Package ‘SimSurvNMarker’

January 20, 2025

Type Package

Title Simulate Survival Time and Markers

Version 0.1.3

Maintainer Benjamin Christoffersen <boennecd@gmail.com>

Description Provides functions to simulate from joint survival and marker models. The user can specific all basis functions of time, random or deterministic covariates, random or deterministic left-truncation and right-censoring times, and model parameters.

License GPL-2

Encoding UTF-8

Suggests testthat (>= 2.1.0), splines, R.rsp, Matrix

RoxygenNote 7.1.1

LinkingTo Rcpp, RcppArmadillo

Imports Rcpp

SystemRequirements C++14

VignetteBuilder R.rsp

URL https://github.com/boennecd/SimSurvNMarker

BugReports https://github.com/boennecd/SimSurvNMarker/issues

NeedsCompilation yes

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

Repository CRAN

Date/Publication 2022-11-07 08:50:16 UTC

Contents

`draw_U` *Samples from a Multivariate Normal Distribution*

Description

Simulates from a multivariate normal distribution and returns a matrix with appropriate dimensions.

Usage

```
draw_U(Psi_chol, n_y)
```

Arguments

`Psi_chol`	Cholesky decomposition of the covariance matrix.
`n_y`	number of markers.

Examples

```
library(SimSurvNMarker)
set.seed(1)
n_y <- 2L
K <- 3L * n_y
Psi <- drop(rWishart(1, K, diag(K)))
Psi_chol <- chol(Psi)

# example
dim(draw_U(Psi_chol, n_y))
samples <- replicate(100, draw_U(Psi_chol, n_y))
samples <- t(apply(samples, 3, c))

colMeans(samples) # ~ zeroes
cov(samples) # ~ Psi
```

`eval_marker` *Fast Evaluation of Time-varying Marker Mean Term*

Description

Evaluates the marker mean given by

$$\vec\mu(s, \vec u) = \vec o + B^\top \vec g(s) + U^\top \vec m(s).$$

Usage

```
eval_marker(ti, B, g_func, U, m_func, offset)
```

Arguments

`ti`	numeric vector with time points.
`B`	coefficient matrix for time-varying fixed effects. Use `NULL` if there is no effect.
`g_func`	basis function for `B` like `poly`.
`U`	random effects matrix for time-varying random effects. Use `NULL` if there is no effects.
`m_func`	basis function for `U` like `poly`.
`offset`	numeric vector with non-time-varying fixed effects.

Examples

```
# compare R version with this function
library(SimSurvNMarker)
set.seed(1)
n <- 100L
n_y <- 3L

ti <- seq(0, 1, length.out = n)
offset <- runif(n_y)
B <- matrix(runif(5L * n_y), nr = 5L)
g_func <- function(x)
  cbind(1, x, x^2, x^3, x^4)
U <- matrix(runif(3L * n_y), nr = 3L)
m_func <- function(x)
  cbind(1, x, x^2)

r_version <- function(ti, B, g_func, U, m_func, offset){
  func <- function(ti)
    drop(crossprod(B, drop(g_func(ti))) + crossprod(U, drop(m_func(ti))))

  vapply(ti, func, numeric(n_y)) + offset
}
```

```
# check that we get the same
stopifnot(isTRUE(all.equal(
  c(r_version  (ti[1], B, g_func, U, m_func, offset)),
    eval_marker(ti[1], B, g_func, U, m_func, offset))))
stopifnot(isTRUE(all.equal(
  r_version  (ti, B, g_func, U, m_func, offset),
  eval_marker(ti, B, g_func, U, m_func, offset))))

# check the computation time
system.time(replicate(100, r_version  (ti, B, g_func, U, m_func, offset)))
system.time(replicate(100, eval_marker(ti, B, g_func, U, m_func, offset)))
```

eval_surv_base_fun *Evaluates the Survival Function without a Marker*

Description

Evaluates the survival function at given points where the hazard is given by

$$h(t) = \exp(\vec\omega^\top\vec b(t) + \delta).$$

Usage

```
eval_surv_base_fun(ti, omega, b_func, gl_dat = get_gl_rule(30L), delta = NULL)
```

Arguments

ti	numeric vector with time points.
omega	numeric vector with coefficients for the baseline hazard.
b_func	basis function for the baseline hazard like poly.
gl_dat	Gauss–Legendre quadrature data. See get_gl_rule.
delta	offset on the log hazard scale. Use NULL if there is no effect.

