Package 'BuyseTest'

August 17, 2016

Type Package

Title Generalized Pairwise Comparisons

Version 1.0

Date 2016-08-15

Author Brice Ozenne[aut, cre], Julien Peron[aut]

Maintainer brice ozenne <brice.ozenne@orange.fr>

Description Implementation of the Generalized Pairwise Comparisons. This test enables to compare two groups of observations in randomized trials(e.g treated vs. control patients) on several prioritized outcomes. Pairwise comparisons require consideration of all possible pairs of individuals, one taken from the treatment group and the other taken from the control group. The outcomes of the two individuals forming a pair are compared. Thresholds of minimal clinically significant differences can be defined. It is possible to analyse simultaneously several outcomes by prioritizing the variables that capture them. The highest priority is assigned to the variable considered the most clinically relevant.
A natural way of handling uninformative or neutral pairs is to consider the outcomes in descending order of priority: whenever a pair is uninformative or neutral for an outcome of higher priority, the outcomes of lower priority are examined In the case of time-to-event endpoint, four methods to handle censored observations are available in this package (Gehan, Peto, Efron, and Peron).

License GPL-3

Depends R (>= 2.10), survival, Rcpp, data.table, snowfall

Imports methods, lava, stats, stats4, tcltk, utils

Suggests testthat

LinkingTo Rcpp, RcppArmadillo

Collate 'BuyseTest-package.R' 'FCT_BuyseTest.R' 'FCT_Valid.R' 'FCT_simul.R' 'FCT_support.R' 'FCTi-computation.R' 'FCTi-initilization.R' 'FCTi-print.R' 'OBJECT_BuyseTest.R' 'METHOD_get.R' 'METHOD_summary.R' 'METHOD_print.R' 'RcppExports.R'

RoxygenNote 5.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2016-08-17 10:40:20

R topics documented:

`BuyseRes-class` *Class "BuyseRes" (output of BuyseTest)*

Description

A `BuyseTest` output is reported in a `BuyseRes` object.

Slots

`levels.treatment` the name of each group. *character vector*.

`endpoint` the name of the endpoint. *character vector*.

`strata` the name of the strata *character vector*.

`threshold` the threshold associated to each endpoint *numeric vector*.

`n_pairs` the total number of pairs. *integer*.

`count_favorable` the probability for a random pair to be favorable for each strata (in rows) and each endpoint (in columns). *matrix*.

`count_unfavorable` the probability for a random pair to be unfavorable for each strata (in rows) and each endpoint (in columns). *matrix*.

`count_neutral` the probability for a random pair to be neutral for each strata (in rows) and each endpoint (in columns). *matrix*.

`count_uninf` the probability for a random pair to be uninformative for each strata (in rows) and each endpoint (in columns). *matrix*.

`index_neutralT` the index in the dataset of the treatment observations from remaining neutral pairs. *integer vector*.

`index_neutralC` the index in the dataset of the control observations from remaining neutral pairs. *integer vector*.

`index_uninfT` the index in the dataset of the treatment observations from remaining uninformative pairs. *integer vector*.

`index_uninfC` the index in the dataset of the control observations from remaining uninformative pairs. *integer vector*.

`delta` the chance of a better outcome (net difference between the probability for a random pair to be favorable minus the probability to be unfavorable divided by the total number of pairs) for each strata (in rows) and each endpoint (in columns). *matrix*.

`delta_boot` the chance of a better outcome within each strata (first dimension), over the endpoint (second dimension) and for each bootstrap dataset (third dimension). *array*.

`Delta_quantile` the randomization test-based 2.5% and the 97.5% quantiles of the cumulative chance of a better outcome (in rows) for each endpoint (in columns). *matrix*.

`p.value` the p.value associated to the chance of a better outcome at each prioritized endpoint. *numeric vector*.

