Package 'BFI'

May 19, 2025

Type Package

Title Bayesian Federated Inference

Version 3.0.1

Date/Publication 2025-05-19 08:40:10 UTC

Author Hassan Pazira [aut, cre] (ORCID:
<https://orcid.org/0000-0002-4266-6877>),
Emanuele Massa [aut] (ORCID: <https://orcid.org/0000-0001-5715-2572>),
Marianne A. Jonker [aut] (ORCID:
<https://orcid.org/0000-0003-0134-8482>)

Maintainer Hassan Pazira <hassan.pazira@radboudumc.nl>

Description The Bayesian Federated Inference ('BFI') method combines inference results obtained from local data sets in the separate centers. In this version of the package, the 'BFI' methodology is programmed for linear, logistic and survival regression models. For GLMs, see Jonker, Pazira and Coolen (2024) <doi:10.1002/sim.10072>; for survival models, see Pazira, Massa, Weijers, Coolen and Jonker (2025) <doi:10.48550/arXiv.2404.17464>; and for heterogeneous populations, see Jonker, Pazira and Coolen (2025) <doi:10.1017/rsm.2025.6>.

License MIT + file LICENSE

URL https://hassanpazira.github.io/BFI/

Encoding UTF-8

Suggests knitr, rmarkdown, roxygen2, devtools, spelling, testthat (>= 3.0.0)

VignetteBuilder knitr

Depends R (>= 3.5.0)

LazyData true

Imports stats

Config/testthat/edition 3

RoxygenNote 7.3.2

Language en-US

NeedsCompilation no

Repository CRAN

Contents

`BFI-package` *Bayesian Federated Inference*

Description

The Bayesian Federated Inference method combines inference results from different (medical) centers without sharing the data. In this version of the package, the user can fit models specifying Gaussian, Binomial (Logistic) and Survival families.

Details

Package:	BFI
Type:	Package
Version:	3.0.1
Date/Publication:	2024-04-27
License:	GPL (>=2)

`MAP.estimation` and `bfi` are the main functions. All other functions are utility functions.

Some examples are provided in the vignettes accompanying this package in order to show how the package can be applied to real data. The vignettes can be found on the package website at https://hassanpazira.github.io/BFI/ or within R once the package has been installed, e.g., via `vignette("BFI", package = "BFI")`.

Author(s)

Hassan Pazira, Emanuele Massa, Marianne A. Jonker
Maintainer: Hassan Pazira <hassan.pazira@radboudumc.nl>

References

Jonker M.A., Pazira H. and Coolen A.C.C. (2024). *Bayesian federated inference for estimating statistical models based on non-shared multicenter data sets*, *Statistics in Medicine*, 43(12): 2421-2438. <https://doi.org/10.1002/sim.10072>

Pazira H., Massa E., Weijers J.A.M., Coolen A.C.C. and Jonker M.A. (2025b). *Bayesian Federated Inference for Survival Models*, *Journal of Applied Statistics (Accepted)*. <https://arxiv.org/abs/2404.17464>

Jonker M.A., Pazira H. and Coolen A.C.C. (2025a). *Bayesian Federated Inference for regression models based on non-shared medical center data*, *Research Synthesis Methods*, 1-41. <https://doi.org/10.1017/rsm.2025.6>

`b.diag` *Create a Block Diagonal Matrix*

Description

Construct a block diagonal matrix using multiple given block matrices.

Usage

```
b.diag(..., fill = 0)
```

Arguments

`...`	individual matrices or one list of matrices.
`fill`	non-block-diagonal elements. Default is 0.

Details

Avoid combining matrices and lists for the `...` argument.

`b.diag()` covers the arguments of type "character".

If a *sparse* matrix needed, run the following:

`library(Matrix); Matrix(b_diag, sparse = TRUE)`

where `b_diag` is the matrix returned by `b.diag()`.

Value

`b.diag()` returns a block diagonal matrix obtained by combining the arguments.

Author(s)

Hassan Pazira
Maintainer: Hassan Pazira <hassan.pazira@radboudumc.nl>

Examples

```
b.diag(1, matrix(1:3, 3,4), diag(3:2))

b.diag(matrix(1:6, 2), as.character(2))

lists <- list(1, 2:3, diag(4:6), 7, cbind(8,9:12), 13:15)
b.diag(lists)
identical(b.diag(lists), b.diag(lapply(lists, as.matrix)))

b.diag(replicate(3, matrix(round(rnorm(9)), 3, 3), simplify=FALSE))
```

---

bfi *Bayesian Federated Inference*

Description

bfi function can be used (on the central server) to combine inference results from separate datasets (without combining the data) to approximate what would have been inferred had the datasets been merged. This function can handle linear, logistic and survival regression models.

Usage

```
bfi(theta_hats = NULL,
    A_hats,
    Lambda,
    family = c("gaussian", "binomial", "survival"),
    basehaz = c("weibul", "exp", "gomp", "poly", "pwexp", "unspecified"),
    stratified = FALSE,
    strat_par = NULL,
    center_spec = NULL,
    theta_A_polys = NULL,
    treat_round = NULL,
    for_ATE = NULL,
    p,
    q_ls,
    center_zero_sample = FALSE,
    which_cent_zeros,
    zero_sample_covs,
    refer_cats,
    zero_cats,
    lev_no_ref_zeros)
```

Arguments

theta_hats	a list of L vectors of the maximum a posteriori (MAP) estimates of the model parameters in the L centers. These vectors must have equal dimensions. See 'Details'.

A_hats — a list of L minus curvature matrices for L centers. These matrices must have equal dimensions. See 'Details'.

Lambda — a list of $L + 1$ matrices. The k matrix is the chosen inverse variance-covariance matrix of the Gaussian distribution that is used as prior distribution in center k, where $k = 1, 2, \ldots, L$. The last matrix is the chosen variance-covariance matrix for the Gaussian prior of the (fictive) combined data set. If `stratified = FALSE`, all $L + 1$ matrices must have equal dimensions. While, if `stratified = TRUE`, the first L matrices must have equal dimensions and the last matrix should have a different (greater) dimention than the others. See 'Details'.

family — a character string representing the family name used for the local centers. Can be abbreviated.

basehaz — a character string representing one of the available baseline hazard functions; `exponential` (`"exp"`), `Weibull` (`"weibul"`, the default), `Gompertz` (`"gomp"`), `exponentiated polynomial` (`"poly"`), `piecewise exponential` (`"pwexp"`), and `unspecified baseline hazard` (`"unspecified"`). It is only used when `family = "survival"`. Can be abbreviated. If `basehaz = "unspecified"`, it means that a (semi-parametric) Cox model is considered, and the parameters (regression coefficients) are estimated using the partial log-likelihood.

stratified — logical flag for performing the stratified analysis. If `stratified = TRUE`, the parameter(s) selected in the `strat_par` argument are allowed to be different across centers (to deal with heterogeneity across centers), except when the argument `center_spec` is not `NULL`. Default is `stratified = FALSE`. See 'Details' and 'Examples'.

strat_par — an integer vector for indicating the stratification parameter(s). It can be used to deal with heterogeneity due to center-specific parameters. For the `"binomial"` and `"gaussian"` families it is a one- or two-element integer vector so that the values 1 and/or 2 are/is used to indicate that the "intercept" and/or "sigma2" are allowed to vary, respectively. For the `"binomial"` family the length of the vector should be at most one which refers to "intercept", and the value of this element should be 1 (to handel heterogeneity across outcome means). For `"gaussian"` this vector can be 1 for indicating the "intercept" only (handeling heterogeneity across outcome means), 2 for indicating the "sigma2" only (handeling heterogeneity due to nuisance parameter), and $c(1, 2)$ for both "intercept" and "sigma2". When `family = "survival"`, this vector can contain any combination of values ranging from 1 to the maximum number of parameters of the baseline hazard function, i.e., 1 for `"exp"`, 2 for `"weibul"` and `"gomp"`, `max_order + 1` for `"poly"`, and `n_intervals` for `"pwexp"`. For example, for `"weibul"`, `strat_par` could be 1, 2 or $c(1, 2)$, where 1 represents ω_1 and 2 represents ω_2. This argument is used only when `stratified = TRUE` and `center_spec = NULL`. Default is `strat_par = NULL`. See 'Details' and 'Examples'.

center_spec — a vector of L elements to account for the heterogeneity across centers due to clustering. This argument is used only when `stratified = TRUE` and `strat_par = NULL`. Each element represents a specific feature of the corresponding center. There must be only one specific value or attribute for each center. This vector could be a numeric, characteristic or factor vector. Note that, the order of the

centers in the vector center_spec must be the same as in the list of the argument theta_hats. The used data type in the argument center_spec must be categorical. Default is center_spec = NULL. See also 'Details' and 'Examples'.

theta_A_polys
: a list with L elements so that each element is the array theta_A_ploy (the output of the MAP.estimation function, MAP.estimation()$theta_A_ploy) for the corresponding center. This argument, theta_A_polys, is only used if family = "survival" and basehaz = "poly". See 'Details' and 'Examples'.

treat_round
: a character string representing the "first" and "second" rounds of estimating the treatment effect.

for_ATE
: a list of L vectors of 9 elements to calculate the average treatment effects (ATEs) only for the binomial and gaussian families. These vectors must have equal dimensions. If treat_round = "first", then for_ATE must be NULL. If treat_round = "second", then for_ATE must be a list for binomial and gaussian, while for survival, for_ATE must be NULL. It should be defined using the output of MAP.estimation()$for_ATE obtained from the first round. See 'Details' and 'Examples'.

p
: an integer representing the number of covariates/coefficients. It can be found from the output of the MAP.estimation function, MAP.estimation()$np). This argument, p, is only used if stratified = TRUE and family = "survival".

q_ls
: a vector with L elements in which each element is the order (minus 1) of the exponentiated polynomial baseline hazard function for the corresponding center, i.e., each element is the value of q_l (the output of the MAP.estimation function, MAP.estimation()$q_l). This argument, q_ls, is only used if family = "survival", family = "survival" and basehaz = "poly". It can also be a scalar which represents the maximum value of the q_l's across the centers.

center_zero_sample
: logical flag indicating whether the center has a categorical covariate with no observations/individuals in one of the categories. It is used to address heterogeneity across centers due to center-specific covariates. Default is center_zero_sample = FALSE. For more detailes see 'References'.

which_cent_zeros
: an integer vector representing the center(s) which has one categorical covariate with no individuals in one of the categories. It is used if center_zero_sample = TRUE.

zero_sample_covs
: a vector in which each element is a character string representing the categorical covariate that has no samples/observations in one of its categories for the corresponding center. Each element of the vector can be obtained from the output of the MAP.estimation function for the corresponding center, MAP.estimation()$zero_sample_cov. It is used when center_zero_sample = TRUE.

refer_cats
: a vector in which each element is a character string representing the reference category for the corresponding center. Each element of the vector can be obtained from the output of the MAP.estimation function for the corresponding center, MAP.estimation()$refer_cat. This vector is used when center_zero_sample = TRUE.

zero_cats
: a vector in which each element is a character string representing the category with no samples/observations for the corresponding center. Each element of

the vector can be obtained from the output of the MAP.estimation function for the corresponding center, i.e., MAP.estimation()$zero_cat. It is used when center_zero_sample = TRUE.

lev_no_ref_zeros

a list in which the number of elements equals the length of the which_cent_zeros argument. Each element of the list is a vector containing the names of the levels of the categorical covariate that has no samples/observations in one of its categories for the corresponding center. However, the name of the category with no samples and the name of the reference category are excluded from this vector. Each element of the list can be obtained from the output of the MAP.estimation function, i.e., MAP.estimation()$lev_no_ref_zero. This argument is used if center_zero_sample = TRUE.

Details

bfi function implements the BFI approach described in the papers Jonker et. al. (2024a), Pazira et. al. (2024) and Jonker et. al. (2024b) given in the references. The inference results gathered from different (L) centers are combined, and the BFI estimates of the model parameters and curvature matrix evaluated at that point are returned.

The inference result from each center must be obtained using the MAP.estimation function separately, and then all of these results (coming from different centers) should be compiled into a list to be used as an input of bfi(). The models in the different centers should be defined in exactly the same way; among others, exactly the same covariates should be included in the models. The parameter vectors should be defined exactly the same, so that the L vectors and matrices in the input lists theta_hat's and A_hat's are defined in the same way (e.g., the covariates need to be included in the models in the same order).

Note that the order of the elements in the lists theta_hats, A_hats and Lambda, must be the same with respect to the centers, so that in every list the element at the ℓ position is from the center ℓ. This should also be the case for the vector center_spec.

If for the locations intercept = FALSE, the stratified analysis is not possible anymore for the binomial family.

If stratified = FALSE, both strat_par and center_spec must be NULL (the defaults), while if stratified = TRUE only one of the two must be NULL.

If stratified = FALSE and all the $L + 1$ matrices in Lambda are equal, it is sufficient to give a (list of) one matrix only. In both cases of the stratified argument (TRUE or FALSE), if only the first L matrices are equal, the argument Lambda can be a list of two matrices, so that the fist matrix represents the chosen variance-covariance matrix for local centers and the second one is the chosen matrix for the combined data set. The last matrix of the list in the argument Lambda can be built by the function inv.prior.cov().

