i>Formats the Parameter Vector</i>
-----------------	-------------------------------------

Description

Formats a parameter vector by putting the model parameters into a list with elements for each type of parameter.

Usage

```
joint_ms_format(object, par = object$start_val)
```

Arguments

object	a joint_ms object from joint_ms_ptr .
par	parameter vector to be formatted.

Value

A list with the following elements:

markers	list with an element for each marker. The lists contains an element called <code>fixef</code> for non-time-varying fixed effects and an element called <code>fixef_vary</code> time-varying fixed effects.
survival	list with an element for each survival outcome. The lists contains an element called <code>fixef</code> for non-time-varying fixed effects, an element called <code>fixef_vary</code> time-varying fixed effects, and an element called <code>associations</code> for the association parameters.
vcov	contains three covariance matrices called <code>vcov_marker</code> , <code>vcov_vary</code> and <code>vcov_surv</code> for the covariance matrix of the markers error term, the time-varying random effects, and the frailties, respectively.

Examples

```

# load in the data
library(survival)
data(pbc, package = "survival")

# re-scale by year
pbcseq <- transform(pbcseq, day_use = day / 365.25)
pbc <- transform(pbc, time_use = time / 365.25)

# create the marker terms
m1 <- marker_term(
  log(bili) ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))
m2 <- marker_term(
  albumin ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))

# base knots on observed event times
bs_term_knots <-
  with(pbc, quantile(time_use[status == 2], probs = seq(0, 1, by = .2)))

boundary <- c(bs_term_knots[ c(1, length(bs_term_knots))])
interior <- c(bs_term_knots[-c(1, length(bs_term_knots))])

# create the survival term
s_term <- surv_term(
  Surv(time_use, status == 2) ~ 1, id = id, data = pbc,
  time_fixef = bs_term(time_use, Boundary.knots = boundary, knots = interior))

# create the C++ object to do the fitting
model_ptr <- joint_ms_ptr(
  markers = list(m1, m2), survival_terms = s_term,
  max_threads = 2L, ders = list(0L, c(0L, -1L)))

# find the starting values
start_vals <- joint_ms_start_val(model_ptr)

# format the starting values
joint_ms_format(model_ptr, start_vals)

```

joint_ms_hess

Computes the Hessian

Description

Computes the Hessian

Usage

```
joint_ms_hess(
  object,
  par,
  quad_rule = object$quad_rule,
  cache_expansions = object$cache_expansions,
  eps = 1e-04,
  scale = 2,
  tol = .Machine$double.eps^(3/5),
  order = 4L,
  gh_quad_rule = object$gh_quad_rule
)
```

Arguments

object	a <code>joint_ms</code> object from joint_ms_ptr .
par	parameter vector for where the lower bound is evaluated at.
quad_rule	list with nodes and weights for a quadrature rule for the integral from zero to one.
cache_expansions	TRUE if the expansions in the numerical integration in the survival parts of the lower bound should be cached (not recomputed). This requires more memory and may be an advantage particularly with expansions that take longer to compute (like ns_term and bs_term). The computation time may be worse particularly if you use more threads as the CPU cache is not well utilized.
eps, scale, tol, order	parameter to pass to <code>psqn</code> . See psqn_hess .
gh_quad_rule	list with two numeric vectors called <code>node</code> and <code>weight</code> with Gauss–Hermite quadrature nodes and weights to handle delayed entry. A low number of quadrature nodes and weights is used when NULL is passed. This seems to work well when delayed entry happens at time with large marginal survival probabilities. The nodes and weights can be obtained e.g. from <code>fastGHQuad::gaussHermiteData</code> .

Value

A list with the following two Hessian matrices:

hessian	Hessian matrix of the model parameters with the variational parameters profiled out.
hessian_all	Hessian matrix of the model and variational parameters.

Examples

```
# load in the data
library(survival)
data(pbc, package = "survival")

# re-scale by year
```

```

pbcseq <- transform(pbcseq, day_use = day / 365.25)
pbc <- transform(pbc, time_use = time / 365.25)

# create the marker terms
m1 <- marker_term(
  log(bili) ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))
m2 <- marker_term(
  albumin ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))

# base knots on observed event times
bs_term_knots <-
  with(pbc, quantile(time_use[status == 2], probs = seq(0, 1, by = .2)))

boundary <- c(bs_term_knots[ c(1, length(bs_term_knots))])
interior <- c(bs_term_knots[-c(1, length(bs_term_knots))])

# create the survival term
s_term <- surv_term(
  Surv(time_use, status == 2) ~ 1, id = id, data = pbc,
  time_fixef = bs_term(time_use, Boundary.knots = boundary, knots = interior))

# create the C++ object to do the fitting
model_ptr <- joint_ms_ptr(
  markers = list(m1, m2), survival_terms = s_term,
  max_threads = 2L, ders = list(0L, c(0L, -1L)))

# find the starting values
start_vals <- joint_ms_start_val(model_ptr)

# optimize lower bound
fit <- joint_ms_opt(object = model_ptr, par = start_vals, gr_tol = .01)

# compute the Hessian
hess <- joint_ms_hess(object = model_ptr, par = fit$par)

# standard errors of the parameters
library(Matrix)
sqrt(diag(solve(hess$hessian)))