See Also

`BuyseTest` for the function computing generalized pairwise comparisons.
`BuyseRes-summary` for the summary of the BuyseTest function results

Examples

```
n.Treatment_testBin <- 500
n.Control_testBin <- 500
prob.Treatment_testBin <- c(0.5,0.75)
prob.Control_testBin <- c(0.5,0.25)

set.seed(10)
data_testBin <- data.frame(treatment=c(rep(1,n.Treatment_testBin),rep(0,n.Treatment_testBin)))
data_testBin$endpoint1 <- c(rbinom(n.Treatment_testBin,size=1,prob=prob.Treatment_testBin[1]),
                        rbinom(n.Control_testBin,size=1,prob=prob.Control_testBin[1]))
data_testBin$endpoint2 <- c(rbinom(n.Control_testBin,size=1,prob=prob.Treatment_testBin[2]),
                        rbinom(n.Control_testBin,size=1,prob=prob.Control_testBin[2]))
data_testBin$strata <- rbinom(n.Treatment_testBin+n.Control_testBin,size=4,prob=0.5)

#### no strata, n.bootsrap=0
BuyseTest_object <- BuyseTest(data=data_testBin,endpoint=c("endpoint1","endpoint2"),
                              treatment="treatment", type=c("bin","bin"))

class(BuyseTest_object)
```

`BuyseRes-getCount` *get Method for Class "BuyseRes"*

Description

Extract the number of pairs.

Usage

```
getCount(object, type)

## S4 method for signature 'BuyseRes'
getCount(object, type)
```

Arguments

object	an R object of class BuyseRes, i.e., output of BuyseTest
type	the type of pairs to be counted. Can be "favorable", "unfavorable", neutral, or uninf. Can also be "all" to select all of them.

Value

A "vector" containing the number of pairs

Examples

```
dt <- simulBT(1e2)
BT <- BuyseTest(data=dt,endpoint="Y_TTE1",treatment="Treatment",
                type="timeToEvent",censoring="event1", n.bootstrap = 0)
getCount(BT)
getCount(BT, type = "favorable")
```

BuyseRes-show *Show Method for Class "BuyseRes"*

Description

Display the main results stored in a BuyseRes object.

Usage

```
## S4 method for signature 'BuyseRes'
show(object)
```

Arguments

object	an R object of class BuyseRes, i.e., output of BuyseTest

See Also

BuyseTest for performing a generalized pairwise comparison.
BuyseRes-summary for a more detailed presentation of the BuyseRes object.

Examples

```
n.Treatment_testBin <- 500
n.Control_testBin <- 500
prob.Treatment_testBin <- c(0.5,0.75)
prob.Control_testBin <- c(0.5,0.25)

set.seed(10)
data_testBin <- data.frame(treatment=c(rep(1,n.Treatment_testBin),rep(0,n.Treatment_testBin)))
data_testBin$endpoint1 <- c(rbinom(n.Treatment_testBin,size=1,prob=prob.Treatment_testBin[1]),
                        rbinom(n.Control_testBin,size=1,prob=prob.Control_testBin[1]))
data_testBin$endpoint2 <- c(rbinom(n.Control_testBin,size=1,prob=prob.Treatment_testBin[2]),
                        rbinom(n.Control_testBin,size=1,prob=prob.Control_testBin[2]))
data_testBin$strata <- rbinom(n.Treatment_testBin+n.Control_testBin,size=4,prob=0.5)

#### no strata
## Not run:
  BuyseTest_object <- BuyseTest(data=data_testBin,endpoint=c("endpoint1","endpoint2"),
                          treatment="treatment",type=c("bin","bin"),n.bootstrap=10000)

## End(Not run)


  BuyseTest_object
```

BuyseRes-summary *Summary Method for Class "BuyseRes"*

Description

Summarize the results from the BuyseTest function.

Usage

```
summary(object, ...)

## S4 method for signature 'BuyseRes'
summary(object, show = "pc", strata = NULL,
  digit = c(2, 3))
```

Arguments

object	an R object of class BuyseRes, i.e., output of BuyseTest
...	arguments to be passed from the generic method to the class specific method [not relevant to the user]
show	the type of result to print. Can be "nb" or "pc" or NULL (nothing is printed). Default is "pc".

strata	the name of the strata to be displayed *character vector*. Default is "global" which displays the overall results.
digit	the number of digit to use for printing the results. *integer*. Default is 3.