If the data type used in the argument center_spec is continuous or categorical with the number of categories equal to the number of centers, one can use stratified = TRUE and center_spec = NULL, and set strat_par not to NULL (i.e., to 1, 2 or both $(1, 2)$). Indeed, in this case, the stratification parameter(s) given in the argument strat_par are assumed to be different across the centers.

When family = 'survival' and basehaz = 'poly', the arguments theta_hats and A_hats should not be provided. Instead, the theta_A_polys and q_ls arguments should be defined using the local information, specifically MAP.estimation()$theta_A_poly and MAP.estimation()$q_l, respectively. See Example 3 in 'Examples'.

For estimating the treatment effect, in the first round (`treat_round = "first"`), the argument `for_ATE` must be `NULL` (the default) and the family must be set to `binomial` (family is handled automatically.)

Value

`bfi` returns a list containing the following components:

`theta_hat`	the vector of estimates obtained by combining the inference results from the L centers with the `'BFI'` methodology. If an intercept was fitted in every center and `stratified = FALSE`, there is only one general "intercept" in this vector, while if `stratified = TRUE` and `strat_par = 1`, there are L different intercepts in the model, for each center one. If `treatment` is not 'NULL', when `treat_round = 'first'`, `theta_hat` gives $\hat{\gamma}_{BFI}$, and when `treat_round = 'second'`, `theta_hat` is the treatment effect $\hat{\zeta}_{BFI}$;
`A_hat`	minus the curvature (or Hessian) matrix obtained by the `'BFI'` method for the combined model. If `stratified = TRUE`, the dimension of the matrix is always greater than when `stratified = FALSE`;
`sd`	the vector of (posterior) standard deviation of the estimates in `theta_hat` obtained from the matrix in `A_hat`, i.e., the vector equals `sqrt(diag(solve(A_hat)))` which equals the square root of the elements at the diagonal of the inverse of the `A_hat` matrix.
`family`	the `family` object used;
`basehaz`	the baseline hazard function used;
`stratified`	whether a stratified analysis was done or not;
`strat_par`	the stratification parameter(s) used;
`Ave_Treat`	the estimates of the average treatment effect. Two diffterent estimations (IPTW and wIPTW) if the family is `gaussian` or `binomial`, and for the `survival` family it is 'NULL'. For more detailes see 'References'.

Author(s)

Hassan Pazira and Marianne Jonker
Maintainer: Hassan Pazira <hassan.pazira@radboudumc.nl>

References

Jonker M.A., Pazira H. and Coolen A.C.C. (2024a). *Bayesian federated inference for estimating statistical models based on non-shared multicenter data sets*, *Statistics in Medicine*, 43(12): 2421-2438. <https://doi.org/10.1002/sim.10072>

Pazira H., Massa E., Weijers J.A.M., Coolen A.C.C. and Jonker M.A. (2025b). *Bayesian Federated Inference for Survival Models*, *Journal of Applied Statistics (Accepted)*. <https://arxiv.org/abs/2404.17464>

Jonker M.A., Pazira H. and Coolen A.C.C. (2025a). *Bayesian Federated Inference for regression models based on non-shared medical center data*, *Research Synthesis Methods*, 1-41. <https://doi.org/10.1017/rsm.2025.6>