```

joint_ms_lb

Evaluates the Lower Bound or the Gradient of the Lower Bound

Description

Evaluates the Lower Bound or the Gradient of the Lower Bound

Usage

```

joint_ms_lb(
  object,
  par,
  n_threads = object$max_threads,
  quad_rule = object$quad_rule,
  cache_expansions = object$cache_expansions,
  gh_quad_rule = object$gh_quad_rule
)

joint_ms_lb_gr(
  object,
  par,
  n_threads = object$max_threads,
  quad_rule = object$quad_rule,
  cache_expansions = object$cache_expansions,
  gh_quad_rule = object$gh_quad_rule
)

```

Arguments

object	a joint_ms object from joint_ms_ptr .
par	parameter vector for where the lower bound is evaluated at.
n_threads	number of threads to use. This is not supported on Windows.
quad_rule	list with nodes and weights for a quadrature rule for the integral from zero to one.
cache_expansions	TRUE if the expansions in the numerical integration in the survival parts of the lower bound should be cached (not recomputed). This requires more memory and may be an advantage particularly with expansions that take longer to compute (like ns_term and bs_term). The computation time may be worse particularly if you use more threads as the CPU cache is not well utilized.
gh_quad_rule	list with two numeric vectors called node and weight with Gauss–Hermite quadrature nodes and weights to handle delayed entry. A low number of quadrature nodes and weights is used when NULL is passed. This seems to work well when delayed entry happens at time with large marginal survival probabilities. The nodes and weights can be obtained e.g. from <code>fastGHQuad::gaussHermiteData</code> .

Value

joint_ms_lb returns a number scalar with the lower bound.
 joint_ms_lb_gr returns a numeric vector with the gradient.

Examples

```

# load in the data
library(survival)

```

```

data(pbc, package = "survival")

# re-scale by year
pbcseq <- transform(pbcseq, day_use = day / 365.25)
pbc <- transform(pbc, time_use = time / 365.25)

# create the marker terms
m1 <- marker_term(
  log(bili) ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))
m2 <- marker_term(
  albumin ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))

# base knots on observed event times
bs_term_knots <-
  with(pbc, quantile(time_use[status == 2], probs = seq(0, 1, by = .2)))

boundary <- c(bs_term_knots[ c(1, length(bs_term_knots))])
interior <- c(bs_term_knots[-c(1, length(bs_term_knots))])

# create the survival term
s_term <- surv_term(
  Surv(time_use, status == 2) ~ 1, id = id, data = pbc,
  time_fixef = bs_term(time_use, Boundary.knots = boundary, knots = interior))

# create the C++ object to do the fitting
model_ptr <- joint_ms_ptr(
  markers = list(m1, m2), survival_terms = s_term,
  max_threads = 2L, ders = list(0L, c(0L, -1L)))

# find the starting values
start_vals <- joint_ms_start_val(model_ptr)

# same lower bound
all.equal(attr(start_vals, "value"), joint_ms_lb(model_ptr, par = start_vals))

```

joint_ms_opt

Optimizes the Lower Bound

Description

Optimizes the Lower Bound

Usage

```

joint_ms_opt(
  object,

```



```

par = object$start_val,
rel_eps = 1e-08,
max_it = 1000L,
n_threads = object$max_threads,
c1 = 1e-04,
c2 = 0.9,
use_bfgs = TRUE,
trace = 0L,
cg_tol = 0.5,
strong_wolfe = TRUE,
max_cg = 0L,
pre_method = 3L,
quad_rule = object$quad_rule,
mask = integer(),
cache_expansions = object$cache_expansions,
gr_tol = -1,
gh_quad_rule = object$gh_quad_rule
)

```

Arguments

object	a joint_ms object from joint_ms_ptr .
par	starting value.
rel_eps, max_it, c1, c2, use_bfgs, trace, cg_tol, strong_wolfe, max_cg, pre_method, mask, gr_tol	arguments to pass to the C++ version of psqn .
n_threads	number of threads to use. This is not supported on Windows.
quad_rule	list with nodes and weights for a quadrature rule for the integral from zero to one.
cache_expansions	TRUE if the expansions in the numerical integration in the survival parts of the lower bound should be cached (not recomputed). This requires more memory and may be an advantage particularly with expansions that take longer to compute (like ns_term and bs_term). The computation time may be worse particularly if you use more threads as the CPU cache is not well utilized.
gh_quad_rule	list with two numeric vectors called node and weight with Gauss–Hermite quadrature nodes and weights to handle delayed entry. A low number of quadrature nodes and weights is used when NULL is passed. This seems to work well when delayed entry happens at time with large marginal survival probabilities. The nodes and weights can be obtained e.g. from <code>fastGHQuad::gaussHermiteData</code> .

Value

A list with the following elements:

par	numeric vector of estimated model parameters.
value	numeric scalar with the value of optimized lower bound.

counts integer vector with the function counts and the number of conjugate gradient iterations. See [psqn](#).

convergence logical for whether the optimization converged.