Examples

```
# Example of a hazard function
b_func <- function(x)
  cbind(1, sin(2 * pi * x), x)
omega <- c(-3, 3, .25)
haz_fun <- function(x)
  exp(drop(b_func(x) %*% omega))

plot(haz_fun, xlim = c(0, 10))

# plot the hazard
library(SimSurvNMarker)
gl_dat <- get_gl_rule(60L)
```

```
plot(function(x) eval_surv_base_fun(ti = x, omega = omega,
                                    b_func = b_func, gl_dat = gl_dat),
     xlim = c(1e-4, 10), ylim = c(0, 1), bty = "l", xlab = "time",
     ylab = "Survival", yaxs = "i")

# using to few nodes gives a wrong result in this case!
gl_dat <- get_gl_rule(15L)
plot(function(x) eval_surv_base_fun(ti = x, omega = omega,
                                    b_func = b_func, gl_dat = gl_dat),
     xlim = c(1e-4, 10), ylim = c(0, 1), bty = "l", xlab = "time",
     ylab = "Survival", yaxs = "i")
```

get_gl_rule *Get Gauss–Legendre Quadrature Nodes and Weights*

Description

Computes Gauss–Legendre Quadrature nodes and weights.

Usage

```
get_gl_rule(n)
```

Arguments

n	number of nodes.

Examples

```
library(SimSurvNMarker)
get_gl_rule(4)
get_gl_rule(25)

# fast
system.time(replicate(10000, get_gl_rule(10)))
system.time(replicate(10000, get_gl_rule(100)))
```

get_ns_spline *Faster Pointwise Function than ns*

Description

Creates a function which can evaluate a natural cubic spline like `ns`.

The result may differ between different BLAS and LAPACK implementations as the QR decomposition is not unique. However, the column space of the returned matrix will always be the same regardless of the BLAS and LAPACK implementation.

Usage

```
get_ns_spline(knots, intercept = TRUE, do_log = TRUE)
```

Arguments

`knots`	sorted numeric vector with boundary and interior knots.
`intercept`	logical for whether to include an intercept.
`do_log`	logical for whether to evaluate the spline at `log(x)` or `x`.

Examples

```
# compare with splines
library(splines)
library(SimSurvNMarker)
xs <- seq(1, 5, length.out = 10L)
bks <- c(1, 5)
iks <- 2:4

# we get the same
if(require(Matrix)){
  r1 <- unclass(ns(xs, knots = iks, Boundary.knots = bks, intercept = TRUE))
  r2 <- get_ns_spline(knots = sort(c(iks, bks)), intercept = TRUE,
                      do_log = FALSE)(xs)

  cat("Rank is correct:      ", rankMatrix(cbind(r1, r2)) == NCOL(r1), "\n")

  r1 <- unclass(ns(log(xs), knots = log(iks), Boundary.knots = log(bks),
                   intercept = TRUE))
  r2 <- get_ns_spline(knots = log(sort(c(iks, bks))), intercept = TRUE,
                      do_log = TRUE)(xs)
  cat("Rank is correct (log):", rankMatrix(cbind(r1, r2)) == NCOL(r1), "\n")
}

# the latter is faster
system.time(
  replicate(100,
            ns(xs, knots = iks, Boundary.knots = bks, intercept = TRUE)))
system.time(
  replicate(100,
            get_ns_spline(knots = sort(c(iks, bks)), intercept = TRUE,
                          do_log = FALSE)(xs)))
func <- get_ns_spline(knots = sort(c(iks, bks)), intercept = TRUE,
                      do_log = FALSE)
system.time(replicate(100, func(xs)))
```

`sim_joint_data_set` *Simulate Individuals from a Joint Survival and Marker Model*

Description

Simulates individuals from the following model

$$\vec U_i \sim N^{(K)}(\vec 0, \Psi)$$
$$\vec Y_{ij} \mid \vec U_i = \vec u_i \sim N^{(r)}(\vec \mu_i(s_{ij}, \vec u_i), \Sigma)$$
$$h(t \mid \vec u) = \exp(\vec\omega^\top\vec b(t) + \vec\delta^\top\vec z_i + \vec 1^\top(diag(\vec\alpha)\otimes\vec x_i^\top)vec(\Gamma) + \vec 1^\top(diag(\vec\alpha)\otimes\vec g(t)^\top)vec(B) + \vec 1^\top(diag(\vec\alpha)\otimes\vec m(t)^\top)\vec u)$$

with

$$\vec \mu_i(s, \vec u) = (I \otimes \vec x_i^\top)vec(\Gamma) + \left(I \otimes \vec g(s)^\top\right) vec(B) + \left(I \otimes \vec m(s)^\top\right)\vec u$$

where $h(t \mid \vec u)$ is the conditional hazard function.