See Also

`MAP.estimation` and `inv.prior.cov`

Examples

```
################################################
##  Example 1:  y ~ Binomial  (L = 2 centers)  ##
################################################

# Setting a seed for reproducibility
set.seed(112358)

#-----------------------------------#
# Data Simulation for Local Center 1 #
#-----------------------------------#
n1 <- 30                                          # sample size of center 1
X1 <- data.frame(x1=rnorm(n1),                    # continuous variable
                 x2=sample(0:2, n1, replace=TRUE)) # categorical variable
# make dummy variables
X1x2_1 <- ifelse(X1$x2 == '1', 1, 0)
X1x2_2 <- ifelse(X1$x2 == '2', 1, 0)
X1$x2  <- as.factor(X1$x2)
# regression coefficients
beta <- 1:4  # beta[1] is the intercept
# linear predictor:
eta1   <- beta[1] + X1$x1 * beta[2] + X1x2_1 * beta[3] + X1x2_2 * beta[4]
# inverse of the link function ( g^{-1}(\eta) = \mu ):
mu1    <- binomial()$linkinv(eta1)
y1     <- rbinom(n1, 1, mu1)

#-----------------------------------#
# Data Simulation for Local Center 2 #
#-----------------------------------#
n2 <- 50                                          # sample size of center 2
X2 <- data.frame(x1=rnorm(n2),                    # continuous variable
                 x2=sample(0:2, n2, replace=TRUE)) # categorical variable
# make dummy variables:
X2x2_1 <- ifelse(X2$x2 == '1', 1, 0)
X2x2_2 <- ifelse(X2$x2 == '2', 1, 0)
X2$x2  <- as.factor(X2$x2)
# linear predictor:
eta2   <- beta[1] + X2$x1 * beta[2] + X2x2_1 * beta[3] + X2x2_2 * beta[4]
# inverse of the link function:
mu2    <- binomial()$linkinv(eta2)
y2     <- rbinom(n2, 1, mu2)

#---------------------------#
# MAP Estimates at Center 1 #
#---------------------------#
# Assume the same inverse covariance matrix (Lambda) for both centers:
Lambda     <- inv.prior.cov(X1, lambda = 0.01, family = 'binomial')
fit1       <- MAP.estimation(y1, X1, family = 'binomial', Lambda)
theta_hat1 <- fit1$theta_hat # intercept and coefficient estimates
A_hat1     <- fit1$A_hat     # minus the curvature matrix

#---------------------------#
```

```
# MAP Estimates at Center 2 #
#---------------------------#
fit2       <- MAP.estimation(y2, X2, family='binomial', Lambda)
theta_hat2 <- fit2$theta_hat
A_hat2     <- fit2$A_hat

#----------------------#
# BFI at Central Server #
#----------------------#
theta_hats <- list(theta_hat1, theta_hat2)
A_hats     <- list(A_hat1, A_hat2)
bfi        <- bfi(theta_hats, A_hats, Lambda, family='binomial')
class(bfi)
summary(bfi, cur_mat=TRUE)

###----------------------###
### Stratified Analysis ###
###----------------------###

# By running the following line an error appears because
# when stratified = TRUE, both 'strat_par' and 'center_spec' can not be NULL:
Just4check1 <- try(bfi(theta_hats, A_hats, Lambda, family = 'binomial',
                   stratified = TRUE), TRUE)
class(Just4check1) # By default, both 'strat_par' and 'center_spec' are NULL!

# By running the following line an error appears because when stratified = TRUE,
# last matrix in 'Lambda' should not have the same dim. as the other local matrices:
Just4check2 <- try(bfi(theta_hats, A_hats, Lambda, stratified = TRUE,
                   strat_par = 1), TRUE)
class(Just4check2) # All matices in Lambda have the same dimension!

# Stratified analysis when 'intercept' varies across two centers:
newLam <- inv.prior.cov(X1, lambda=c(0.1, 0.3), family = 'binomial',
                        stratified = TRUE, strat_par = 1)
bfi <- bfi(theta_hats, A_hats, list(Lambda, newLam), family = 'binomial',
           stratified=TRUE, strat_par=1)
summary(bfi, cur_mat=TRUE)


###----------------------###
###  Treatment Effect   ###
###----------------------###

set.seed(112358)

#-----------------------------------#
# Data Simulation for Local Center 1 #
#-----------------------------------#
n1 <- 30                                          # sample size of center 1
X1 <- data.frame(x1=rnorm(n1),                    # continuous variable
                 treatment=sample(1:2, n1, replace=TRUE)) # categorical variable
X1$treatment  <- as.factor(X1$treatment)
```

```
# regression coefficients
beta <- 1:3  # beta[1] is the intercept
# make dummy variable
X1x2_2 <- ifelse(X1$treatment == '2', 1, 0)
# linear predictor:
eta1   <- beta[1] + X1$x1 * beta[2] + X1x2_2 * beta[3]
# inverse of the link function ( g^{-1}(\eta) = \mu ):
mu1    <- binomial()$linkinv(eta1)
y1     <- rbinom(n1, 1, mu1)

#-----------------------------------#
# Data Simulation for Local Center 2 #
#-----------------------------------#
n2 <- 50                                          # sample size of center 2
X2 <- data.frame(x1=rnorm(n2),                    # continuous variable
                 treatment=sample(1:2, n2, replace=TRUE)) # categorical variable
X2$treatment  <- as.factor(X2$treatment)
# make dummy variables:
X2x2_2 <- ifelse(X2$treatment == '2', 1, 0)
# linear predictor:
eta2   <- beta[1] + X2$x1 * beta[2] + X2x2_2 * beta[3]
# inverse of the link function:
mu2    <- binomial()$linkinv(eta2)
y2     <- rbinom(n2, 1, mu2)

# The algorithm works even if the order of the covariates are not
# the same across centers
X2 <- X2[,c("treatment","x1")]

#----------------------#
#  Observational data   #
#----------------------#

# For observational data (RWD), we need two rounds for estimating treatment effect:

#-------------#
# First Round #
#-------------#

## Center 1:
Lambda1 <- inv.prior.cov(X1, lambda = 0.01, family = 'binomial',
                         treatment = "treatment", treat_round="first")
fit1_r1 <- MAP.estimation(y1, X1, family = 'binomial', Lambda = Lambda1,
                          treatment = "treatment", treat_round = "first")
# In the first round, the output is without the treatment!
summary(fit1_r1)

## Center 2:
Lambda2 <- inv.prior.cov(X2, lambda = 0.01, family = 'binomial',
                         treatment = "treatment", treat_round="first")
fit2_r1 <- MAP.estimation(y2, X2, family = 'binomial', Lambda = Lambda2,
                          treatment = "treatment", treat_round = "first")
fit2_r1
```

```
## Centeral Server:
theta_hats_r1 <- list(fit1_r1$theta_hat, fit2_r1$theta_hat)
A_hats_r1 <- list(fit1_r1$A_hat, fit2_r1$A_hat)
fitbfi_r1 <- bfi(theta_hats_r1, A_hats_r1, Lambda1, family = 'binomial',
                 treat_round = "first")
summary(fitbfi_r1, cur_mat = TRUE)

#--------------#
# Second Round #
#--------------#

## Center 1:
Lambda11 <- inv.prior.cov(X1, lambda = 0.01, family = 'binomial',
                          treatment = "treatment", treat_round="second")
fit1_r2 <- MAP.estimation(y1, X1, family = 'binomial', Lambda = Lambda11,
                          treatment = "treatment", treat_round = "second",
                          gamma_bfi = fitbfi_r1$theta_hat)
# In the second round, the output is only with the treatment!
summary(fit1_r2)

## Center 2:
Lambda22 <- inv.prior.cov(X2, lambda = 0.01, family = 'binomial',
                         treatment = "treatment", treat_round="second")
fit2_r2 <- MAP.estimation(y2, X2, family = 'binomial', Lambda = Lambda22,
                          treatment = "treatment", treat_round = "second",
                          gamma_bfi = fitbfi_r1$theta_hat)
fit2_r2$propensity # Propensity Score
fit2_r2$for_ATE # will be used in central server
fit2_r2

## Centeral Server:
theta_hats_r2 <- list(fit1_r2$theta_hat, fit2_r2$theta_hat)
A_hats_r2 <- list(fit1_r2$A_hat, fit2_r2$A_hat)
for_ATEs <- list(fit1_r2$for_ATE, fit2_r2$for_ATE)
fitbfi_r2 <- bfi(theta_hats_r2, A_hats_r2, Lambda11, family = 'binomial',
                 treat_round = "second", for_ATE = for_ATEs)
fitbfi_r2$S_var
fitbfi_r2$Ave_Treat
summary(fitbfi_r2)

#--------------------#
#  Randomized Trial  #
#--------------------#

# For Randomized Control Trial (RCT), we need only one round (the second round) for
# estimating treatment effect. Because we do not need to estimate propensity score.
# For example, in a 1:1 randomized trial, the propensity scores are, by definition,
# equal to 0.5. Here we use 'RCT_propens', instead of 'gamma_bfi':

## Center 1:
Lambda11 <- inv.prior.cov(X1, lambda = 0.01, family = 'binomial',
                          treatment = "treatment", treat_round="second")
```

```
fit1_r2 <- MAP.estimation(y1, X1, family = 'binomial', Lambda = Lambda11,
                          treatment = "treatment", treat_round = "second",
                          RCT_propens = rep(0.5, n1)) # gamma_bfi = NULL
summary(fit1_r2)

## Center 2:
Lambda22 <- inv.prior.cov(X2, lambda = 0.01, family = 'binomial',
                          treatment = "treatment", treat_round="second")
fit2_r2 <- MAP.estimation(y2, X2, family = 'binomial', Lambda = Lambda22,
                          treatment = "treatment", treat_round = "second",
                          RCT_propens = rep(0.5, n2)) # gamma_bfi = NULL
fit2_r2$for_ATE # will be used in central server
fit2_r2

## Centeral Server:
theta_hats_r2 <- list(fit1_r2$theta_hat, fit2_r2$theta_hat)
A_hats_r2 <- list(fit1_r2$A_hat, fit2_r2$A_hat)
for_ATEs <- list(fit1_r2$for_ATE, fit2_r2$for_ATE)
fitbfi_r2 <- bfi(theta_hats_r2, A_hats_r2, Lambda11, family = 'binomial',
                 treat_round = "second", for_ATE = for_ATEs)
fitbfi_r2$S_var
fitbfi_r2$Ave_Treat
summary(fitbfi_r2)


################################################
##  Example 2:  y ~ Gaussian  (L = 3 centers)  ##
################################################

# Setting a seed for reproducibility
set.seed(112358)

p     <- 3                    # number of coefficients without 'intercept'
theta <- c(1, rep(2, p), 1.5)  # reg. coef.s ('intercept' is 1) & 'sigma2' = 1.5

#-----------------------------------#
# Data Simulation for Local Center 1 #
#-----------------------------------#
n1   <- 30                                      # sample size of center 1
X1   <- data.frame(matrix(rnorm(n1 * p), n1, p)) # continuous variables
# linear predictor:
eta1 <- theta[1] + as.matrix(X1)
# inverse of the link function ( g^{-1}(\eta) = \mu ):
mu1  <- gaussian()$linkinv(eta1)
y1   <- rnorm(n1, mu1, sd = sqrt(theta[5]))

#-----------------------------------#
# Data Simulation for Local Center 2 #
#-----------------------------------#
n2   <- 40                                      # sample size of center 2
X2   <- data.frame(matrix(rnorm(n2 * p), n2, p)) # continuous variables
# linear predictor:
eta2 <- theta[1] + as.matrix(X2)
```

```
# inverse of the link function:
mu2  <- gaussian()$linkinv(eta2)
y2   <- rnorm(n2, mu2, sd = sqrt(theta[5]))

#-----------------------------------#
# Data Simulation for Local Center 3 #
#-----------------------------------#
n3   <- 50                                        # sample size of center 3
X3   <- data.frame(matrix(rnorm(n3 * p), n3, p)) # continuous variables
# linear predictor:
eta3 <- theta[1] + as.matrix(X3)
# inverse of the link function:
mu3  <- gaussian()$linkinv(eta3)
y3   <- rnorm(n3, mu3, sd = sqrt(theta[5]))

#---------------------------#
# Inverse Covariance Matrix #
#---------------------------#
# Creating the inverse covariance matrix for the Gaussian prior distribution:
# the same for both centers
Lambda <- inv.prior.cov(X1, lambda = 0.05, family='gaussian')

#---------------------------#
# MAP Estimates at Center 1 #
#---------------------------#
fit1       <- MAP.estimation(y1, X1, family = 'gaussian', Lambda)
theta_hat1 <- fit1$theta_hat # intercept and coefficient estimates
A_hat1     <- fit1$A_hat     # minus the curvature matrix

#---------------------------#
# MAP Estimates at Center 2 #
#---------------------------#
fit2       <- MAP.estimation(y2, X2, family = 'gaussian', Lambda)
theta_hat2 <- fit2$theta_hat
A_hat2     <- fit2$A_hat

#---------------------------#
# MAP Estimates at Center 3 #
#---------------------------#
fit3       <- MAP.estimation(y3, X3, family = 'gaussian', Lambda)
theta_hat3 <- fit3$theta_hat
A_hat3     <- fit3$A_hat

#-----------------------#
# BFI at Central Server #
#-----------------------#
A_hats     <- list(A_hat1, A_hat2, A_hat3)
theta_hats <- list(theta_hat1, theta_hat2, theta_hat3)
bfi        <- bfi(theta_hats, A_hats, Lambda, family = 'gaussian')
summary(bfi, cur_mat=TRUE)

###---------------------###
### Stratified Analysis ###
```

```
###---------------------###

# Stratified analysis when 'intercept' varies across two centers:
newLam1 <- inv.prior.cov(X1, lambda = c(0.1,0.3), family = 'gaussian',
                         stratified = TRUE, strat_par = 1, L = 3)
# 'newLam1' is used as the prior for combined data and
# 'Lambda' is used as the prior for locals
list_newLam1 <- list(Lambda, newLam1)
bfi1 <- bfi(theta_hats, A_hats, list_newLam1, family = 'gaussian',
            stratified = TRUE, strat_par = 1)
summary(bfi1, cur_mat = TRUE)

# Stratified analysis when 'sigma2' varies across two centers:
newLam2 <- inv.prior.cov(X1, lambda = c(0.1,0.3), family = 'gaussian',
                         stratified = TRUE, strat_par = 2, L = 3)
# 'newLam2' is used as the prior for combined data and 'Lambda' is used as
# the prior for locals
list_newLam2 <- list(Lambda, newLam2)
bfi2 <- bfi(theta_hats, A_hats, list_newLam2, family = 'gaussian',
            stratified = TRUE, strat_par=2)
summary(bfi2, cur_mat = TRUE)

# Stratified analysis when 'intercept' and 'sigma2' vary across 2 centers:
newLam3 <- inv.prior.cov(X1, lambda = c(0.1,0.2,0.3), family = 'gaussian',
                         stratified = TRUE, strat_par = c(1, 2), L = 3)
# 'newLam3' is used as the prior for combined data and 'Lambda' is used as
# the prior for locals
list_newLam3 <- list(Lambda, newLam3)
bfi3 <- bfi(theta_hats, A_hats, list_newLam3, family = 'gaussian',
            stratified = TRUE, strat_par = 1:2)
summary(bfi3, cur_mat = TRUE)

###----------------------------###
### Center Specific Covariates ###
###----------------------------###

# Assume the first and third centers have the same center-specific covariate value
# of 'High', while this value for the second center is 'Low', i.e.,
# center_spec = c('High','Low','High')
newLam4 <- inv.prior.cov(X1, lambda=c(0.1, 0.2, 0.3), family='gaussian',
                         stratified = TRUE, center_spec = c('High','Low','High'),
                         L = 3)
# 'newLam4' is used as the prior for combined data and 'Lambda' is used as
# the prior for locals
l_newLam4 <- list(Lambda, newLam4)
bfi4 <- bfi(theta_hats, A_hats, l_newLam4, family = 'gaussian',
            stratified = TRUE, center_spec = c('High','Low','High'))
summary(bfi4, cur_mat = TRUE)


###---------------------###
###  Treatment Effect   ###
###---------------------###
```

```
set.seed(112358)

#-----------------------------#
# New Data for Local Center 1 #
#-----------------------------#
# Generating new data with 'treatment' variable
# We cansider the first variable (X1$X1) to be the treatment:
X1$X1 <- sample(0:1, n1, replace=TRUE) # categorical variable
eta1  <- theta[1] + as.matrix(X1)
mu1   <- gaussian()$linkinv(eta1)
y1    <- rnorm(n1, mu1, sd = sqrt(theta[5]))

#-----------------------------#
# New Data for Local Center 2 #
#-----------------------------#
# We cansider the first variable (X2$X1) to be the treatment:
X2$X1 <- sample(0:1, n2, replace=TRUE) # categorical variable
eta2  <- theta[1] + as.matrix(X2)
mu2   <- gaussian()$linkinv(eta2)
y2    <- rnorm(n2, mu2, sd = sqrt(theta[5]))

#-----------------------------#
# New Data for Local Center 3 #
#-----------------------------#
# We cansider the first variable (X3$X1) to be the treatment:
X3$X1 <- sample(0:1, n3, replace=TRUE) # categorical variable
# linear predictor:
eta3  <- theta[1] + as.matrix(X3)
# inverse of the link function:
mu3   <- gaussian()$linkinv(eta3)
y3    <- rnorm(n3, mu3, sd = sqrt(theta[5]))

#-----------------------#
#  Observational data   #
#-----------------------#

# For observational data (RWD), we need two rounds for estimating treatment effect:

#-------------#
# First Round #
#-------------#

## Center 1:
Lambda1 <- inv.prior.cov(X1, lambda = 0.01, family = 'binomial',
                         treatment = "X1", treat_round="first")
# When treat_round = "first", the family will automatically set to 'binomial',
# even if family = 'gaussian' or family = 'survival'.
fit1_r1 <- MAP.estimation(y1, X1, family = 'gaussian', Lambda = Lambda1,
                          treatment = "X1", treat_round = "first")
# Althghou family = 'gaussian', the output is based on 'binomial'!
# The output without the treatment (X1) in the first round!
summary(fit1_r1)
```

```
## Center 2:
Lambda2 <- inv.prior.cov(X2, lambda = 0.01, family = 'gaussian',
                         treatment = "X1", treat_round="first")
fit2_r1 <- MAP.estimation(y2, X2, family = 'gaussian', Lambda = Lambda2,
                          treatment = "X1", treat_round = "first")
fit2_r1

## Center 3:
Lambda3 <- inv.prior.cov(X3, lambda = 0.01, family = 'gaussian',
                         treatment = "X1", treat_round="first")
fit3_r1 <- MAP.estimation(y3, X3, family = 'gaussian', Lambda = Lambda3,
                          treatment = "X1", treat_round = "first")

## Centeral Server:
theta_hats_r1 <- list(fit1_r1$theta_hat, fit2_r1$theta_hat, fit3_r1$theta_hat)
A_hats_r1 <- list(fit1_r1$A_hat, fit2_r1$A_hat, fit3_r1$A_hat)
fitbfi_r1 <- bfi(theta_hats_r1, A_hats_r1, Lambda1, family = 'gaussian',
                 treat_round = "first") # same results with 'binomial'
# The output without the treatment (X1) in the first round!
summary(fitbfi_r1, cur_mat = TRUE)

#--------------#
# Second Round #
#--------------#

## Center 1:
Lambda11 <- inv.prior.cov(X1, lambda = 0.01, family = 'gaussian',
                          treatment = "X1", treat_round="second")
fit1_r2 <- MAP.estimation(y1, X1, family = 'gaussian', Lambda = Lambda11,
                          treatment = "X1", treat_round = "second",
                          gamma_bfi = fitbfi_r1$theta_hat)
# The output with only the treatment (X1) in the second round!
summary(fit1_r2)

## Center 2:
Lambda22 <- inv.prior.cov(X2, lambda = 0.01, family = 'gaussian', treatment = "X1",
                          treat_round="second")
fit2_r2 <- MAP.estimation(y2, X2, family = 'gaussian', Lambda = Lambda22,
                          treatment = "X1", treat_round = "second",
                          gamma_bfi = fitbfi_r1$theta_hat)

## Center 3:
Lambda33 <- inv.prior.cov(X3, lambda = 0.01, family = 'gaussian', treatment = "X1",
                          treat_round="second")
fit3_r2 <- MAP.estimation(y3, X3, family = 'gaussian', Lambda = Lambda33,
                          treatment = "X1", treat_round = "second",
                          gamma_bfi = fitbfi_r1$theta_hat)

## Centeral Server:
theta_hats_r2 <- list(fit1_r2$theta_hat, fit2_r2$theta_hat, fit3_r2$theta_hat)
A_hats_r2 <- list(fit1_r2$A_hat, fit2_r2$A_hat, fit3_r2$A_hat)
for_ATEs <- list(fit1_r2$for_ATE, fit2_r2$for_ATE, fit3_r2$for_ATE)
```

```
fitbfi_r2 <- bfi(theta_hats_r2, A_hats_r2, Lambda11, family = 'gaussian',
                 treat_round = "second", for_ATE = for_ATEs)
fitbfi_r2$Ave_Treat
fitbfi_r2$S_var
summary(fitbfi_r2)


#####################################################
##  Example 3:  Survival family  (L = 2 centers)  ##
#####################################################

# Setting a seed for reproducibility
set.seed(112358)

p <- 3
theta <- c(1:4, 5, 6)  # regression coefficients (1:4) & omega's (5:6)

#-------------------------------------------#
# Simulating Survival data for Local Center 1 #
#-------------------------------------------#
n1 <- 30
X1 <- data.frame(matrix(rnorm(n1 * p), n1, p)) # continuous (normal) variables
# Simulating survival data ('time' and 'status') from 'Weibull' with
# a predefined censoring rate of 0.3:
y1 <- surv.simulate(Z = list(X1), beta = theta[1:p], a = theta[5],
                    b = theta[6], u1 = 0.1, cen_rate = 0.3,
                    gen_data_from = "weibul")$D[[1]][, 1:2]

## MAP Estimates at Center 1
Lambda <- inv.prior.cov(X1, lambda = c(0.1, 1), family = "survival",
                        basehaz = "poly")
fit1 <- MAP.estimation(y1, X1, family = 'survival', Lambda = Lambda,
                       basehaz = "poly")
theta_hat1 <- fit1$theta_hat  # coefficient estimates
A_hat1     <- fit1$A_hat      # minus the curvature matrix
summary(fit1, cur_mat=TRUE)
fit1$theta_A_poly # Only when family = "survival" and basehaz ="poly"

#-------------------------------------------#
# Simulating Survival data for Local Center 2 #
#-------------------------------------------#
n2 <- 30
X2 <- data.frame(matrix(rnorm(n2 * p), n2, p)) # continuous (normal) variables
# Survival simulated data from 'Weibull' with a predefined censoring rate of 0.3:
y2 <- surv.simulate(Z = list(X2), beta = theta[1:p], a = theta[5],
                    b = theta[6],u1 = 0.1, cen_rate = 0.3,
                    gen_data_from = "weibul")$D[[1]][, 1:2]

## MAP Estimates at Center 2
fit2 <- MAP.estimation(y2, X2, family = 'survival', Lambda = Lambda,
                       basehaz = "poly")
theta_hat2 <- fit2$theta_hat
A_hat2 <- fit2$A_hat
```

```
summary(fit2, cur_mat=TRUE)

#----------------------#
# BFI at Central Server #
#----------------------#
# When family = 'survival' and basehaz = "poly", only 'theta_A_polys'
# should be defined instead of 'theta_hats' and 'A_hats':
theta_A_hats <- list(fit1$theta_A_poly, fit2$theta_A_poly)
qls <- c(fit1$q_l, fit2$q_l)
bfi <- bfi(Lambda = Lambda, family = 'survival', theta_A_polys = theta_A_hats,
           basehaz = "poly", q_ls = qls)
summary(bfi, cur_mat=TRUE)


###---------------------###
### Stratified Analysis ###
###---------------------###

# Stratified analysis when first parameter ('omega_0') varies across two centers:
(newLam0 <- inv.prior.cov(X1, lambda = c(rep(1, 3), 0.3, 0.7, rep(2,2)),
                          family = 'survival', stratified = TRUE,
                          basehaz = c("poly"), strat_par = 1, L = 2))
# 'newLam0' is used as the prior for combined data and 'Lambda' is used as for locals:
list_newLam0 <- list(Lambda, newLam0)
bfi0 <- bfi(Lambda = list_newLam0, family = 'survival', theta_A_polys = theta_A_hats,
            stratified = TRUE, basehaz = c("poly"), p = 3, q_ls = qls, strat_par = 1)
summary(bfi0, cur_mat = TRUE)


# Stratified analysis when the first and second parameters ('omega_0' and 'omega_1')
# vary across two centers:
newLam1 <- inv.prior.cov(X1, lambda = c(rep(1, 3), 0.3, 0.7, 0.5, 0.8, 2),
                         family = 'survival', stratified = TRUE, basehaz = c("poly"),
                         strat_par = c(1, 2), L = 2)
# 'newLam1' is used as the prior for combined data:
list_newLam1 <- list(Lambda, newLam1)
bfi1 <- bfi(Lambda = list_newLam1, family = 'survival', theta_A_polys = theta_A_hats,
            stratified = TRUE, basehaz = c("poly"), p = 3, q_ls = qls,
            strat_par = c(1, 2))
summary(bfi1, cur_mat = TRUE)


###---------------------###
###  Treatment Effect   ###
###---------------------###

set.seed(112358)

#-----------------------------#
# New Data for Local Center 1 #
#-----------------------------#
# Generating new data with 'treatment' variable
# We cansider the first variable (X1$X1) to be the treatment
```

```
X1$X1 <- sample(0:1, n1, replace=TRUE) # categorical variable
y1 <- surv.simulate(Z = list(X1), beta = theta[1:p], a = theta[5], b = theta[6],
                    u1 = 0.1, cen_rate = 0.3, gen_data_from = "weibul")$D[[1]][, 1:2]

#-----------------------------#
# New Data for Local Center 2 #
#-----------------------------#
# We cansider the first variable (X2$X1) to be the treatment!
X2$X1 <- sample(0:1, n2, replace=TRUE) # categorical variable
y2 <- surv.simulate(Z = list(X2), beta = theta[1:p], a = theta[5], b = theta[6],
                    u1 = 0.1, cen_rate = 0.3, gen_data_from = "weibul")$D[[1]][, 1:2]

#-------------#
# First Round #
#-------------#

## Center 1:
Lambda1 <- inv.prior.cov(X1, lambda = 0.01, family = 'survival',
                         treatment = "X1", treat_round="first")
# When treat_round = "first", the family will automatically set to 'binomial',
# even if family = 'gaussian' or family = 'survival'.
fit1_r1 <- MAP.estimation(y1, X1, family = 'survival', # 'basehaz' is not needed!
                          Lambda = Lambda1, treatment = "X1", treat_round = "first")
# While family = 'survival', the output is based on 'binomial' with no 'Intercept'!
# The output without the treatment (X1) in the first round!
summary(fit1_r1)

## Center 2:
Lambda2 <- inv.prior.cov(X2, lambda = 0.01, family = 'survival',
                         treatment = "X1", treat_round="first")
fit2_r1 <- MAP.estimation(y2, X2, family = 'survival', Lambda = Lambda2,
                          treatment = "X1", treat_round = "first")
fit2_r1

## Centeral Server:
theta_hats_r1 <- list(fit1_r1$theta_hat, fit2_r1$theta_hat)
A_hats_r1 <- list(fit1_r1$A_hat, fit2_r1$A_hat)
fitbfi_r1 <- bfi(theta_hats_r1, A_hats_r1, Lambda1, family = 'survival',
                 treat_round = "first")
# In the first round output is based on 'binomial', and without
# the intercept and treatment (X1):
summary(fitbfi_r1, cur_mat = TRUE)

#--------------#
# Second Round #
#--------------#

## Center 1:
Lambda11 <- inv.prior.cov(X1, lambda = 0.01, family = 'survival',
                          basehaz = "unspecified", treatment = "X1",
                          treat_round="second")
fit1_r2 <- MAP.estimation(y1, X1, family = 'survival', Lambda = Lambda11,
                          basehaz = "unspecified", treatment = "X1",
```

```
                          treat_round = "second", gamma_bfi = fitbfi_r1$theta_hat)
# The output with only the treatment (X1) in the second round!
summary(fit1_r2)

## Center 2:
Lambda22 <- inv.prior.cov(X2, lambda = 0.01, family = 'survival',
                          basehaz = "unspecified", treatment = "X1",
                          treat_round="second")
fit2_r2 <- MAP.estimation(y2, X2, family = 'survival', basehaz = "unspecified",
                          Lambda = Lambda22, treatment = "X1",
                          treat_round = "second", gamma_bfi = fitbfi_r1$theta_hat)
fit2_r2

## Centeral Server:
theta_hats_r2 <- list(fit1_r2$theta_hat, fit2_r2$theta_hat)
A_hats_r2 <- list(fit1_r2$A_hat, fit2_r2$A_hat)
fitbfi_r2 <- bfi(theta_hats_r2, A_hats_r2, Lambda11, family = 'survival',
                 basehaz = "unspecified", treat_round = "second")
# When family = 'survival', 'for_ATE' is not calculated.
summary(fitbfi_r2)
```