Examples

```
# load in the data
library(survival)
data(pbc, package = "survival")

# re-scale by year
pbcseq <- transform(pbcseq, day_use = day / 365.25)
pbc <- transform(pbc, time_use = time / 365.25)

# create the marker terms
m1 <- marker_term(
  log(bili) ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))
m2 <- marker_term(
  albumin ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))

# base knots on observed event times
bs_term_knots <-
  with(pbc, quantile(time_use[status == 2], probs = seq(0, 1, by = .2)))

boundary <- c(bs_term_knots[ c(1, length(bs_term_knots))])
interior <- c(bs_term_knots[-c(1, length(bs_term_knots))])

# create the survival term
s_term <- surv_term(
  Surv(time_use, status == 2) ~ 1, id = id, data = pbc,
  time_fixef = bs_term(time_use, Boundary.knots = boundary, knots = interior))

# create the C++ object to do the fitting
model_ptr <- joint_ms_ptr(
  markers = list(m1, m2), survival_terms = s_term,
  max_threads = 2L, ders = list(0L, c(0L, -1L)))

# find the starting values
start_vals <- joint_ms_start_val(model_ptr)

# optimize lower bound
fit <- joint_ms_opt(object = model_ptr, par = start_vals, gr_tol = .01)

# formatted maximum likelihood estimators
joint_ms_format(model_ptr, fit$par)
```

joint_ms_profile *Approximate Likelihood Ratio based Confidence Intervals*

Description

Approximate Likelihood Ratio based Confidence Intervals

Usage

```
joint_ms_profile(
  object,
  opt_out,
  which_prof,
  delta,
  level = 0.95,
  max_step = 15L,
  rel_eps = 1e-08,
  max_it = 1000L,
  n_threads = object$max_threads,
  c1 = 1e-04,
  c2 = 0.9,
  use_bfgs = TRUE,
  trace = 0L,
  cg_tol = 0.5,
  strong_wolfe = TRUE,
  max_cg = 0L,
  pre_method = 3L,
  quad_rule = object$quad_rule,
  verbose = TRUE,
  mask = integer(),
  cache_expansions = object$cache_expansions,
  gr_tol = -1,
  hess = NULL
)
```

Arguments

object	a joint_ms object from joint_ms_ptr .
opt_out	maximum lower bound estimator from joint_ms_opt .
which_prof	index of the parameter to profile.
delta	numeric scalar greater than zero for the initial step size. Steps are made of size $2^{(\text{iteration} - 1)} * \text{delta}$. A guess of the standard deviation is a good value.
level	confidence level.
max_step	maximum number of steps to take in each direction when constructing the approximate profile likelihood curve.

rel_eps, max_it, c1, c2, use_bfgs, trace, cg_tol, strong_wolfe, max_cg, pre_method, mask, gr_tol	arguments to pass to the C++ version of <code>psqn</code> .
n_threads	number of threads to use. This is not supported on Windows.
quad_rule	list with nodes and weights for a quadrature rule for the integral from zero to one.
verbose	logical for whether to print output during the construction of the approximate profile likelihood curve.
cache_expansions	TRUE if the expansions in the numerical integration in the survival parts of the lower bound should be cached (not recomputed). This requires more memory and may be an advantage particularly with expansions that take longer to compute (like <code>ns_term</code> and <code>bs_term</code>). The computation time may be worse particularly if you use more threads as the CPU cache is not well utilized.
hess	the Hessian from <code>joint_ms_hess</code> . It is used to get better starting values along the profile likelihood curve. Use NULL if it is not passed.

Value

A list with the following elements:

confs	profile likelihood based confidence interval.
xs	the value of the parameter at which the profile likelihood is evaluated at.
p_log_Lik	numeric scalar with the profile log-likelihood.
data	list of lists of the output of each point where the profile likelihood is evaluated with the optimal parameter values of the other parameters given the constrained value of the parameter that is being profiled and the optimal value of the lower bound.

Examples

```
# load in the data
library(survival)
data(pbc, package = "survival")

# re-scale by year
pbcseq <- transform(pbcseq, day_use = day / 365.25)
pbc <- transform(pbc, time_use = time / 365.25)

# create the marker terms
m1 <- marker_term(
  log(bili) ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))
m2 <- marker_term(
  albumin ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))
```

```

# base knots on observed event times
bs_term_knots <-
  with(pbc, quantile(time_use[status == 2], probs = seq(0, 1, by = .2)))

boundary <- c(bs_term_knots[ c(1, length(bs_term_knots))])
interior <- c(bs_term_knots[-c(1, length(bs_term_knots))])

# create the survival term
s_term <- surv_term(
  Surv(time_use, status == 2) ~ 1, id = id, data = pbc,
  time_fixef = bs_term(time_use, Boundary.knots = boundary, knots = interior))

# create the C++ object to do the fitting
model_ptr <- joint_ms_ptr(
  markers = list(m1, m2), survival_terms = s_term,
  max_threads = 2L, ders = list(0L, c(0L, -1L)))

# find the starting values
start_vals <- joint_ms_start_val(model_ptr)

# optimize lower bound
fit <- joint_ms_opt(object = model_ptr, par = start_vals, gr_tol = .01)

# compute the Hessian
hess <- joint_ms_hess(object = model_ptr, par = fit$par)

# compute the standard errors
library(Matrix)
se <- sqrt(diag(solve(hess$hessian)))

# find index for the first association parameter
which_prof <- model_ptr$indices$survival[[1]]$associations[1]

# initial step size for finding the confidence interval limits
delta <- 2*se[which_prof]

# compute profile likelihood based confidence interval
# for the first association parameter
profile_CI <- joint_ms_profile(
  object = model_ptr, opt_out = fit, which_prof = which_prof,
  delta= delta, gr_tol = .01)

# comparison of CIs
profile_CI$confs
fit$par[which_prof]+c(-1,1)*qnorm(0.975)*se[which_prof]