Usage

```
sim_joint_data_set(
  n_obs,
  B,
  Psi,
  omega,
  delta,
  alpha,
  sigma,
  gamma,
  b_func,
  m_func,
  g_func,
  r_z,
  r_left_trunc,
  r_right_cens,
  r_n_marker,
  r_x,
  r_obs_time,
  y_max,
  use_fixed_latent = TRUE,
  m_func_surv = m_func,
  g_func_surv = g_func,
  gl_dat = get_gl_rule(30L),
  tol = .Machine$double.eps^(1/4)
)
```

Arguments

`n_obs`	integer with the number of individuals to draw.
`B`	coefficient matrix for time-varying fixed effects. Use `NULL` if there is no effect.
`Psi`	the random effects' covariance matrix.
`omega`	numeric vector with coefficients for the baseline hazard.
`delta`	coefficients for fixed effects in the log hazard.
`alpha`	numeric vector with association parameters.
`sigma`	the noise's covariance matrix.
`gamma`	coefficient matrix for the non-time-varying fixed effects. Use `NULL` if there is no effect.
`b_func`	basis function for the baseline hazard like `poly`.
`m_func`	basis function for `U` like `poly`.
`g_func`	basis function for `B` like `poly`.
`r_z`	generator for the covariates in the log hazard. Takes an integer for the individual's id.
`r_left_trunc`	generator for the left-truncation time. Takes an integer for the individual's id.
`r_right_cens`	generator for the right-censoring time. Takes an integer for the individual's id.
`r_n_marker`	function to generate the number of observed markers. Takes an integer for the individual's id.
`r_x`	generator for the covariates in for the markers. Takes an integer for the individual's id.
`r_obs_time`	function to generate the observations times given the number of observed markers. Takes an integer for the number of markers and an integer for the individual's id.
`y_max`	maximum survival time before administrative censoring.
`use_fixed_latent`	logical for whether to include the $\vec{1}^\top(diag(\vec{\alpha}) \otimes \vec{x}_i^\top)vec(\Gamma)$ term in the log hazard. Useful if derivatives of the latent mean should be used.
`m_func_surv`	basis function for `U` like `poly` in the log hazard. Can be different from `m_func`. Useful if derivatives of the latent mean should be used.
`g_func_surv`	basis function for `B` like `poly` in the log hazard. Can be different from `g_func`. Useful if derivatives of the latent mean should be used.
`gl_dat`	Gauss–Legendre quadrature data. See `get_gl_rule`.
`tol`	convergence tolerance passed to `uniroot`.

See Also

See the examples on Github at `https://github.com/boennecd/SimSurvNMarker/tree/master/inst/test-data` or this vignette `vignette("SimSurvNMarker", package = "SimSurvNMarker")`.