`hazards.fun` *Compute the estimated (baseline/cumulative) hazard and (baseline) survival functions*

Description

For a given vector of times, `hazards.fun` computes the estimated baseline hazard, cumulative baseline hazard, hazard, baseline survival, and survival functions. It can be used for prediction on a new sample.

Usage

```
hazards.fun(time,
            z = NULL,
            p,
            theta_hat,
            basehaz = c("weibul", "exp", "gomp", "poly", "pwexp"),
            q_max,
            timax)
```

Arguments

`time`	the vector containing the time values for which the hazard rate is computed. If the argument `z` is not `NULL`, then the length of the argument `time` should be the number of columns of `z`, which is p.

z	a new observation vector of length p. If z = NULL (the default), then the relative risk ($\boldsymbol{z}^\top \boldsymbol{\beta}$) is considered a vector of 1 with length n.
p	the number of coefficients. It is taken equal to the number of elements of the argument z, if z is not NULL.
theta_hat	a vector contains the values of the estimated parameters. The first p values represent the coefficient parameters ($\boldsymbol{\beta}$), while the remaining values pertain to the parameters of the baseline hazard function ($\boldsymbol{\omega}$).
basehaz	a character string representing one of the available baseline hazard functions; exponential ("exp"), Weibull ("weibul", the default), Gompertz ("gomp"), exponentiated polynomial ("poly"), and piecewise exponential ("pwexp"). Can be abbreviated.
q_max	a value represents the order of the exponentiated polynomial baseline hazard function. This argument should only be used when basehaz = "poly". In the case of multiple centers, the maximum value of the orders should be used. ql.LRT() can be used for obtaining of the order of each center.
timax	a value represents the minimum (or maximum) value of the maximum times observed in the different centers. This argument should only be used when basehaz = "pwexp".

Details

hazards.fun computes the estimated baseline hazard, cumulative baseline hazard, hazard, baseline survival, and survival functions at different time points specified in the argument time.

The function hazards.fun() can be used for prediction purposes with new sample. The arguments time and z should be provided for the new data.

Value

hazards.fun returns a list containing the following components:

bhazard	the vector of estimates of the baseline hazard function at the time points given by the argument time;
cbhazard	the vector of estimates of the cumulative baseline hazard function at the time points specified in the argument time;
bsurvival	the vector of estimates of the baseline survival function at the time points given by the argument time;
hazard	the vector of estimates of the hazard function at the time points given by the argument time;
chazard	the vector of estimates of the cumulative hazard function at the time points specified in the argument time;
survival	the vector of estimates of the survival function at the time points given by the argument time.

Author(s)

Hassan Pazira
Maintainer: Hassan Pazira <hassan.pazira@radboudumc.nl>

References

Pazira H., Massa E., Weijers J.A.M., Coolen A.C.C. and Jonker M.A. (2025b). *Bayesian Federated Inference for Survival Models*, *Journal of Applied Statistics (Accepted)*. <https://arxiv.org/abs/2404.17464>

See Also

MAP.estimation

Examples

```
# Setting a seed for reproducibility
set.seed(1123)

##-------------------------
## Simulating Survival data
##-------------------------

n <- 40
p <- 7
Original_data <- data.frame(matrix(rnorm((n+1) * p), (n+1), p))
X <- Original_data[1:n,]
X_new <- Original_data[(n+1),]
# Simulating survival data from Exponential distribution
# with a predefined censoring rate of 0.2:
Orig_y <- surv.simulate(Z = Original_data, beta = rep(1,p), a = exp(1),
                        cen_rate = 0.2, gen_data_from = "exp")$D[[1]][,1:2]
y <- Orig_y[1:n,]
y_new <- Orig_y[(n+1),]
time_points <- seq(0, max(y$time), length.out=20)

#------------------------
# Weibull baseline hazard
#------------------------

Lambda <- inv.prior.cov(X, lambda = c(0.5, 1), family = 'survival', basehaz = 'weibul')
fit_weib <- MAP.estimation(y, X, family = 'survival', Lambda = Lambda,
                           basehaz = "weibul")
# reltive risk is 1:
hazards.fun(time = time_points, p = p, theta_hat = fit_weib$theta_hat,
            basehaz = "weibul")

#-------------------------
# Gompertz baseline hazard
#-------------------------
fit_gomp <- MAP.estimation(y, X, family = 'survival', Lambda = Lambda,
                           basehaz = "gomp")
# different time points
hazards.fun(time=1:max(y*2), p = p, theta_hat = fit_gomp$theta_hat,
            basehaz = "gomp")


##---------------------------
```

```
## Prediction for a new sample
##----------------------------

## Exponentiated polynomial (poly) baseline hazard:
Lambda <- inv.prior.cov(X, lambda = c(0.5, 1), family = 'survival', basehaz = "poly")
fit_poly <- MAP.estimation(y, X, family = 'survival', Lambda = Lambda,
                           basehaz = "poly")
hazards.fun(time = y_new$time, z = X_new, theta_hat = fit_poly$theta_hat,
            basehaz = "poly", q_max = fit_poly$q_l)

## Piecewise Exponential (pwexp) baseline hazard:
Lambda <- inv.prior.cov(X, lambda = c(0.5, 1), family = 'survival', basehaz = "pwexp")
fit_pw <- MAP.estimation(y, X, family='survival', Lambda=Lambda, basehaz="pwexp",
                         min_max_times = max(y))
hazards.fun(time = y_new$time, z = X_new, theta_hat = fit_pw$theta_hat,
            basehaz = "pwexp", timax = max(y))
```

`inv.prior.cov` *Creates an inverse covariance matrix for a Gaussian prior*

Description

`inv.prior.cov` constructs a diagonal inverse covariance matrix for the Gaussian prior distribution based on the design matrix of covariates. This construction accounts for the number of regression parameters, especially when dealing with categorical covariates. For a linear model, it also includes an additional row and column to represent the variance of the measurement error. In the case of a survival model, it considers the parameters of the baseline hazard function as well.

Usage

```
inv.prior.cov(X,
              lambda = 1,
              L = 2,
              family = c("gaussian", "binomial", "survival"),
              treatment = NULL,
              treat_round = NULL,
              intercept = TRUE,
              stratified = FALSE,
              strat_par = NULL,
              center_spec = NULL,
            basehaz = c("weibul", "exp", "gomp", "poly", "pwexp", "unspecified"),
              max_order = 2,
              n_intervals = 4)
```

Arguments

`X`	design matrix of dimension $n \times p$, where n is the number of samples observed, and p is the number of predictors/variables so excluding the intercept.