```

joint_ms_ptr

Creates a joint_ms Object to Estimate a Joint Survival and Marker Model

Description

Creates a `joint_ms` Object to Estimate a Joint Survival and Marker Model

Usage

```
joint_ms_ptr(
  markers = list(),
  survival_terms = list(),
  max_threads = 1L,
  quad_rule = NULL,
  cache_expansions = TRUE,
  gh_quad_rule = NULL,
  ders = NULL
)
```

Arguments

<code>markers</code>	either an object from marker_term or a list of such objects.
<code>survival_terms</code>	either an object from surv_term or a list of such objects.
<code>max_threads</code>	maximum number of threads to use.
<code>quad_rule</code>	list with nodes and weights for a quadrature rule for the integral from zero to one.
<code>cache_expansions</code>	TRUE if the expansions in the numerical integration in the survival parts of the lower bound should be cached (not recomputed). This requires more memory and may be an advantage particularly with expansions that take longer to compute (like ns_term and bs_term). The computation time may be worse particularly if you use more threads as the CPU cache is not well utilized.
<code>gh_quad_rule</code>	list with two numeric vectors called <code>node</code> and <code>weight</code> with Gauss–Hermite quadrature nodes and weights to handle delayed entry. A low number of quadrature nodes and weights is used when NULL is passed. This seems to work well when delayed entry happens at time with large marginal survival probabilities. The nodes and weights can be obtained e.g. from <code>fastGHQuad::gaussHermiteData</code> .
<code>ders</code>	a list of lists with integer vectors for how the survival outcomes are linked to the markers. 0 implies present values, -1 is integral of, and 1 is the derivative. NULL implies the present value of the random effect for all markers. Note that the number of integer vectors should be equal to the number of markers.

Value

An object of `joint_ms` class with the needed C++ and R objects to estimate the model.

See Also

[joint_ms_opt](#), [joint_ms_lb](#), [joint_ms_hess](#), and [joint_ms_start_val](#).

Examples

```

# load in the data
library(survival)
data(pbc, package = "survival")

# re-scale by year
pbcseq <- transform(pbcseq, day_use = day / 365.25)
pbc <- transform(pbc, time_use = time / 365.25)

# create the marker terms
m1 <- marker_term(
  log(bili) ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))
m2 <- marker_term(
  albumin ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))

# base knots on observed event times
bs_term_knots <-
  with(pbc, quantile(time_use[status == 2], probs = seq(0, 1, by = .2)))

boundary <- c(bs_term_knots[ c(1, length(bs_term_knots))])
interior <- c(bs_term_knots[-c(1, length(bs_term_knots))])

# create the survival term
s_term <- surv_term(
  Surv(time_use, status == 2) ~ 1, id = id, data = pbc,
  time_fixef = bs_term(time_use, Boundary.knots = boundary, knots = interior))

# create the C++ object to do the fitting
model_ptr <- joint_ms_ptr(
  markers = list(m1, m2), survival_terms = s_term,
  max_threads = 2L)

```

joint_ms_set_vcov *Sets the Covariance Parameters*

Description

Sets the covariance matrices to the passed values. The function also sets covariance matrices for the variational distributions to the same values.

Usage

```

joint_ms_set_vcov(
  object,
  vcov_vary,

```

```
vcov_surv,
par = object$start_val,
va_mean = NULL
)
```

Arguments

object	a joint_ms object from joint_ms_ptr .
vcov_vary	the covariance matrix for the time-varying effects.
vcov_surv	the covariance matrix for the frailties.
par	parameter vector to be formatted.
va_mean	a matrix with the number of rows equal to the number of random effects per observation and the number of columns is the number of observations. The order for the observations needs to be the same as the id element of object.

Value

Numeric vector with model parameters.

Examples

```
# load in the data
library(survival)
data(pbc, package = "survival")

# re-scale by year
pbcseq <- transform(pbcseq, day_use = day / 365.25)
pbc <- transform(pbc, time_use = time / 365.25)

# create the marker terms
m1 <- marker_term(
  log(bili) ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))
m2 <- marker_term(
  albumin ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))

# base knots on observed event times
bs_term_knots <-
  with(pbc, quantile(time_use[status == 2], probs = seq(0, 1, by = .2)))

boundary <- c(bs_term_knots[ c(1, length(bs_term_knots))])
interior <- c(bs_term_knots[-c(1, length(bs_term_knots))])

# create the survival term
s_term <- surv_term(
  Surv(time_use, status == 2) ~ 1, id = id, data = pbc,
  time_fixef = bs_term(time_use, Boundary.knots = boundary, knots = interior))
```



```

# create the C++ object to do the fitting
model_ptr <- joint_ms_ptr(
  markers = list(m1, m2), survival_terms = s_term,
  max_threads = 2L, ders = list(0L, c(0L, -1L)))

# compute var-covar matrices with the first set of starting values
joint_ms_format(object = model_ptr)$vcov
joint_ms_va_par(object = model_ptr)[[1]]

# altering var-covar matrices
alter_pars <- joint_ms_set_vcov(
  object = model_ptr,
  vcov_vary = diag(1:4),
  vcov_surv = matrix(0,0,0))

# altered var-covar matrices
joint_ms_format(object = model_ptr, par = alter_pars)$vcov
joint_ms_va_par(object = model_ptr, par = alter_pars)[[1]]