Details

WARNING : the confidence interval is computed using quantiles of the distribution of the cumulative proportion in favor of the treatment under the null hypothesis. It thus may not be valid if this hypothesis is rejected. In particular, if the cumulative proportion in favor of the treatment is close to 1, the upper limit of the confidence interval may exceed 1.

Value

A "List" composed of two matrices containing the endpoint (and the strata) in rows and the results of the pairwise comparison in columns:

- [[nb]] : The favorable, unfavorable, neutral and uninformative pairs are reported in number of pairs classified in each category ("n.favorable","n.unfavorable","n.neutral","n.uninformative").
- [[pc]] : The favorable, unfavorable, neutral and uninformative pairs are reported in percentage of pair classified in each category ("pc.favorable","pc.unfavorable","pc.neutral","pc.uninformative").

"delta" indicates for each endpoint (and each strata) thechance of a better outcome and "Delta" the cumulative chance of a better outcome.
The confidence interval and the p.value are given for the cumulative chance of a better outcome ("CIinf.Delta","CIsup.Delta","p.value")

See Also

BuyseTest for performing a generalized pairwise comparison.
BuyseRes-class for a presentation of the BuyseRes object.

Examples

```
n.Treatment_testBin <- 500
n.Control_testBin <- 500
prob.Treatment_testBin <- c(0.5,0.75)
prob.Control_testBin <- c(0.5,0.25)

set.seed(10)
data_testBin <- data.frame(treatment=c(rep(1,n.Treatment_testBin),rep(0,n.Treatment_testBin)))
data_testBin$endpoint1 <- c(rbinom(n.Treatment_testBin,size=1,prob=prob.Treatment_testBin[1]),
                            rbinom(n.Control_testBin,size=1,prob=prob.Control_testBin[1]))
data_testBin$endpoint2 <- c(rbinom(n.Control_testBin,size=1,prob=prob.Treatment_testBin[2]),
                            rbinom(n.Control_testBin,size=1,prob=prob.Control_testBin[2]))
data_testBin$strata <- rbinom(n.Treatment_testBin+n.Control_testBin,size=4,prob=0.5)

#### no strata
## Not run:
  BuyseTest_object <- BuyseTest(data=data_testBin,endpoint=c("endpoint1","endpoint2"),
```

```
                                treatment="treatment",type,=c("bin","bin"),n.bootstrap=10000)

## End(Not run)


  summary_BuyseTest_object <- summary(BuyseTest_object)
```

BuyseTest *Generalized Pairwise Comparisons*

Description

Performs Generalized Pairwise Comparisons for binary, continuous and time-to-event outcomes.

Usage

```
BuyseTest(data, treatment, endpoint, type, threshold = NULL, strata = NULL,
  censoring = NULL, method = "Peron", n.bootstrap = 0,
  prob.alloc = NULL, stratified = FALSE, alternative = "two.sided",
  seed = 10, cpus = 1, trace = 3)
```

Arguments

data	A data.frame containing the variables.
treatment	the name of the treatment variable identifying the control and the experimental group. *character*.
endpoint	the name of the endpoint variable(s). *character vector*.
type	the type of each endpoint. *character vector*. Can be "binary", "continuous" or "timeToEvent".
threshold	the thresholds, one for each endpoint variable. *numeric vector*. Default is NULL indicating no threshold.
strata	the name of the strata variable(s). *numeric vector*. Default is NULL indicating only one strata.
censoring	the name of the censoring variable(s), one for each endpoint. *character vector*. Default is NULL.
method	Is defined when at least one time-to-event outcome is analyzed. Defines the method used to handle pairs which can not be decidely classified as favorable, unfavorable, or neutral because of censored observations. Can be "Gehan", "Peto", "Efron", or "Peron". See details.
n.bootstrap	the number of bootstrap samples used for computing the confidence interval and the p.values. *integer*. Default is 0 meaning no bootstrap (and thus only ponctual estimation).
prob.alloc	the resampling probability for assignement to the experimental group in the bootstrap samples. *double*. Default is NULL indicating to use the proportion of patients in the experimental group.

stratified	Should the boostrap be a stratified bootstrap? *logical*. Default is FALSE.
alternative	a *character* specifying the alternative hypothesis. Must be one of "two.sided", "greater" or "less". Default is "two.sided".
seed	the seed to consider for the bootstrap. *integer*. Default is 10.
cpus	the number of CPU to use. *integer*. Default is 1.
trace	Should the execution of the function be traced ? *integer*. Default is 3.