`lambda` the vector used as the diagonal of the (inverse covariance) matrix that will be created by `inv.prior.cov()`. The length of the vector depends on the number of columns of `X`, type of the covariates (continuous/dichotomous or categorical), `family`, whether an `intercept` is included in the model, and whether `stratified` analysis is desired. When `stratified = FALSE`, `lambda` could be a single positive number (if all values in the vector are equal), a vector of two elements (the first is used for regression parameters including "intercept" and the second for the "sigma2" in the gaussian family or for the baseline hazard parameters in the survival case), or a vector of length equal to the number of model parameters. However, the length of `lambda` is different when `stratified = TRUE`, see 'Details' for more information. Default is `lambda = 1`, which means all model parameters are set to one.

`L` the number of centers. This argument is used only when `stratified = TRUE`. Default is `L = 2`. See 'Details' and 'Examples'.

`family` a description of the error distribution. This is a character string naming a family of the model. In the current version of the package, the family of model can be `"gaussian"` (with `identity` link function), `"binomial"` (with `logit` link function), or `"survival"`. Can be abbreviated. By default the `gaussian` family is used. In case of a linear regression model, `family = "gaussian"`, there is an extra model parameter for the variance of measurement error. While in the case of survival model, `family = "survival"`, the number of the model parameters depend on the choice of baseline hazard functions, see 'Details' for more information.

`treatment` a character string representing the name or place of the binary covariate, respectively. This covariate indicates whether the patient got the new treatment ($z_{\ell i} = 1$) or the placebo/standard treatment ($z_{\ell i} = 0$). For both the first and second rounds, it should not be 'NULL'. See 'Details'.

`treat_round` a character string representing the `'first'` or `'second'` round of estimating treatment effects. In the first round, `treat_round = 'first'`, the local estimates of the coefficients (γ_ℓ) is estimated. In the second round, `treat_round = 'second'`, the `propensity` scores and the statistical summaries (`for_ATE`) are calculated.

`intercept` logical flag for having an intercept. It is not used when `family = "survival"`. By changing the `intercept` the dimension of the inverse covariance matrix changes. If `intercept = TRUE` (the default), the output matrix created by `inv.prior.cov()` has one row and one column related to `intercept`, while if `intercept = FALSE`, the resulting matrix does not have the row and column called `intercept`.

`stratified` logical flag for performing the stratified analysis. If `stratified = TRUE`, the parameter(s) selected in the `strat_par` argument are allowed to be different across centers to deal with heterogeneity across centers. This argument should only be used when designing the inverse covariance matrix for the (fictive) combined data, i.e., the last matrix for the `Lambda` argument in `bfi()`. If `inv.prior.cov()` is used for the analysis in the local centers (to build the L first matrices for the `Lambda` argument in `bfi()`), this argument should be `FALSE`, even if the BFI analysis is stratified. Default is `stratified = FALSE`. See 'Details' and 'Examples'.

strat_par — a integer vector for indicating the stratification parameter(s). It can be used to deal with heterogeneity due to center-specific parameters. For the "binomial" and "gaussian" families it is a one- or two-element integer vector so that the values 1 and/or 2 are/is used to indicate that the "intercept" and/or "sigma2" are allowed to vary, respectively. For the "binomial" family the length of the vector should be one which refers to "intercept", and the value of this element should be 1 (to handel heterogeneity across outcome means). For "gaussian" this vector can be 1 for indicating the "intercept" only (handeling heterogeneity across outcome means), 2 for indicating the "sigma2" only (handeling heterogeneity due to nuisance parameter), and c(1, 2) for both "intercept" and "sigma2". When family = "survival", this vector can contain any combination from 1 to the maximum number of parameters of the baseline function, i.e., 1 for "exp", 2 for "weibul" and "gomp", max_order + 1 for "poly", and n_intervals for "pwexp". This argument is only used when stratified = TRUE. Default is strat_par = NULL. If stratified = TRUE, strat_par can not be NULL except when center_spec is not NULL for handeling heterogeneity due to clustering and missing covariates. See 'Details' and 'Examples'.

center_spec — a vector of L elements for representing the center specific variable. This argument is used only when stratified = TRUE and strat_par = NULL. Each element represents a specific feature of the corresponding center. There must be only one specific value or attribute for each center. This vector could be a numeric, characteristic or factor vector. Note that, the order of the centers in the vector center_spec must be the same as in the list of the argument theta_hats in the function bfi(). The used data type in the argument center_spec must be categorical. Default is center_spec = NULL. See also 'Details' and 'Examples'.

basehaz — a character string representing one of the available baseline hazard functions; exponential ("exp"), Weibull ("weibul", the default), Gompertz ("gomp"), exponentiated polynomial ("poly"), piecewise constant exponential ("pwexp"), and unspecified baseline hazard ("unspecified"). Can be abbreviated. It is only used when family = "survival".

max_order — an integer representing the maximum value of q_l, which is the order/degree minus 1 of the exponentiated polynomial baseline hazard function. This argument is only used when family = "survival" and basehaz = "poly". Default is 2.

n_intervals — an integer representing the number of intervals in the piecewise exponential baseline hazard function. This argument is only used when family = "survival" and basehaz = "pwexp". Default is 4.

Details

inv.prior.cov creates a diagonal matrix with the vector lambda as its diagonal. The argument stratified = TRUE should only be used to construct a matrix for the prior density in case of stratification in the fictive combined data. Never be used for the construction of the matrix for analysis in the centers.

When stratified = FALSE, the length of the vector lambda depends on the covariate matrix X, family, basehaz, and whether an "intercept" is included in the model. For example, if the design matrix X has p columns with continuous or dichotomous covariates, family = gaussian, and

intercept = TRUE, then lambda should have $p+2$ elements. In this case, if in X there is a categorical covariate with $q > 2$ categories, then the length of lambda increases with $q - 2$.

All values of lambda should be non-negative as they represent the inverse of the variance of the Gaussian prior. This argument is considered as the inverse of the variance of the prior distribution for: $(\beta_0, \boldsymbol{\beta})$ if family = "binomial" and intercept = TRUE; $(\beta_0, \boldsymbol{\beta}, \sigma^2)$ if family = "gaussian" and intercept = TRUE; and $(\boldsymbol{\beta}, \boldsymbol{\omega})$ if family = "survival".

If all values in the vector lambda equal, one value is enough to be given as entry. If lambda is a scalar, the function inv.prior.cov sets each value at the diagonal equal to lambda. When lambda is two dimensional: if family = "binomial", the first and second values are used for the inverse of the variance of the prior distribution for the intercept (β_0) and regression parameters ($\boldsymbol{\beta}$), respectively; If family = "gaussian", the first and second values are used for the inverse of the variance of the prior distribution for the regression parameters including the intercept ($\beta_0, \boldsymbol{\beta}$) and variance of the measurement error (σ^2), respectively; If family = "survival", the first and second values are used for the inverse of the variance of the prior distribution for the regression parameters ($\boldsymbol{\beta}$) and baseline hazard parameters (ω), respectively. But if stratified = TRUE the length of the vector lambda must be equal to the number of parameters in the combined model.

If intercept = FALSE, for the binomial family the stratified analysis is not possible therefore stratified can not be TRUE.

If stratified = FALSE, both strat_par and center_spec must be NULL (the defaults), while if stratified = TRUE only one of the two must be NULL.

If stratified = TRUE and family = "survival", strat_par = 1 refers to ω_0 when basehaz = "poly", and to ω_1 for other baseline hazards.

The output of inv.prior.cov() can be used in the main functions MAP.estimation() and bfi().

Value

inv.prior.cov returns a diagonal matrix. The dimension of the matrix depends on the number of columns of X, type of the covariates (continuous/dichotomous or categorical), intercept, family, and basehaz.

Author(s)

Hassan Pazira and Marianne Jonker
Maintainer: Hassan Pazira <hassan.pazira@radboudumc.nl>

References

Jonker M.A., Pazira H. and Coolen A.C.C. (2024). *Bayesian federated inference for estimating statistical models based on non-shared multicenter data sets*, *Statistics in Medicine*, 43(12): 2421-2438. <https://doi.org/10.1002/sim.10072>

Pazira H., Massa E., Weijers J.A.M., Coolen A.C.C. and Jonker M.A. (2025b). *Bayesian Federated Inference for Survival Models*, *Journal of Applied Statistics (Accepted)*. <https://arxiv.org/abs/2404.17464>

Jonker M.A., Pazira H. and Coolen A.C.C. (2025a). *Bayesian Federated Inference for regression models based on non-shared medical center data*, *Research Synthesis Methods*, 1-41. <https://doi.org/10.1017/rsm.2025.6>

See Also

MAP.estimation

Examples

```
#----------------
# Data Simulation
#----------------
X <- data.frame(x1=rnorm(50),                    # standard normal variable
                x2=sample(0:2, 50, replace=TRUE), # categorical variable
                x3=sample(0:1, 50, replace=TRUE)) # dichotomous variable
X$x2 <- as.factor(X$x2)
X$x3 <- as.factor(X$x3)

# The (inverse) variance value (lambda=0.05) is assumed to be
# the same for Gaussian prior of all parameters (for non-stratified)

#-------------------------------------------------
# Inverse Covariance Matrix for the Gaussian prior
#-------------------------------------------------
# y ~ Binomial with 'intercept'
inv.prior.cov(X, lambda = 0.05, family = 'binomial')
# returns a 5-by-5 matrix

# y ~ Binomial without 'intercept'
inv.prior.cov(X, lambda = 0.05, family = "binomial", intercept = FALSE)
# a 4-by-4 matrix

# y ~ Gaussian with 'intercept'
inv.prior.cov(X, lambda = 0.05, family = 'gaussian')
# returns a 6-by-6 matrix

# Survival family with 'weibul' baseline hazard
inv.prior.cov(X, lambda = c(0.05, 0.1), family = 'survival')
# returns a 6-by-6 matrix

# Survival family with 'pwexp' baseline hazard (4 intervals)
inv.prior.cov(X, lambda = 0.05, family = 'survival', basehaz = "pwexp")
# returns a 8-by-8 matrix

# Survival family with 'poly' baseline hazard
inv.prior.cov(X, lambda = c(0.05, 0.1), family = 'survival', basehaz = "poly")
# returns a 7-by-7 matrix

#--------------------
# Stratified analysis
#--------------------
# y ~ Binomial when 'intercept' varies across 3 centers:
inv.prior.cov(X, lambda = c(.2, 1), family = 'binomial', stratified = TRUE,
              strat_par = 1, L = 3)

# y ~ Gaussian when 'intercept' and 'sigma2' vary across 2 centers; y ~ Gaussian
```

```
inv.prior.cov(X, lambda = c(1, 2, 3), family = "gaussian", stratified = TRUE,
              strat_par = c(1, 2))

# y ~ Gaussian when 'sigma2' varies across 2 centers (with 'intercept')
inv.prior.cov(X, lambda = c(1, 2, 3), family='gaussian', stratified = TRUE,
              strat_par = 2)

# y ~ Gaussian when 'sigma2' varies across 2 centers (without 'intercept')
inv.prior.cov(X, lambda = c(2, 3), family = "gaussian", intercept = FALSE,
              stratified=TRUE, strat_par = 2)

#-------------------------
# Center specific covariate
#-------------------------
# center specific covariate has K = 2 categories across 4 centers; y ~ Binomial
inv.prior.cov(X, lambda = c(0.1:2), family = 'binomial', stratified = TRUE,
              center_spec = c("Iran","Netherlands","Netherlands","Iran"), L=4)

# center specific covariate has K = 3 categories across 5 centers; y ~ Gaussian
inv.prior.cov(X, lambda = c(0.5:3), family = 'gaussian', stratified = TRUE,
              center_spec = c("Medium","Big","Small","Big","Small"), L = 5)

# center specific covariate has K = 4 categories across 5 centers; y ~ Gaussian
inv.prior.cov(X, lambda = 1, family = 'gaussian', stratified = TRUE,
              center_spec = c(3,1:4), L=5)
```

`MAP.estimation` *Maximum A Posteriori estimation*

Description

`MAP.estimation` function is used (in local centers) to compute Maximum A Posterior (MAP) estimators of the parameters for Generalized Linear Models (GLM) and Survival models.