```

joint_ms_start_val *Quick Heuristic for the Starting Values*

Description

Quick Heuristic for the Starting Values

Usage

```

joint_ms_start_val(
  object,
  par = object$start_val,
  rel_eps = 1e-08,
  max_it = 1000L,
  n_threads = object$max_threads,
  c1 = 1e-04,
  c2 = 0.9,
  use_bfgs = TRUE,
  trace = 0,
  cg_tol = 0.5,
  strong_wolfe = TRUE,
  max_cg = 0,
  pre_method = 3L,
  quad_rule = object$quad_rule,
  mask = integer(),
  cache_expansions = object$cache_expansions,
  gr_tol = -1,
  gh_quad_rule = object$gh_quad_rule
)

```

Arguments

object	a joint_ms object from <code>joint_ms_ptr</code> .
par	starting value.
rel_eps, max_it, c1, c2, use_bfgs, trace, cg_tol, strong_wolfe, max_cg, pre_method, mask, gr_tol	arguments to pass to the C++ version of <code>psqn</code> .
n_threads	number of threads to use. This is not supported on Windows.
quad_rule	list with nodes and weights for a quadrature rule for the integral from zero to one.
cache_expansions	TRUE if the expansions in the numerical integration in the survival parts of the lower bound should be cached (not recomputed). This requires more memory and may be an advantage particularly with expansions that take longer to compute (like <code>ns_term</code> and <code>bs_term</code>). The computation time may be worse particularly if you use more threads as the CPU cache is not well utilized.
gh_quad_rule	list with two numeric vectors called <code>node</code> and <code>weight</code> with Gauss–Hermite quadrature nodes and weights to handle delayed entry. A low number of quadrature nodes and weights is used when NULL is passed. This seems to work well when delayed entry happens at time with large marginal survival probabilities. The nodes and weights can be obtained e.g. from <code>fastGHQuad::gaussHermiteData</code> .

Value

Numeric vector of starting values for the model parameters.

Examples

```
# load in the data
library(survival)
data(pbc, package = "survival")

# re-scale by year
pbcseq <- transform(pbcseq, day_use = day / 365.25)
pbc <- transform(pbc, time_use = time / 365.25)

# create the marker terms
m1 <- marker_term(
  log(bili) ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))
m2 <- marker_term(
  albumin ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))

# base knots on observed event times
bs_term_knots <-
  with(pbc, quantile(time_use[status == 2], probs = seq(0, 1, by = .2)))
```

```

boundary <- c(bs_term_knots[ c(1, length(bs_term_knots))])
interior <- c(bs_term_knots[-c(1, length(bs_term_knots))])

# create the survival term
s_term <- surv_term(
  Surv(time_use, status == 2) ~ 1, id = id, data = pbc,
  time_fixef = bs_term(time_use, Boundary.knots = boundary, knots = interior))

# create the C++ object to do the fitting
model_ptr <- joint_ms_ptr(
  markers = list(m1, m2), survival_terms = s_term,
  max_threads = 2L, ders = list(0L, c(0L, -1L)))

# find the starting values
start_vals <- joint_ms_start_val(model_ptr)

```

joint_ms_va_par

Extracts the Variational Parameters

Description

Computes the estimated variational parameters for each individual.

Usage

```
joint_ms_va_par(object, par = object$start_val)
```

Arguments

object	a joint_ms object from joint_ms_ptr .
par	parameter vector to be formatted.

Value

A list with one list for each individual with the estimated mean and covariance matrix.

Examples

```

# load in the data
library(survival)
data(pbc, package = "survival")

# re-scale by year
pbcseq <- transform(pbcseq, day_use = day / 365.25)
pbc <- transform(pbc, time_use = time / 365.25)

# create the marker terms
m1 <- marker_term(

```

```

log(bili) ~ 1, id = id, data = pbcseq,
time_fixef = bs_term(day_use, df = 5L),
time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))
m2 <- marker_term(
  albumin ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))

# base knots on observed event times
bs_term_knots <-
  with(pbc, quantile(time_use[status == 2], probs = seq(0, 1, by = .2)))

boundary <- c(bs_term_knots[ c(1, length(bs_term_knots))])
interior <- c(bs_term_knots[-c(1, length(bs_term_knots))])

# create the survival term
s_term <- surv_term(
  Surv(time_use, status == 2) ~ 1, id = id, data = pbc,
  time_fixef = bs_term(time_use, Boundary.knots = boundary, knots = interior))

# create the C++ object to do the fitting
model_ptr <- joint_ms_ptr(
  markers = list(m1, m2), survival_terms = s_term,
  max_threads = 2L, ders = list(0L, c(0L, -1L)))

# find the starting values
start_vals <- joint_ms_start_val(model_ptr)

# extract variational parameters for each individual
VA_pars <- joint_ms_va_par(object = model_ptr, par = start_vals)

# number of sets of variational parameters is equal to the number of subjects
length(VA_pars)
length(unique(pbc$id))

# mean and var-covar matrix for 1st individual
VA_pars[[1]]

```

marker_term

Creates Data for One Type of Marker

Description

Creates Data for One Type of Marker

Usage

```
marker_term(formula, id, data, time_fixef, time_rng)
```

Arguments

formula	a two-sided formula with the marker outcome on the left-hand side and fixed effect covariates on the right-hand side.
id	the variable for the id of each individual.
data	a <code>data.frame</code> or environment to look at up the variables in.
time_fixef	the time-varying fixed effects. See .e.g. poly_term .
time_rng	the time-varying random effects. See .e.g. poly_term .

Details

The `time_fixef` should likely not include an intercept as this is often included in `formula`. Use `poly_term(degree = 0, raw = TRUE, intercept = TRUE)` if you want only a random intercept.