Details

Treatment: The variable corresponding to treatment in data must have only two levels (e.g. 0 and 1).
Endpoint, threshold, censoring, and type: Arguments endpoint, threshold, censoring and type must have the same length.
threshold must be NA for binary endpoints and positive for continuous or time to event endpoints. censoring must be NA for binary or continuous endpoints and indicate a variable in data for time to event endpoints. Short forms for endpoint type are "bin" (binary endpoint), "cont" (continuous endpoint), "TTE" (time-to-event endpoint).

Bootstrap: The number of bootstrap replications (argument n.bootstrap) must be specified to enable the computation of the confidence intervals and the p.value. A large number of bootstrap samples (e.g. n.bootstrap=10000) are needed to obtain accurate CI and p.value. See (Buyse et al., 2010) for more details.

Trace: 3 reports all messages 2 reports all messages except silent parallelization messages, 1 reports only the percentage of advancement of the bootstrap, and 0 remains silent.

cpus parallelization: Argument cpus can be set to "all" to use all available cpus. The parallelization relies on the *snowfall* package (function *sfClusterApplyLB*). The detection of the number of cpus relies on the detectCores function from the *parallel* package .

Dealing with neutral or uninformative pairs: Neutral pairs correspond to pairs for which the difference between the endpoint of the control observation and the endpoint of the treatment observation is (in absolute value) below the threshold. When threshold=0, neutral pairs correspond to pairs with equal outcome.
Uninformative pairs correspond to pairs for which the censoring prevend from classifying them into favorable, unfavorable or neutral. Neutral or uninformative pairs for an endpoint with priority l are, when available, analysed on the endpoint with priority l-1.

Method: Pairs which can not be decidely classified as favorable, unfavorable, or neutral because of censored observations can be classified uninformative (method="Gehan"). Another solution is to estimate the probability for such pair to be classified as favorable, unfavorable, or neutral based on the survival functions. method="Peto" estimate these probabilities using the common Kaplan-Meier estimator of the survival function for treated and control patients. method="Efron", and method="Peron" estimate these probabilities using separate Kaplan-Meier estimators of the survival functions for the two groups of patients. When the largest observation is censored, it is not possible to estimate the survival probability by the Kaplan-Meier estimator beyond this time point. method="Efron" treats the largest observations in each patient group as if it were uncensored. method="Peron" treats the probability of survival beyond the last observation as NA, resulting in a non null probability that the pair is uninformative

Value

An R object of class BuyseRes.

References

Marc Buyse (2010) Generalized pairwise comparisons of prioritized endpoints in the two-sample problem. *Statistics in Medicine* **vol. 29** 3245-3257
Efron B (1967) The two sample problem with censored data *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability* **vol. 4** 831-583
Peto R, Peto J (1972) Asymptotically efficient rank invariant test procedures *J R Stat Soc A* **vol. 135(2)** 185-198
Gehan EA (1965) A generalized two-sample Wilcoxon test for doubly censored data *Biometrika* **vol. 52(3)** 650-653

See Also

BuyseRes-summary for a summary of the results of generalized pairwise comparison.
BuyseRes-class for a presentation of the BuyseRes object.
constStrata to create a strata variable from several clinical variables.