Usage

```
MAP.estimation(y,
               X,
               family = c("gaussian", "binomial", "survival"),
               Lambda,
               intercept = TRUE,
            basehaz = c("weibul", "exp", "gomp", "poly", "pwexp", "unspecified"),
               treatment = NULL,
               treat_round = NULL,
               refer_treat,
               gamma_bfi = NULL,
               RCT_propens = NULL,
               initial = NULL,
```

```
alpha = 0.1,
max_order = 2,
n_intervals = 4,
min_max_times,
center_zero_sample = FALSE,
zero_sample_cov,
refer_cat,
zero_cat,
control = list())
```

Arguments

y	response vector. If the "binomial" family is used, this argument is a vector with entries 0 (failure) or 1 (success). Alternatively, for this family, the response can be a matrix where the first column is the number of "successes" and the second column is the number of "failures". For the "survival" family, the response is a matrix where the first column is the survival time, named "time", and the second column is the censoring indicator, named "status", with 0 indicating censoring time and 1 indicating event time.
X	design matrix of dimension $n_\ell \times p$, where p is the number of covariates or predictors and n_ℓ is the number of indeviduals in the local center. If there is a categorical covariate, then the function factor() should be used to encode the covariate as a factor. Note that the order of the covariates must be the same across the centers; otherwise, the output estimates of bfi() will be incorrect.
family	a description of the error distribution. This is a character string naming a family of the model. In the current version of the package, the family of model can be "gaussian" (with identity link function), "binomial" (with logit link function), or "survival". Can be abbreviated. By default the "gaussian" family is used. In case of a linear regression model, family = "gaussian", there is an extra model parameter for the variance of measurement error. While in the case of survival model, family = "survival", the number of the model parameters depend on the choice of baseline hazard functions, see 'Details' for more information.
Lambda	the inverse variance-covariance matrix of the Gaussian distribution that is used as prior distribution for the model parameters. The dimension of the matrix depends on the number of columns of X, type of the covariates (continuous / dichotomous or categorical), family, and whether an intercept is included (if applicable). However, Lambda can be easily created by inv.prior.cov(). See inv.prior.cov for more information.
intercept	logical flag for fitting an intercept. If intercept=TRUE (the default), the intercept is fitted, i.e., it is included in the model, and if intercept = FALSE it is set to zero, i.e., it's not in the model. This argument is not used if family = "survival".
basehaz	a character string representing one of the available baseline hazard functions; exponential ("exp"), Weibull ("weibul", the default), Gompertz ("gomp"), exponentiated polynomial ("poly"), piecewise constant exponential ("pwexp"), and unspecified baseline hazard ("unspecified"). Can be abbreviated. It

is used only when family = "survival". If local sample size is large and the shape of the baseline hazard function is completely unknown, the "exponentiated polynomial" and "piecewise exponential" hazard functions would be preferred above the lower dimensional alternatives. However, if the local samples size is low, one should be careful using the "piecewise exponential" hazard function with many intervals. If basehaz = "unspecified", it means that a (semi-parametric) Cox model is considered, and the parameters (regression coefficients) are estimated using the partial log-likelihood. If treatment is not NULL, then basehaz must be set to "unspecified", as the regression coefficients are estimated using the weighted partial log-likelihood.

treatment — a character string representing the name or place of the binary covariate, respectively. This covariate indicates whether the patient got the new treatment ($z_{\ell i} = 1$) or the placebo/standard treatment ($z_{\ell i} = 0$). The treatment effect is estimated when this argument is NOT 'NULL'. If it is set to 'NULL' (the default), the treatment effect will not be estimated. For both the first and second rounds, it should not be 'NULL'. See 'Details'.

treat_round — a character string representing the 'first' or 'second' round of estimating treatment effects. In the first round, treat_round = 'first', the local estimates of the coefficients (γ_ℓ) is estimated. In the second round, treat_round = 'second', the treatment effect, propensity scores and the statistical summaries (for_ATE, only for 'binomial' and 'gaussian' families) are calculated to be sent to the central server for estimating the BFI treatment effect ($\hat{\zeta}_{BFI}$) and average treatment effects (ATEs).

refer_treat — a character string representing the reference category of the treatment variable. The reference category is considered as $z_{\ell i} = 0$. This argument is used when treatment is not 'NULL'. Default is refer_treat = levels(X$treatment)[1].

gamma_bfi — a vector specifying the BFI estimates of the coefficients received from the central server in the first round. It can be defined by the output of MAP.estimation()$theta_hat obtained from the first round. The length of gamma_bfi equals the number of regression coefficients, including the intercept if intercept=TRUE, but excluding ζ, which represents the treatment effect, as well as the nuisance parameter σ in the gaussian family and any parameters of the baseline hazard ($\boldsymbol{\omega}$) for survival. This argument is used only when the argument treatment is not 'NULL'. If treatment is not 'NULL' but gamma_bfi = NULL, then the argument RCT_propens must not be 'NULL', indicating an RCT study. See 'Details'.

RCT_propens — a vector specifying the propensity scores, which represent the probability of receiving the treatment given the covariates, which are known in the randomized studies (RCTs). For example, in a 1:1 randomized trial, the propensity scores are, by definition, equal to 1/2 (or 0.5), whereas in an unbalanced randomized trial, e.g., a 2:1 trial, the propensity scores are now 2/3 and 1/3 for the two arms, respectively. The length of RCT_propens equals to the number of individuals in the local center denoted as n_ℓ. This argument is used only when the study is a randomized control trial, i.e., the propensity scores are known for this local center. In this case, there is only 'one' round, and the argument treatment must not be 'NULL', whereas gamma_bfi = NULL. Indeed, when 'treatment' is not 'NULL', one of the arguments 'RCT_propens' or 'gamma_bfi' could be 'NULL'. See 'Details'.

`initial` a vector specifying initial values for the parameters to be optimized over. The length of `initial` is equal to the number of model parameters and thus, is equal to the number of rows or columns of `Lambda`. Since the `'L-BFGS-B'` method is used in the algorithm, these values should always be finite. Default is a vector of zeros, except for the `survival` family with the `poly` function, where it is a vector with the first p elements as zeros for coefficients ($\boldsymbol{\beta}$) and -0.5 for the remaining parameters ($\boldsymbol{\omega}$). For the `gaussian` family, the last element of the `initial` vector could also be considered negative, because the Gaussian prior was applied to $log(\sigma^2)$.

`alpha` a significance level used in the chi-squared distribution (with one degree of freedom and 1-α representing the upper quantile) to conduct a likelihood ratio test for obtaining the order of the exponentiated polynomial baseline hazard function. It is only used when `family = "survival"` and `basehaz = "poly"`. Default is 0.1. See 'Details'.

`max_order` an integer representing the maximum value of `q_l`, which is the order/degree minus 1 of the exponentiated polynomial baseline hazard function. This argument is only used when `family = "survival"` and `basehaz = "poly"`. Default is 2.

`n_intervals` an integer representing the number of intervals in the piecewise exponential baseline hazard function. This argument is only used when `family = "survival"` and `basehaz = "pwexp"`. Default is 4.

`min_max_times` a scalar representing the minimum of the maximum event times observed in the centers. The value of this argument should be defined by the central server (which has access to the maximum event times of all the centers) and is only used when `family = "survival"` and `basehaz = "pwexp"`.

`center_zero_sample` logical flag indicating whether the center has a categorical covariate with no observations/individuals in one of the categories. Default is `center_zero_sample = FALSE`.

`zero_sample_cov` either a character string or an integer representing the categorical covariate that has no samples/observations in one of its categories. This covariate should have at least two categories, one of which is the reference. It is used when `center_zero_sample = TRUE`.

`refer_cat` a character string representing the reference category. The category with no observations (the argument `zero_cat`) cannot be used as the reference in the argument `refer_cat`. It is used when `center_zero_sample = TRUE`.

`zero_cat` a character string representing the category with no samples/observations. It is used when `center_zero_sample = TRUE`.

`control` a list of control parameters. See 'Details'.

Details

`MAP.estimation` function finds the Maximum A Posteriori (MAP) estimates of the model parameters by maximizing the log-posterior density with respect to the parameters, i.e., the estimates equal the values for which the log-posterior density is maximal (the posterior mode). In other words,

`MAP.estimation()` optimizes the log-posterior density with respect to the parameter vector to obtain its MAP estimates. In addition to the model parameters (i.e., coefficients ($\boldsymbol{\beta}$) and variance error (σ_e^2) for `gaussian` or the parameters of the baseline hazard ($\boldsymbol{\omega}$) for `survival`), the curvature matrix (Hessian of the log-posterior) is estimated around the mode.

The `MAP.estimation` function returns an object of class 'bfi'. Therefore, `summary()` can be used for the object returned by `MAP.estimation()`.

For the case where `family = "survival"` and `basehaz = "poly"`, we assume that in all centers the q_ℓ's are equal. However, the order of the estimated polynomials may vary across the centers so that each center can have different number of parameters, say q_ℓ+1. After obtaining the estimates within the local centers (by using `MAP.estimation()`) and having all estimates in the central server, we choose the order of the polynomial approximation for the combined data to be the maximum of the orders of the local polynomial functions, i.e., $\max\{q_1, \ldots, q_L\}$, to approximate the global baseline hazard (exponentiated polynomial) function more accurately. This is because the higher-order polynomial approximation can capture more complex features and details in the combined data. Using the higher-order approximation ensures that we account for the higher-order moments and features present in the data while maintaining accuracy. As a result, all potential cases are stored in the `theta_A_poly` argument to be used in `bfi()` by the central server. For further information on the `survival` family, refer to the 'References' section.

The three arguments `'treatment'`, `'treat_round'`, `'refer_treat'`, `'gamma_bfi'`, and `'RCT_propens'` are related to the estimation of the treatment effect. For observational and non-randomized studies, the treatment effect is estimated in two rounds; In the first round, $\hat{\boldsymbol{\beta}}_\ell$ (or $\hat{\boldsymbol{\gamma}}_\ell$) are estimated locally and in the central server $\hat{\boldsymbol{\beta}}_{BFI}$ (or $\hat{\boldsymbol{\gamma}}_{BFI}$) is estimated and then is sent to all local centers for the second round to estimate propensity scores, weights, treatment effect and ATEs. In the first round, the argument `treatment` should not be 'NULL' and `treat_round = "first"`, while `gamma_bfi = NULL` and `RCT_propens = NULL`. Moreover, in the first round, the family must be set to `binomial`, however this is handled automatically. In the second round, local weighted MAP estimate of the treatment effects and propensity scores are estimated, and along with some summary statistics are sent to the central server to estimate the average treatment effects ATEs (in this case `treatment` and `gamma_bfi` should not be 'NULL' and `treat_round = "second"`, but `RCT_propens = NULL`). In contrast, for the randomized control trial (RCT), the treatment effect can be estimated by only one round as the propensity scores are known (in this case `treatment` and `RCT_propens` should not be 'NULL', but `gamma_bfi = NULL`). NOTE: the argument `gamma_bfi` should not include estimates of the nuisance parameter σ in the `gaussian` family or any parameters of the baseline hazard ($\boldsymbol{\omega}$) and the intercept for `survival`. For more examples on treatment effect estimation, see the 'Examples' section of `bfi`.

To solve unconstrained and bound-constrained optimization problems, the `MAP.estimation` function utilizes an optimization algorithm called Limited-memory Broyden-Fletcher-Goldfarb-Shanno with Bound Constraints (L-BFGS-B), Byrd et. al. (1995). The L-BFGS-B algorithm is a limited-memory "quasi-Newton" method that iteratively updates the parameter estimates by approximating the inverse Hessian matrix using gradient information from the history of previous iterations. This approach allows the algorithm to approximate the curvature of the posterior distribution and efficiently search for the optimal solution, which makes it computationally efficient for problems with a large number of variables.

By default, the algorithm uses a relative change in the objective function as the convergence criterion. When the change in the objective function between iterations falls below a certain threshold ('`factr`') the algorithm is considered to have converged. The convergence can be checked with the argument `convergence` in the output. See 'Value'.

In case of convergence issue, it may be necessary to investigate and adjust optimization parameters to facilitate convergence. It can be done using the `initial` and `control` arguments. By the argument `initial` the initial points of the interative optimization algorithm can be changed, and the argument `control` is a list that can supply any of the following components:

`maxit`: is the maximum number of iterations. Default is 150;

`factr`: controls the convergence of the `'L-BFGS-B'` method. Convergence occurs when the reduction in the objective is within this factor of the machine tolerance. Default for `factr` is 1e7, which gives a tolerance of about 1e-9. The exact tolerance can be checked by multiplying this value by `.Machine$double.eps`;

`pgtol`: helps to control the convergence of the `'L-BFGS-B'` method. It is a tolerance on the projected gradient in the current search direction, i.e., the iteration will stop when the maximum component of the projected gradient is less than or equal to `pgtol`, where `pgtol`≥ 0. Default is zero, when the check is suppressed;

`trace`: is a non-negative integer. If positive, tracing information on the progress of the optimization is produced. Higher values may produce more tracing information: for the method `'L-BFGS-B'` there are six levels of tracing. To understand exactly what these do see the source code of `optim` function in the **stats** package;

`REPORT`: is the frequency of reports for the `'L-BFGS-B'` method if `'control$trace'` is positive. Default is every 10 iterations;

`lmm`: is an integer giving the number of `BFGS` updates retained in the `'L-BFGS-B'` method. Default is 5.