Value

An object of class `marker_term` containing longitudinal data.

Examples

```
# load in the data
library(survival)
data(pbc, package = "survival")

# re-scale by year
pbcseq <- transform(pbcseq, day_use = day / 365.25)
pbc <- transform(pbc, time_use = time / 365.25)

# create the marker terms
m1 <- marker_term(
  log(bili) ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))
m2 <- marker_term(
  albumin ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))
```

ns_term

Term for a Basis Matrix for Natural Cubic Splines

Description

Term for a Basis Matrix for Natural Cubic Splines

Usage

```
ns_term(
  x = numeric(),
  df = NULL,
  knots = NULL,
  intercept = FALSE,
  Boundary.knots = range(if (use_log) log(x) else x),
  use_log = FALSE
)
```

Arguments

`x`, `df`, `knots`, `intercept`, `Boundary.knots`
 same as `ns`.

`use_log` TRUE if the polynomials should be in the log of the argument.

Value

A list like `ns` with an additional element called `eval` to evaluate the basis. See [VAJointSurv-terms](#).

See Also

[poly_term](#), [bs_term](#), [weighted_term](#), and [stacked_term](#).

Examples

```
vals <- c(0.41, 0.29, 0.44, 0.1, 0.18, 0.65, 0.29, 0.85, 0.36, 0.47)
spline_basis <- ns_term(vals,df = 3)
# evaluate spline basis at 0.5
spline_basis$eval(0.5)
# evaluate first derivative of spline basis at 0.5
spline_basis$eval(0.5, der = 1)
```

plot_marker

Plots a Markers Mean Curve with Pointwise Quantiles

Description

Plots a Markers Mean Curve with Pointwise Quantiles

Usage

```
plot_marker(
  time_fixef,
  time_rng,
  fixef_vary,
  x_range,
  vcov_vary,
```

```

    p = 0.95,
    xlab = "Time",
    ylab = "Marker",
    newdata = NULL,
    ...
  )

```

Arguments

time_fixef	the time-varying fixed effects. See .e.g. poly_term .
time_rng	the time-varying random effects. See .e.g. poly_term .
fixef_vary	fixed effect coefficients for time_fixef.
x_range	2D numeric vector with start and end points.
vcov_vary	the covariance matrix for time_rng.
p	coverage of the two quantiles.
xlab, ylab, ...	arguments passed to plot .
newdata	data.frame with data for the weights if any.

Value

A list containing data for plotting.

Examples

```

# load in the data
library(survival)
data(pbc, package = "survival")

# re-scale by year
pbcseq <- transform(pbcseq, day_use = day / 365.25)
pbc <- transform(pbc, time_use = time / 365.25)

# create the marker terms
m1 <- marker_term(
  log(bili) ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))

fixef_vary <- c(-0.1048, 0.2583, 1.0578, 2.4006, 2.9734)
vcov_vary <- rbind(c(0.96580, 0.09543), c(0.09543, 0.03998))

# plot marker's trajectory
plot_marker(
  time_fixef = m1$time_fixef,
  time_rng = m1$time_rng,
  fixef_vary = fixef_vary,
  vcov_vary = vcov_vary, x_range = c(0,5))

```

plot_surv

*Plots Quantiles of the Conditional Hazards***Description**

Plots Quantiles of the Conditional Hazards

Usage

```
plot_surv(
  time_fixef,
  time_rng,
  x_range,
  fixef_vary,
  vcov_vary,
  frailty_var,
  ps = c(0.025, 0.5, 0.975),
  log_hazard_shift = 0,
  associations,
  xlab = "Time",
  ylab = "Hazard",
  ders = NULL,
  newdata = NULL,
  ...
)
```

Arguments

time_fixef	the time-varying fixed effects. See .e.g. poly_term . This is for the baseline hazard. Note that many basis expansions have boundary knots. It is important that these are set to cover the full range of survival times including time zero for some expansions.
time_rng	an expansion or a list of expansions for the time-varying random effects of the markers. See marker_term .
x_range	two dimensional numerical vector with the range the hazard should be plotted in.
fixef_vary	fixed effect coefficients for time_fixef.
vcov_vary	covariance matrix for the expansion or expansions in time_rng.
frailty_var	variance of the frailty.
ps	quantiles to plot.
log_hazard_shift	possible shift on the log hazard.
associations	association parameter for each time_rng or possible multiple parameters for each time_rng if ders is supplied.

xlab, ylab, ... arguments passed to `matplot`.

ders a `list` with `integer` vectors for how the survival outcome is linked to the markers. 0 implies present values, -1 is integral of, and 1 is the derivative. NULL implies the present value of the random effect for all markers.

newdata `data.frame` with data for the weights if any.

Value

A list containing data for plotting.