Examples

```
#### real example : Veteran dataset of the survival package ####
#### Only one endpoint. Type = Time-to-event. Thresold = 0. Stratfication by histological subtype
#### method = "Gehan"

## Not run:
  data(veteran,package="survival")
  library(BuyseTest)
  BT_Gehan <- BuyseTest(data=veteran,endpoint="time",treatment="trt",strata="celltype",
                        type="timeToEvent",censoring="status",threshold=0,
                        n.bootstrap=10000,method="Gehan",cpus="all")

  summary_Gehan <- summary(BT_Gehan)

  #### method = "Peron"

  BT_Peron <- BuyseTest(data=veteran,endpoint="time",treatment="trt",strata="celltype",
                        type="timeToEvent",censoring="status",threshold=0,
                        n.bootstrap=1000,method="Peron",cpus="all")

  summary_Peron <- summary(BT_Peron)

## End(Not run)

#### Several endpoints :
#######Survival, a time-to-event endpoint
#######Toxicity, a continuous/ordinal endpoint : 6 grades of maximal adverse event
```

```
set.seed(10)

n.Treatment <- 100
n.Control <- 100
prob.Treatment_TOX <- c(0.5,0.25,0.10,0.075,0.05,0.025)
prob.Control_TOX <- c(0.7,0.15,0.05,0.05,0.025,0.025)

lambda.Treatment_TTE <- 0.6
lambda.Control_TTE <- 1

data_test <- data.frame(treatment=c(rep(1,n.Treatment),
                                    rep(0,n.Control)  ))
data_test$toxicity <- c(apply(rmultinom(n.Treatment,size=1,
                                        prob=prob.Treatment_TOX)==1,2,which),
                        apply(rmultinom(n.Control,size=1,
                                        prob=prob.Control_TOX)==1,2,which))

data_test$toxicityInv <-6-data_test$toxicity

data_test$EventTime <- c(rexp(n.Treatment,rate=lambda.Treatment_TTE),
                         rexp(n.Control,rate=lambda.Control_TTE))
data_test$CensoringTime <- c(rexp(n.Treatment,rate=lambda.Treatment_TTE),
                             rexp(n.Control,rate=lambda.Control_TTE))
data_test$CensoringTime[data_test$CensoringTime>4] <- 4

data_test$Survival <- apply(data_test[,c("EventTime","CensoringTime")],1,min)
data_test$event <- as.numeric(apply(data_test[,c("EventTime","CensoringTime")],
                                    1,which.min)==1)

resKM_tempo <- survfit(Surv(data_test[,"Survival"],data_test[,"event"])~data_test$treatment)
plot(resKM_tempo)

all.endpoints <- c("Survival","toxicityInv","Survival","toxicityInv","Survival","toxicityInv")

#### method = "Gehan".
## Not run:
BT_Gehan <- BuyseTest(data=data_test,method="Gehan",
                      endpoint=all.endpoints,
                      treatment="treatment",
                      censoring=c("event",NA,"event",NA,"event",NA),
                      type=c("TTE","cont","TTE","cont","TTE","cont"),
                    threshold=c(1.5,3,0.75,2,0.25,1),n.bootstrap=100,trace=2,cpus="all")

## End(Not run)

summary(BT_Gehan)

#### method = "Peron".

## Not run:
  BT_Peron <- BuyseTest(data=data_test,method="Peron",
                      endpoint=all.endpoints,
                      treatment="treatment",
```

```
                      censoring=c("event",NA,"event",NA,"event",NA),
                      type=c("TTE","cont","TTE","cont","TTE","cont"),
                    threshold=c(1.5,3,0.75,2,0.25,1),n.bootstrap=100,trace=2,cpus="all")

## End(Not run)

summary(BT_Peron)
```

`BuyseTest_cpp` *C++ function performing the pairwise comparison over several endpoints.*

Description

`BuyseTest_Gehan_cpp` and `BuyseTest_PetoEfron_cpp` functions calls for each endpoint and each strata the pairwise comparison function suited to the type of endpoint and store the resuts.