Value

`MAP.estimation` returns a list containing the following components:

`theta_hat`	the vector corresponding to the maximum a posteriori (MAP) estimates of the parameters. For the `gaussian` family, although a Gaussian prior was applied to $\log(\sigma^2)$, the last element of this vector was back-transformed to σ^2. When `treatment` is not NULL and `treat_round = 'first'`, this is the MAP estimates of only regression coefficients (γ_ℓ) except the treatment effect ζ_ℓ. While `treatment` is not NULL and `treat_round = 'second'`, this is $\hat{\zeta}_\ell$, the weighted MAP estimate of the treatment effect ζ_ℓ in center ℓ;
`A_hat`	minus the curvature (or Hessian) matrix around the point `theta_hat`. The dimension of the matrix is the same as the argument `Lambda`;
`sd`	the vector of (posterior) standard deviation of the MAP estimates in `theta_hat`, that is `sqrt(diag(solve(A_hat)))`;
`Lambda`	the inverse variance-covariance matrix of the Gaussian distribution that is used as prior distribution for the parameters. It's exactly the same as the argument `Lambda`;
`formula`	the formula applied;
`names`	the names of the model parameters;
`n`	sample size, n_ℓ;
`np`	the number of coefficients;

`q_l` the order/degree minus 1 of the exponentiated polynomial baseline hazard function determined for the current center by the likelihood ratio test. This output argument, `q_l`, is only shown when `family = "survival"` and `basehaz = "poly"`, and will be used in the function `bfi()`;

`theta_A_poly` an array where the first component is a matrix with columns representing the MAP estimates of the parameters for different `q_l`'s, i.e., `q_l`, `q_l+1`, ..., `max_order`. The other components are minus the curvature matrices for different `q_l`'s, i.e., `q_l`, `q_l+1`, ..., `max_order`. Therefore, the first non-NA curvature matrix is equal to the output argument `A_hat`. This output argument, `theta_A_poly`, is only shown if `family = "survival"` and `basehaz = "poly"`, and will be used in the function `bfi()`;

`lev_no_ref_zer` a vector containing the names of the levels of the categorical covariate that has no samples/observations in one of its categories. The name of the category with no samples and the name of the reference category are excluded from this vector. This argument is shown when `family = "survival"` and `basehaz = "poly"`, and will be used in the function `bfi()`;

`treatment` a character string representing the name or place of the binary covariate, respectively. If it is set to 'NULL', the treatment effect will not be estimated;

`refer_treat` the reference category of the treatment. It is shown when `treatment` is not 'NULL', and can be used in the function `bfi()`;

`gamma_bfi` a vector specifying the BFI estimates of the coefficients received from the central server in the first round. If `treatment = NULL`, then `gamma_bfi` must also be 'NULL';

`RCT_propens` a vector specifying the propensity scores, which represent the probability of receiving the treatment given the covariates, which are known in the randomized studies (RCTs). If `treatment = NULL`, then `RCT_propens` must also be 'NULL';

`propensity` a vector specifying the propensity scores (the probability a patient gets the treatment given the characteristics measured at baseline) calculated by $Pr(Z_\ell = 1|X_\ell)$;

`for_ATE` a vector used in the central server to calculate the average treatment effect (ATE). For `family` of `binomial` and `gaussian`, its elements are:

first $m_{\ell 1}$, the number of patients in the treatment group, where $n_\ell = m_{\ell 1} + m_{\ell 2}$,
second $m_{\ell 2}$, the number of patients in the reference group, where $m_{\ell 2} = n_\ell - m_{\ell 1}$,
third $\sum_{i=1}^{n_\ell} z_{\ell i} y_{\ell i}$,
fourth $\sum_{i=1}^{n_\ell} (z_{\ell i} y_{\ell i})^2$,
fifth $\sum_{i=1}^{n_\ell} z_{\ell i}/e_{\ell i}$,
sixth $\sum_{i=1}^{n_\ell} z_{\ell i} y_{\ell i}/e_{\ell i}$,
seventh $\sum_{i=1}^{n_\ell} (1 - z_{\ell i})/(1 - e_{\ell i})$,
eighth $\sum_{i=1}^{n_\ell} (1 - z_{\ell i}) y_{\ell i}/(1 - e_{\ell i})$,
ninth $\sum_{i=1}^{n_\ell} (1 - z_{\ell i}) y_{\ell i}$,
but for `survival`, it's 'NULL';

`zero_sample_cov` the categorical covariate that has no samples/observations in one of its categories. It is shown when `center_zero_sample = TRUE`, and can be used in the function `bfi()`;

refer_cat	the reference category. It is shown when center_zero_sample = TRUE, and can be used in the function bfi();
zero_cat	the category with no samples/observations. It is shown when center_zero_sample = TRUE, and can be used in the function bfi();
value	the value of minus the log-likelihood posterior density evaluated at theta_hat;
family	the family used;
basehaz	the baseline hazard function used;
intercept	logical flag used to fit an intercept if TRUE, or set to zero if FALSE;
convergence	an integer value used to encode the warnings and the errors related to the algorithm used to fit the model. The values returned are: **0** algorithm has converged, **1** maximum number of iterations ('maxit') has been reached, **2** Warning from the 'L-BFGS-B' method. See the message after this value;
control	the list of control parameters used to compute the MAP estimates.

Author(s)

Hassan Pazira and Marianne Jonker
Maintainer: Hassan Pazira <hassan.pazira@radboudumc.nl>

References

Jonker M.A., Pazira H. and Coolen A.C.C. (2024). *Bayesian federated inference for estimating statistical models based on non-shared multicenter data sets*, *Statistics in Medicine*, 43(12): 2421-2438. <https://doi.org/10.1002/sim.10072>

Pazira H., Massa E., Weijers J.A.M., Coolen A.C.C. and Jonker M.A. (2025b). *Bayesian Federated Inference for Survival Models*, *Journal of Applied Statistics (Accepted)*. <https://arxiv.org/abs/2404.17464>

Jonker M.A., Pazira H. and Coolen A.C.C. (2025a). *Bayesian Federated Inference for regression models based on non-shared medical center data*, *Research Synthesis Methods*, 1-41. <https://doi.org/10.1017/rsm.2025.6>

Byrd R.H., Lu P., Nocedal J. and Zhu C. (1995). *A limited memory algorithm for bound constrained optimization*. SIAM Journal on Scientific Computing, 16, 1190-1208. <https://doi.org/10.1137/0916069>

See Also

bfi, inv.prior.cov and summary.bfi

Examples

```
###--------------###
### y ~ Gaussian ###
###--------------###

# Setting a seed for reproducibility
set.seed(11235)

# model parameters: coefficients and sigma2 = 1.5
```

```
theta <- c(1, 2, 2, 2, 1.5)

#----------------
# Data Simulation
#----------------
n   <- 30   # sample size
p   <- 3    # number of coefficients without intercept
X   <- data.frame(matrix(rnorm(n * p), n, p)) # continuous variables
# linear predictor:
eta <- theta[1] + theta[2] * X$X1 + theta[3] * X$X2 + theta[4] * X$X3
# inverse of the link function ( g^{-1}(\eta) = \mu ):
mu  <- gaussian()$linkinv(eta)
y   <- rnorm(n, mu, sd = sqrt(theta[5]))

# Load the BFI package
library(BFI)

#-----------------------------------------------
# MAP estimations for theta and curvature matrix
#-----------------------------------------------
# MAP estimates with 'intercept'
Lambda <- inv.prior.cov(X, lambda = c(0.1, 1), family = "gaussian")
(fit <- MAP.estimation(y, X, family = "gaussian", Lambda))
class(fit)
summary(fit, cur_mat = TRUE)

# MAP estimates without 'intercept'
Lambda <- inv.prior.cov(X, lambda = c(0.1, 1), family = 'gaussian',
                        intercept = FALSE)
(fit1 <- MAP.estimation(y, X, family = 'gaussian', Lambda, intercept = FALSE))
summary(fit1, cur_mat = TRUE)



###-----------------###
### Survival family ###
###-----------------###

# Setting a seed for reproducibility
set.seed(112358)

#-------------------------
# Simulating Survival data
#-------------------------
n    <- 50
beta <- 1:4
p    <- length(beta)
X    <- data.frame(matrix(rnorm(n * p), n, p)) # continuous (normal) variables

## Simulating survival data from Weibull with a predefined censoring rate of 0.3
y <- surv.simulate(Z = list(X), beta = beta, a = 5, b = exp(1.8), u1 = 0.1,
                   cen_rate = 0.3, gen_data_from = "weibul")$D[[1]][, 1:2]
```

```
#--------------------------------------
# MAP estimations with "weibul" function
#--------------------------------------
Lambda <- inv.prior.cov(X, lambda = c(0.1, 1), family = 'survival',
                        basehaz = "weibul")
fit2 <- MAP.estimation(y, X, family = 'survival', Lambda = Lambda,
                       basehaz = "weibul")
fit2
summary(fit2, cur_mat = TRUE)

#-------------------------------------
# MAP estimations with "poly" function
#-------------------------------------
Lambda <- inv.prior.cov(X, lambda = c(0.1, 1), family = 'survival',
                        basehaz = 'poly')
fit3 <- MAP.estimation(y, X, family = "survival", Lambda = Lambda,
                       basehaz = "poly")
# Degree of the exponentiated polynomial baseline hazard
fit3$q_l + 1
# theta_hat for (beta_1, ..., beta_p, omega_0, ..., omega_{q_l})
fit3$theta_A_poly[,,1][,fit3$q_l+1] # equal to fit3$theta_hat
# A_hat
fit3$theta_A_poly[,,fit3$q_l+2] # equal to fit3$A_hat
summary(fit3, cur_mat = TRUE)

#-----------------------------------------------------
# MAP estimations with "pwexp" function with 3 intervals
#-----------------------------------------------------
# Assume we have 4 centers
Lambda <- inv.prior.cov(X, lambda = c(0.1, 1), family = 'survival',
                        basehaz = 'pwexp', n_intervals = 3)
# For this baseline hazard ("pwexp"), we need to know
# maximum survival times of the 3 other centers:
max_times <- c(max(rexp(30)), max(rexp(50)), max(rexp(70)))
# Minimum of the maximum values of the survival times of all 4 centers is:
min_max_times <- min(max(y$time), max_times)
fit4 <- MAP.estimation(y, X, family = "survival", Lambda = Lambda,
                       basehaz = "pwexp", n_intervals = 3,
                       min_max_times=max(y$time))
summary(fit4, cur_mat = TRUE)


#-------------------------
# Semi-parametric Cox model
#-------------------------
Lambda <- inv.prior.cov(X, lambda = c(0.1), family = 'survival',
                        basehaz = "unspecified")
fit5 <- MAP.estimation(y, X, family = 'survival', Lambda = Lambda,
                       basehaz = "unspecified")
summary(fit5, cur_mat = TRUE)
```

n.par *The Number of Predictors, Coefficients, and Observations*

Description

n.par returns the number of regression parameters, covariates and observations present in X based on the selected family.

Usage

```
n.par(X, family = c("gaussian", "binomial", "survival"))
```

Arguments

X	design matrix of dimension $n \times p$, where n is the number of samples observed, and p is the number of predictors/covariables. It could be a matrix or a list of matrices.
family	a description of the error distribution used to specify the model. This should be a character string, either "gaussian", "binomial", or "survival". Can be abbreviated. By default the gaussian family is used.

Details

orig.names and covar.names are the same if the all covariates in X are continuous. However, if there are at least one categorical variable in X with more than two categories, they are different.

Value

n.par returns a list containing the following components:

n.reg.par	the number of regression parameters;
n.covar	the number of covariates;
n.sample	the number of samples/observations;
orig.names	the original variable names excluding dummy variable names;
covar.names	the variables names, including any dummy variable names (if applicable).

Author(s)

Hassan Pazira
Maintainer: Hassan Pazira <hassan.pazira@radboudumc.nl>

Examples

```
#--------------------
# family = "gaussian"
#--------------------

X0 <- data.frame(x1 = rnorm(50),                     # standard normal variable
                 x2 = sample(0:2, 50, replace=TRUE), # categorical variable
                 x3 = sample(0:1, 50, replace=TRUE)) # dichotomous variable
n.par(X0) # without dummy variables
X0$x2 <- as.factor(X0$x2)
X0$x3 <- as.factor(X0$x3)
n.par(X0)  # with dummy variables

X1 <- data.frame(Intercept = rep(1,30),
                 x1 = rnorm(30),                     # continuous variable
                 x2 = sample(0:2, 30, replace=TRUE)) # categorical variable
n.par(X1) # without dummy variables
X1$x2  <- as.factor(X1$x2)
n.par(X1) # without dummy variables

# a list of two data sets:
X01 <- list(X0, X1)
n.par(X01)
```

`Nurses` *Nurses' stress in different hospitals*

Description

This dataset comprises three-level simulated data extracted for a hypothetical study investigating stress levels within hospital settings. The dataset focuses on nurses working in specific wards within various hospitals. It includes several variables, such as nurse age (measured in years), nurse experience (measured in years), nurse gender (`0` for `male`, `1` for `female`), ward type (`0` for `general care`, `1` for `special care`), and hospital size (`0` for `small`, `1` for `medium`, `2` for `large`). The dataset in the package is obtained from the original dataset by leaving out some of the unused columns.

Usage

```
data(Nurses)
```

Source

https://multilevel-analysis.sites.uu.nl/datasets/

References

Hox, J., Moerbeek, M., and van de Schoot, R. (2010). *Multilevel Analysis: Techniques and Applications*, Second Edition (2nd ed.). *Routledge*. <https://doi.org/10.4324/9780203852279>

summary.bfi *Summarizing BFI Fits*

Description

Summary method for an object with class 'bfi' created by the MAP.estimation and bfi functions.