Examples

```
# load in the data
library(survival)
data(pbc, package = "survival")

# re-scale by year
pbcseq <- transform(pbcseq, day_use = day / 365.25)
pbc <- transform(pbc, time_use = time / 365.25)

# create the marker terms
m1 <- marker_term(
  log(bili) ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))
m2 <- marker_term(
  albumin ~ 1, id = id, data = pbcseq,
  time_fixef = bs_term(day_use, df = 5L),
  time_rng = poly_term(day_use, degree = 1L, raw = TRUE, intercept = TRUE))

# base knots on observed event times
bs_term_knots <-
  with(pbc, quantile(time_use[status == 2], probs = seq(0, 1, by = .2)))

boundary <- c(bs_term_knots[ c(1, length(bs_term_knots))])
interior <- c(bs_term_knots[-c(1, length(bs_term_knots))])

# create the survival term
s_term <- surv_term(
  Surv(time_use, status == 2) ~ 1, id = id, data = pbc,
  time_fixef = bs_term(time_use, Boundary.knots = boundary, knots = interior))

# expansion of time for the fixed effects in the survival term
time_fixef <- s_term$time_fixef
# expansion of time for the random effects in the marker terms
time_rng <- list(m1$time_rng, m2$time_rng)
# no frailty
frailty_var <- matrix(0L,1)
# var-covar matrix for time-varying random effects
vcov_vary <- c(0.9658, 0.0954, -0.1756, -0.0418, 0.0954, 0.04, -0.0276,
              -0.0128, -0.1756, -0.0276, 0.1189, 0.0077, -0.0418, -0.0128,
              0.0077, 0.0057) |> matrix(4L)
```

```

# coefficients for time-varying fixed effects
fixef_vary <- c(1.0495, -0.2004, 1.4167, 1.255, 2.5007, 4.8545, 4.7889)
# association parameters
associations <- c(0.8627, -3.2358, 0.1842)
# constant shift on the log-hazard scale
log_hazard_shift <- -4.498513
# specify how the survival outcome is linked with markers
ders = list(0L, c(0L, -1L))

# plot the hazard with pointwise quantiles
plot_surv(
  time_fixef = time_fixef,
  time_rng = time_rng,
  x_range = c(0, 5), vcov_vary = vcov_vary, frailty_var = frailty_var,
  ps = c(.25, .5, .75), log_hazard_shift = log_hazard_shift,
  fixef_vary = fixef_vary, associations = associations, ders = ders)

```

poly_term

Term for Orthogonal Polynomials

Description

Term for Orthogonal Polynomials

Usage

```

poly_term(
  x = numeric(),
  degree = 1,
  coefs = NULL,
  raw = FALSE,
  intercept = FALSE,
  use_log = FALSE
)

```

Arguments

x, degree, coefs, raw
 same as [poly](#).

intercept TRUE if there should be an intercept.

use_log TRUE if the polynomials should be in the log of the argument.

Value

A list like [poly](#) with an additional element called `eval` to evaluate the basis. See [VAJointSurv-terms](#).

See Also

[bs_term](#), [ns_term](#), [weighted_term](#), and [stacked_term](#).

Examples

```
vals <- c(0.41, 0.29, 0.44, 0.1, 0.18, 0.65, 0.29, 0.85, 0.36, 0.47)
spline_basis <- poly_term(vals, degree = 3, raw = TRUE)
# evaluate spline basis at 0.5
spline_basis$eval(0.5)
# evaluate first derivative of spline basis at 0.5
spline_basis$eval(0.5, der = 1)
```

stacked_term	<i>Term for a Basis Matrix for of Different Types of Terms</i>
--------------	--

Description

Creates a basis matrix consisting of different types of terms. E.g. to create a varying-coefficient.

Usage

```
stacked_term(...)
```

Arguments

... term objects from the package.

Value

A list with an element called eval to evaluate the basis. See [VAJointSurv-terms](#).

See Also

[poly_term](#), [bs_term](#), [ns_term](#), and [weighted_term](#).

Examples

```
vals <- c(0.41, 0.29, 0.44, 0.1, 0.18, 0.65, 0.29, 0.85, 0.36, 0.47)

spline_basis1 <- ns_term(vals, df = 3)
spline_basis2 <- bs_term(vals, df = 3)

# create stacked term from two spline bases
stacked_basis <- stacked_term(spline_basis1, spline_basis2)

# evaluate stacked basis at 0.5
stacked_basis$eval(0.5)
# evaluate first derivative of stacked basis at 0.5
stacked_basis$eval(0.5, der = 1)
```

surv_term	<i>Creates Data for One Type of Survival Outcome</i>
-----------	--

Description

Creates Data for One Type of Survival Outcome

Usage

```
surv_term(formula, id, data, time_fixef, with_frailty = FALSE, delayed = NULL)
```

Arguments

formula	a two-sided formula with the survival outcome on the left-hand side and fixed effect covariates on the right-hand side. The left-hand side needs to be a Surv object and can be either right-censored and left-truncated.
id	the variable for the id of each individual.
data	data.frame with at least the time variable.
time_fixef	the time-varying fixed effects. See .e.g. poly_term . This is for the baseline hazard. Note that many basis expansions have boundary knots. It is important that these are set to cover the full range of survival times including time zero for some expansions.
with_frailty	TRUE if there should be a frailty term.
delayed	a vector with an entry which is TRUE if the left-truncation time from the survival outcome is from a delayed entry.

Details

The `time_fixef` should likely not include an intercept as this is often included in `formula`.

The `delayed` argument is to account for delayed entry with terminal events when observations are sampled in a way such that they must not have had the event prior to their left-truncation time. In this case, the proper complete data likelihood is

$$\frac{a(u)h(t_{ij} | u)^{d_{ij}} S(t_{ij} | u)g(u)}{\int a(u)S(v_{ij} | u)du}$$

and not

$$a(u)h(t_{ij} | u)^{d_{ij}} \frac{S(t_{ij} | u)}{S(v_{ij} | u)} g(u)$$

where h is conditional hazard, S is the conditional survival function, g is additional conditional likelihood factors from other outcomes, a is the random effect distribution, t_{ij} is the observed time, d_{ij} is an event indicator, and v_{ij} is the left truncation time.