`sim_marker` and `surv_func_joint`

Examples

```
#####
# example with polynomial basis functions
b_func <- function(x){
  x <- x - 1
  cbind(x^3, x^2, x)
}
g_func <- function(x){
  x <- x - 1
  cbind(x^3, x^2, x)
}
m_func <- function(x){
  x <- x - 1
  cbind(x^2, x, 1)
}

# parameters
delta <- c(-.5, -.5, .5)
gamma <- matrix(c(.25, .5, 0, -.4, 0, .3), 3, 2)
omega <- c(1.4, -1.2, -2.1)
Psi <- structure(c(0.18, 0.05, -0.05, 0.1, -0.02, 0.06, 0.05, 0.34, -0.25,
                   -0.06, -0.03, 0.29, -0.05, -0.25, 0.24, 0.04, 0.04,
                   -0.12, 0.1, -0.06, 0.04, 0.34, 0, -0.04, -0.02, -0.03,
                   0.04, 0, 0.1, -0.08, 0.06, 0.29, -0.12, -0.04, -0.08,
                   0.51), .Dim = c(6L, 6L))
B <- structure(c(-0.57, 0.17, -0.48, 0.58, 1, 0.86), .Dim = 3:2)
sig <- diag(c(.6, .3)^2)
alpha <- c(.5, .9)

# generator functions
r_n_marker <- function(id)
  # the number of markers is Poisson distributed
  rpois(1, 10) + 1L
r_obs_time <- function(id, n_markes)
  # the observations times are uniform distributed
  sort(runif(n_markes, 0, 2))
r_z <- function(id)
  # return a design matrix for a dummy setup
  cbind(1, (id %% 3) == 1, (id %% 3) == 2)
r_x <- r_z # same covariates for the fixed effects
r_left_trunc <- function(id)
  # no left-truncation
  0
r_right_cens <- function(id)
  # right-censoring time is exponentially distributed
  rexp(1, rate = .5)

# simulate
gl_dat <- get_gl_rule(30L)
y_max <- 2
n_obs <- 100L
set.seed(1)
```

```
dat <- sim_joint_data_set(
  n_obs = n_obs, B = B, Psi = Psi, omega = omega, delta = delta,
  alpha = alpha, sigma = sig, gamma = gamma, b_func = b_func,
  m_func = m_func, g_func = g_func, r_z = r_z, r_left_trunc = r_left_trunc,
  r_right_cens = r_right_cens, r_n_marker = r_n_marker, r_x = r_x,
  r_obs_time = r_obs_time, y_max = y_max)

# checks
stopifnot(
  NROW(dat$survival_data) == n_obs,
  NROW(dat$marker_data) >= n_obs,
  all(dat$survival_data$y <= y_max))
```

`sim_marker` *Simulate a Number of Observed Marker for an Individual*

Description

Simulates from

$$\vec U_i \sim N^{(K)}(\vec 0, \Psi)$$
$$\vec Y_{ij} \mid \vec U_i = \vec u_i \sim N^{(r)}(\vec\mu(s_{ij}, \vec u_i), \Sigma)$$

with

$$\vec\mu(s, \vec u) = \vec o + \left(I \otimes \vec g(s)^\top\right) vec(B) + \left(I \otimes \vec m(s)^\top\right) \vec u.$$

The number of observations and the observations times, s_{ij}s, are determined from the passed generating functions.

Usage

```
sim_marker(
  B,
  U,
  sigma_chol,
  r_n_marker,
  r_obs_time,
  m_func,
  g_func,
  offset,
  id = 1L
)
```

Arguments

`B`	coefficient matrix for time-varying fixed effects. Use `NULL` if there is no effect.
`U`	random effects matrix for time-varying random effects. Use `NULL` if there is no effects.
`sigma_chol`	Cholesky decomposition of the noise's covariance matrix.
`r_n_marker`	function to generate the number of observed markers. Takes an integer for the individual's id.
`r_obs_time`	function to generate the observations times given the number of observed markers. Takes an integer for the number of markers and an integer for the individual's id.
`m_func`	basis function for `U` like `poly`.
`g_func`	basis function for `B` like `poly`.
`offset`	numeric vector with non-time-varying fixed effects.
`id`	integer with id passed to `r_n_marker` and `r_obs_time`.

See Also

`draw_U`, `eval_marker`

Examples

```
#####
# example with polynomial basis functions
g_func <- function(x){
  x <- x - 1
  cbind(x^3, x^2, x)
}
m_func <- function(x){
  x <- x - 1
  cbind(x^2, x, 1)
}

# parameters
gamma <- matrix(c(.25, .5, 0, -.4, 0, .3), 3, 2)
Psi <- structure(c(0.18, 0.05, -0.05, 0.1, -0.02, 0.06, 0.05, 0.34, -0.25,
                   -0.06, -0.03, 0.29, -0.05, -0.25, 0.24, 0.04, 0.04,
                   -0.12, 0.1, -0.06, 0.04, 0.34, 0, -0.04, -0.02, -0.03,
                   0.04, 0, 0.1, -0.08, 0.06, 0.29, -0.12, -0.04, -0.08,
                   0.51), .Dim = c(6L, 6L))
B <- structure(c(-0.57, 0.17, -0.48, 0.58, 1, 0.86), .Dim = 3:2)
sig <- diag(c(.6, .3)^2)

# generator functions
r_n_marker <- function(id){
  cat(sprintf("r_n_marker: passed id is %d\n", id))
  # the number of markers is Poisson distributed
  rpois(1, 10) + 1L
}
```

```
r_obs_time <- function(id, n_markes){
  cat(sprintf("r_obs_time: passed id is %d\n", id))
  # the observations times are uniform distributed
  sort(runif(n_markes, 0, 2))
}

# simulate marker
set.seed(1)
U <- draw_U(chol(Psi), NCOL(B))
sim_marker(B = B, U = U, sigma_chol = chol(sig), r_n_marker = r_n_marker,
           r_obs_time = r_obs_time, m_func = m_func, g_func = g_func,
           offset = NULL, id = 1L)
```