Usage

```
## S3 method for class 'bfi'
summary(object,
        cur_mat = FALSE,
        digits = max(3, getOption("digits") - 3),
        ...)
```

Arguments

object	fitted bfi object.
cur_mat	logical; if TRUE, minus the curvature matrix around the estimated parameters is returned and printed. Default is FALSE.
digits	significant digits in printout.
...	additional arguments affecting the summary produced.

Details

summary.bfi() gives information about the MAP estimates of parameters of the model. It can be used for the bfi objects built by the MAP.estimation and bfi functions.

The output of the summary method shows the details of the model, i.e. formula, family and link function used to specify the generalized linear model, followed by information about the estimates, standard deviations and credible intervals. Information about the log-likelihood posterior and convergence status are also provided.

By default, summary.bfi function does not return (minus) the curvature matrix, but the user can use cur_mat = TRUE to print it.

Value

summary.bfi returns an object of class summary.bfi, a list with the following components:

theta_hat	the component from object. The last element of this vector is the estimate of the dispersion parameter (sigma2) if family = "gaussian". See the MAP.estimation and bfi functions.
A_hat	the component from object. See the MAP.estimation and bfi functions.
sd	the component from object. If family = "gaussian", the last element of this vector is the square root of the estimated dispersion. See the MAP.estimation and bfi functions.

Lambda	the component from object. See the MAP.estimation function.
formula	the component from object. See the MAP.estimation function.
n	the component from object. See the MAP.estimation function.
np	the component from object. See the MAP.estimation function.
family	the component from object. See the MAP.estimation function.
intercept	the component from object. See the MAP.estimation function.
convergence	the component from object. See the MAP.estimation function.
control	the component from object. See the MAP.estimation function.
stratified	the component from object. See the bfi function.
estimate	the estimated regression coefficients, i.e., without the estimate sigma2.
logLikPost	the value of the log-likelihood posterior density evaluated at estimates (theta_hat).
link	the link function only for GLMs, not for the survival family. By default the gaussian family with identity link function and the binomial family with logit link function are used.
dispersion	the estimated variance of the random error, i.e., sigma2. The dispersion is taken as 1 for the binomial family.
CI	a 95% credible interval of the MAP estimates of the parameters.

Author(s)

Hassan Pazira
Maintainer: Hassan Pazira <hassan.pazira@radboudumc.nl>

See Also

MAP.estimation and bfi

Examples

```
#-------------
# y ~ Gaussian
#-------------
# model assumption:
theta <- c(1, 2, 3, 4, 1.5)  # coefficients and sigma2 = 1.5

#----------------
# Data Simulation
#----------------
n      <- 40
X      <- data.frame(x1=rnorm(n),                    # continuous variable
                     x2=sample(1:3, n, replace=TRUE)) # categorical variable
Xx2_1  <- ifelse(X$x2 == '2', 1, 0)
Xx2_2  <- ifelse(X$x2 == '3', 1, 0)
X$x2   <- as.factor(X$x2)
eta    <- theta[1] + theta[2] * X$x1 + theta[3] * Xx2_1 + theta[4] * Xx2_2
mu     <- gaussian()$linkinv(eta)
y      <- rnorm(n, mu, sd = sqrt(theta[5]))
```

```
#----------------
# MAP estimations
#----------------
Lambda <- inv.prior.cov(X, lambda = c(0.1, 0.5), family = "gaussian")
fit    <- MAP.estimation(y, X, family = "gaussian", Lambda)
class(fit)

#-------------------------
# Summary of MAP estimates
#-------------------------
summary(fit)
sumfit <- summary(fit, cur_mat = TRUE)
sumfit$estimate
sumfit$logLikPost
sumfit$dispersion
sumfit$CI
class(sumfit)
```

`surv.simulate` *Generate survival data with predefined censoring rates for proportional hazards models*

Description

`surv.simulate` simulates one or multiple (right-censored) survival datasets for proportional hazards models by simultaneously incorporating a baseline hazard function from three different survival distributions (exponential, Weibull and Gompertz), a random censoring time generated from a uniform distribution with an known/unknown upper limit, and a set of baseline covariates. When the upper limit of the uniform censoring time distribution is unknown, `surv.simulate` can be used separately to obtain the upper limit with a predefined censoring rate.

Usage

```
surv.simulate(L = 1,
              Z,
              beta,
              a,
              b,
              u1 = 0,
              u2,
              cen_rate,
              gen_data_from = c("exp", "weibul", "gomp"),
              only_u2 = FALSE,
              n.rep = 100,
              Trace = FALSE)
```

Arguments

L	the number of datasets to be generated. Default is L = 1.
Z	a list of L design matrices of dimension $n_\ell \times p$, where n_ℓ is the number of samples observed for the ℓ^{th} dataset and p is the number of covariables. When L = 1, Z can be a matrix.
beta	the vector of the (true) coefficients values, with a length of p (the number of covariates).
a	scale parameter, which should be non-negative. See 'Details' for the form of the cumulative hazard that can be used.
b	shape/location parameter, which should be non-negative. It is not used when gen_data_from = "exp". See 'Details' for the form of the cumulative hazard that can be used.
u1	a known non-negative lower limit of the uniform distribution for generating random censoring time. Default is u1 = 0. If cen_rate is not equal to 0, then u1 does not need to be defined.
u2	an non-negative upper limit of the uniform random censoring time distribution. The upper limit can be unknown (u2 = NULL, the default), or predefined. When this argument is assumed to be unknown, u2 = NULL, it is calculated by the algorithm within surv.simulate(). However, if the argument u2 is known, the censoring rate cannot be predefined (meaning there is no control over it) and is calculated based on the generated dataset. See 'Details' and 'References'.
cen_rate	a value representing the proportion of observations in the simulated survival data that are censored. The range of this argument is from 0 to 1. When the upper limit is known, cen_rate can nor be predefined. If there is no censoring (cen_rate = 0), the lower (u1) and upper (u2) limits of the uniform distribution do not need to be specified.
gen_data_from	a description of the distribution from which the time to event is generated. This is a character string and can be exponential ("exp"), Weibull ("weibul"), or Gompertz ("gomp"). Can be abbreviated. By default, the exponential distribution is used.
only_u2	logical flag for calculating only the upper limit of the uniform censoring time distribution. If only_u2 = TRUE, the dataset(s) are not generated. If only_u2 = TRUE, the arguments Z and u2 do not need to be specified, and cen_rate should not be set to 0. Default is only_u2 = FALSE.
n.rep	a scalar specifying the number of iterations. This argument is exclusively used in the case of the Gompertz distribution. Default is 100.
Trace	logical flag indicating whether the output of the desired u2 and the censoring proportion for different datasets should be produced for each iteration. It works gen_data_from = "gomp".

Details

surv.simulate function generates L simulated right-censored survival datasets from exponential, Weibull, or Gompertz distributions, incorporating the covariates, Z, distributed according to a multivariate normal distribution, with censoring time generated from a uniform distribution Uniform(u1, u2), where u1 is known but u2 can be either known or unknown.

surv.simulate() can also be used to calculate the unknown upper limit of the uniform distribution, u2, with a predefined censoring rate. To do this, set u2 = NULL and only_u2 = TRUE. In this case, the datasets are not generated; only u2 is.

surv.simulate() uses a root-finding algorithm to select the censoring parameter that achieves predefined censoring rates in the simulated survival data.

When gen_data_from = "exp":

- the cumulative baseline hazard function is considered as $\Lambda_0 = at$,
- the event time for the ℓ^{th} dataset, T_ℓ, is computed by $-log(u)\ exp(-Z_\ell \boldsymbol{\beta})/a$, where u follows a standard uniform distribution;

For gen_data_from = "weibul":

- the cumulative hazard function is as $\Lambda_0 = at^b$,
- the event time is computed by $T_\ell = (-log(u)\ exp(-Z_\ell \boldsymbol{\beta})/a)^{1/b}$, where u follows a standard uniform distribution;

For gen_data_from = "gomp":

- the cumulative hazard function is as $\Lambda_0 = a(exp(bt) - 1)/b$,
- the event time is computed by $T_\ell = \log(1 - log(u)\ exp(-Z_\ell \boldsymbol{\beta})b/a)/b$, where u follows a standard uniform distribution;

Finally the survival time is obtained by $\tilde{T}_\ell = \min\{T_\ell, C_\ell\}$.

The function will be updated for gen_data_from = "gomp".

Value

surv.simulate returns a list containing the following components:

D	a list of L data frames, with dimension $n_\ell \times (p + 2)$. The first and second columns, named time and status, contain the simulated survival time and the censoring indicator, respectively, where 0 means censored and 1 means uncensored;
censor_propor	the vector of censoring proportions in the simulated datasets D, containing L elements;
u1	the lower limit of the uniform distribution used to generate random censoring times with a predefined censoring rate. Sometimes this output is less than the value entered by the user, as it is adjusted to achieve the desired amount of censoring rate;
u2	the upper limit of the uniform distribution used to generate random censoring times. If u2 = NULL, this output will be the estimated upper limit necessary to achieve the desired censoring rate across the L datasets.

Author(s)

Hassan Pazira
Maintainer: Hassan Pazira <hassan.pazira@radboudumc.nl>

References

Pazira H., Massa E., Weijers J.A.M., Coolen A.C.C. and Jonker M.A. (2025b). *Bayesian Federated Inference for Survival Models*, *Journal of Applied Statistics (Accepted)*. <https://arxiv.org/abs/2404.17464>

See Also

MAP.estimation

Examples

```
# Setting a seed for reproducibility
set.seed(1123)

#-------------------------
# Simulating Survival data
#-------------------------
N    <- c(7, 10, 13) # the sample sizes of 3 datasets
beta <- 1:4
p    <- length(beta)
L    <- 3

# Define a function to generate multivariate normal samples
mvrnorm_new <- function(n, mu, Sigma) {
    pp <- length(mu)
    e <- matrix(rnorm(n * pp), nrow = n)
    return(crossprod(t(e), chol(Sigma)) + matrix(mu, n, pp, byrow = TRUE))
}
Z <- list()
for (z in seq_len(L)) {
    Z[[z]] <- mvrnorm_new(n = N[z], mu = rep(0, p),
                          Sigma = diag(rep(1, p),p))
    colnames(Z[[z]]) <- paste0("Z_",seq_len(ncol(Z[[z]])))
}

# One simulated dataset from exponential distribution with no censoring:
surv_data <- surv.simulate(Z = Z[[1]], beta = beta, a = exp(-.9),
                           cen_rate = 0, gen_data_from = "exp")
surv_data
surv_data$D[[1]][,1:2] # The simulated survival data

# Calculate only 'u2' with a predefined censoring rate of 0.4:
u2_new <- surv.simulate(Z = Z[1:2], beta = beta, a = exp(-.9),
                        b = exp(1.8), u1 = 0.1, only_u2 = TRUE,
                        cen_rate = 0.4, gen_data_from = "weibul")$u2
u2_new

# Two simulated datasets with a known 'u2':
# Using 'u2_new' to help control over censoring rate (was chosen 0.4)
surv.simulate(Z = Z[1:2], beta = beta, a = exp(-.9), b = exp(1.8),
              u1 = 0.05, u2 = u2_new, gen_data_from = "weibul")

# Three simulated datasets from 'weibul' with an unknown 'u2':
```

```
surv.simulate(Z = Z, beta = beta, a = exp(-1), b = exp(1),
               u1 = 0.01, cen_rate = 0.3, gen_data_from = "weibul")

# Two simulated datasets from 'gomp' with unknown 'u2' and censoring rate of 0.3:
surv.simulate(Z = Z[2:3], beta = beta, a = exp(1), b = exp(2), u1 = 0.1,
              cen_rate = 0.3, gen_data_from = "gomp", Trace = TRUE)
```

`trauma` *Trauma patients from different hospitals*

Description

This data set consists of data of 371 trauma patients from three hospitals. The binary variable `mortality` is used as an outcome, and variables `age`, `sex`, the Injury Severity Score (`ISS`, ranging from 1 (low) to 75 (high)) and the Glasgow Coma Scale (`GCS`, which expresses the level of consciousness, ranging from 3 (low) to 15 (high)) are used as covariates. There are three types of hospitals: peripheral hospital without a neuro-surgical unit (`Status = 1`), peripheral hospital with a neuro-surgical unit (`Status = 2`), and academic medical center (`Status = 3`). Originally, the data come from a multi center study collected with a different aim. For educational purposes minor changes have been made, see the references below.

Usage

```
data(trauma)
```

References

Jonker M.A., Pazira H. and Coolen A.C.C. (2024). *Bayesian federated inference for estimating statistical models based on non-shared multicenter data sets*, *Statistics in Medicine*, 43(12): 2421-2438. <https://doi.org/10.1002/sim.10072>

Draaisma J.M.Th, de Haan A.F.J., Goris R.J.A. (1989). *Preventable Trauma Deaths in the Netherlands - A prospective Multicentre Study*, The journal of Trauma, Vol. 29(11), 1552-1557.

Index