The denominator in the proper complete likelihood becomes the expectation over all delayed entries when a cluster has more than one delayed entry. See van den Berg and Drepper (2016) and Crowther et al. (2016) for further details.

Value

An object of class `surv_term` with data required for survival outcome.

References

Crowther MJ, Andersson TM, Lambert PC, Abrams KR & Humphreys K (2016). *Joint modelling of longitudinal and survival data: incorporating delayed entry and an assessment of model misspecification*. *Stat Med*, 35(7):1193-1209. doi:10.1002/sim.6779

van den Berg GJ & Drepper B (2016). *Inference for Shared-Frailty Survival Models with Left-Truncated Data*. *Econometric Reviews*, 35:6, 1075-1098, doi: 10.1080/07474938.2014.975640

Examples

```
# load in the data
library(survival)
data(pbc, package = "survival")

# re-scale by year
pbcseq <- transform(pbcseq, day_use = day / 365.25)
pbc <- transform(pbc, time_use = time / 365.25)

# base knots on observed event times
bs_term_knots <-
  with(pbc, quantile(time_use[status == 2], probs = seq(0, 1, by = .2)))

boundary <- c(bs_term_knots[ c(1, length(bs_term_knots))])
interior <- c(bs_term_knots[-c(1, length(bs_term_knots))])

# create the survival term
s_term <- surv_term(
  Surv(time_use, status == 2) ~ 1, id = id, data = pbc,
  time_fixef = bs_term(time_use, Boundary.knots = boundary, knots = interior))
```

VAJointSurv-terms

Expansions in the VAJointSurv package

Description

The VAJointSurv package uses different functions to allow for expansions in time possibly with covariate interactions. The main usage of the functions is internally but they do provide an element called ‘eval()’ which is a function to evaluate the expansion. These functions take the following arguments:

- `x` numeric vector with points at which to evaluate the expansion.
- `der` integer indicating whether to evaluate the expansion, its integral, or the derivative.
- `lower_limit` possible lower limit if integration is performed.
- `newdata` a [data.frame](#) with new data if this is required. E.g. for [weighted_term](#).

The supported terms are [ns_term](#), [bs_term](#), [poly_term](#), [weighted_term](#), and a [stacked_term](#).

weighted_term	<i>Term for a Basis Matrix for Weighted Term</i>
---------------	--

Description

Creates a weighted basis matrix where the entries are weighted with a numeric vector to e.g. create a varying-coefficient.

Usage

```
weighted_term(x, weight)
```

Arguments

x	a term type from the package.
weight	a symbol for the weight. Notice that the symbol is first first used when the eval function on the returned object is called.

Value

A list with an element called eval to evaluate the basis. See [VAJointSurv-terms](#).

See Also

[poly_term](#), [bs_term](#), [ns_term](#), and [stacked_term](#).

Examples

```
vals <- c(0.41, 0.29, 0.44, 0.1, 0.18, 0.65, 0.29, 0.85, 0.36, 0.47)

spline_basis <- ns_term(vals, df = 3)
ws <- c(4,5)
# create a weighted term
w_term <- weighted_term(spline_basis, weights)

# evaluate weighted basis at 0.5 and 0.7 with weights 4 and 5
w_term$eval(c(0.5,0.7), newdata = data.frame(weights = ws))
# evaluate the first derivative of weighted basis at 0.5 and 0.7
# with weights 4 and 5
w_term$eval(c(0.5,0.7), newdata = data.frame(weights = ws), der = 1)
```

Index

bs, [2](#), [3](#)
bs_term, [2](#), [5](#), [7](#), [9](#), [12](#), [14](#), [18](#), [22](#), [26](#), [27](#), [29](#),
[30](#)

data.frame, [28](#), [29](#)

formula, [21](#), [28](#)

integer, [14](#), [25](#)

joint_ms_format, [3](#)
joint_ms_hess, [4](#), [12](#), [14](#)
joint_ms_lb, [6](#), [14](#)
joint_ms_lb_gr (joint_ms_lb), [6](#)
joint_ms_opt, [8](#), [11](#), [14](#)
joint_ms_profile, [11](#)
joint_ms_ptr, [3](#), [5](#), [7](#), [9](#), [11](#), [13](#), [16](#), [18](#), [19](#)
joint_ms_set_vcov, [15](#)
joint_ms_start_val, [14](#), [17](#)
joint_ms_va_par, [19](#)

list, [14](#), [25](#)

marker_term, [14](#), [20](#), [24](#)
matplot, [25](#)

ns, [22](#)
ns_term, [3](#), [5](#), [7](#), [9](#), [12](#), [14](#), [18](#), [21](#), [26](#), [27](#), [29](#),
[30](#)

plot, [23](#)
plot_marker, [22](#)
plot_surv, [24](#)
poly, [26](#)
poly_term, [3](#), [21–24](#), [26](#), [27–30](#)
psqn, [9](#), [10](#), [12](#), [18](#)
psqn_hess, [5](#)

stacked_term, [3](#), [22](#), [26](#), [27](#), [29](#), [30](#)
Surv, [28](#)
surv_term, [14](#), [28](#)

VAJointSurv-terms, [29](#)
weighted_term, [3](#), [22](#), [26](#), [27](#), [29](#), [30](#)