surv_func_joint *Evaluates the Conditional Survival Function Given the Random Effects*

Description

Evaluates the conditional survival function given the random effects, $\vec{U}$. The conditional hazard function is

$$h(t \mid \vec{u}) = \exp(\vec{\omega}^\top \vec{b}(t) + \delta + \vec{\alpha}^\top \vec{o} + \vec{1}^\top (diag(\vec{\alpha}) \otimes \vec{g}(t)^\top) vec(B) + \vec{1}^\top (diag(\vec{\alpha}) \otimes \vec{m}(t)^\top) \vec{u}).$$

Usage

```
surv_func_joint(
  ti,
  B,
  U,
  omega,
  delta,
  alpha,
  b_func,
  m_func,
  gl_dat = get_gl_rule(30L),
  g_func,
  offset
)
```

Arguments

`ti`	numeric vector with time points.
`B`	coefficient matrix for time-varying fixed effects. Use `NULL` if there is no effect.
`U`	random effects matrix for time-varying random effects. Use `NULL` if there is no effects.

`omega`	numeric vector with coefficients for the baseline hazard.
`delta`	offset on the log hazard scale. Use `NULL` if there is no effect.
`alpha`	numeric vector with association parameters.
`b_func`	basis function for the baseline hazard like `poly`.
`m_func`	basis function for `U` like `poly`.
`gl_dat`	Gauss–Legendre quadrature data. See `get_gl_rule`.
`g_func`	basis function for `B` like `poly`.
`offset`	numeric vector with non-time-varying fixed effects.

See Also

`sim_marker`, `draw_U`, `eval_surv_base_fun`

Examples

```
#####
# example with polynomial basis functions
b_func <- function(x){
  x <- x - 1
  cbind(x^3, x^2, x)
}
g_func <- function(x){
  x <- x - 1
  cbind(x^3, x^2, x)
}
m_func <- function(x){
  x <- x - 1
  cbind(x^2, x, 1)
}

# parameters
omega <- c(1.4, -1.2, -2.1)
Psi <- structure(c(0.18, 0.05, -0.05, 0.1, -0.02, 0.06, 0.05, 0.34, -0.25,
                   -0.06, -0.03, 0.29, -0.05, -0.25, 0.24, 0.04, 0.04,
                   -0.12, 0.1, -0.06, 0.04, 0.34, 0, -0.04, -0.02, -0.03,
                   0.04, 0, 0.1, -0.08, 0.06, 0.29, -0.12, -0.04, -0.08,
                   0.51), .Dim = c(6L, 6L))
B <- structure(c(-0.57, 0.17, -0.48, 0.58, 1, 0.86), .Dim = 3:2)
alpha <- c(.5, .9)

# simulate and draw survival curve
gl_dat <- get_gl_rule(30L)
set.seed(1)
U <- draw_U(chol(Psi), NCOL(B))
tis <- seq(0, 2, length.out = 100)
Survs <- surv_func_joint(ti = tis, B = B, U = U, omega = omega,
                         delta = NULL, alpha = alpha, b_func = b_func,
                         m_func = m_func, gl_dat = gl_dat, g_func = g_func,
                         offset = NULL)
par_old <- par(mar = c(5, 5, 1, 1))
```

```
plot(tis, Survs, xlab = "Time", ylab = "Survival", type = "l",
     ylim = c(0, 1), bty = "l", xaxs = "i", yaxs = "i")
par(par_old)
```

Index