Usage

```
BuyseTest_Gehan_cpp(Treatment, Control, threshold, type, delta_Treatment,
  delta_Control, D, returnIndex, strataT, strataC, n_strata, n_TTE)

BuyseTest_PetoEfronPeron_cpp(Treatment, Control, threshold, type,
  delta_Treatment, delta_Control, D, returnIndex, strataT, strataC, n_strata,
  n_TTE, Wscheme, index_survivalM1, threshold_TTEM1, list_survivalT,
  list_survivalC, PEP)
```

Arguments

`Treatment`	A matrix containing the values of each endpoint (in columns) for the treatment observations (in rows). *const arma::mat&*. Must have D columns.
`Control`	A matrix containing the values of each endpoint (in columns) for the control observations (in rows). *const arma::mat&*.
`threshold`	Store the thresholds associated to each endpoint. *const NumericVector&*. Must have length D. The threshold is ignored for binary endpoints. Must have D columns.
`type`	The type of each endpoint (1 binary, 2 continuous, 3 TTE). *const IntegerVector&*. Must have length D.
`delta_Treatment`	A matrix containing in the type of event (0 censoring, 1 event) for each TTE endpoint (in columns) and treatment observations (in rows). *const arma::mat&* containing binary integers. Must have n_TTE columns. Ignored if n_TTE equals 0.
`delta_Control`	A matrix containing the nature of observations in the control group (in rows) (0 censoring, 1 event) for each TTE endpoint (in columns) . *const arma::mat&* containing binary integers. Must have n_TTE columns. Ignored if n_TTE equals 0.

D	The number of endpoints. Strictly positive *const int*.
returnIndex	Should the indexes of the neutral or uninformative pairs be returned. *const bool*.
strataT	A list containing the indexes of treatment observations belonging for each strata. *List&* of vector containing positive integers.
strataC	A list containing the indexes of control observations belonging for each strata. *List&* of vector containing positive integers.
n_strata	The number of strata . Strictly positive *const int*.
n_TTE	The number of time-to-event endpoints. Positive *const int*.
Wscheme	The matrix describing the weighting strategy. For each endpoint (except the first) in column, weights of each pair are initialized at 1 and multiplyied by the weight of the endpoints in raws where there is a 1. *const arma::mat&*. Must have n_TTE lines and D-1 columns.
index_survivalM1	The position, among all the survival endpoints, of the last same endpoint (computed with a different threshold). If it is the first time that the TTE endpoint is used it is set to -1. *const IntegerVector*. Must have length n_TTE.
threshold_TTEM1	The previous latest threshold of each TTE endpoint. When it is the first time that the TTE endpoint is used it is set to -1. *const NumericVector*. Must have length n_TTE.
list_survivalT	A list of matrix containing the survival estimates (-threshold, 0, +threshold ...) for each event of the treatment group (in rows). *List&*. Must have length n_TTE. Each matrix must have 3 (if method is Peto, only one survival function is computed) or 11 (if method is Efron or Peron, two survival functions are computed) columns. Ignored if method is Gehan.
list_survivalC	A list of matrix containing the survival estimates (-threshold, 0, +threshold ...) for each event of the control group (in rows). *List&*. Must have length n_TTE. Each matrix must have 3 (if method is Peto) or 11 (if method is Efron or Peron) columns. Ignored if method is Gehan.
PEP	The type of method used to compare censored pairs (1 Peto, 2 Efron, 3 Peron).*const int*.

`constStrata` *Strata creation*

Description

Create strata from several variables.

Usage

```
constStrata(data, strata, sep = ".", lex.order = FALSE, trace = TRUE,
  as.numeric = FALSE)
```

Arguments

`data`	A `data.frame` containing the variables.
`strata`	A vector of the variables capturing the stratification factors. *character vector*.
`sep`	string to construct the new level labels by joining the constituent ones. *character*. Default is `"."`
`lex.order`	Should the order of factor concatenation be lexically ordered ? *logical*. Default is FALSE.
`trace`	Should the execution of the function be traced ? *logical*. Default is `TRUE`.
`as.numeric`	Should the strata be converted from factors to numeric? *logical*. Default is `FALSE`.

Details

This function uses the `interaction` function from the *base* package to form the strata.

Value

A *factor vector* or a *numeric vector*.

Examples

```
data(veteran,package="survival")

# strata with two variables : celltype and karno
veteran$strata1 <- constStrata(veteran,c("celltype","karno"))
table(veteran$strata1)

# strata with three variables : celltype, karno and age dichotomized at 60 years
veteran$age60 <- veteran$age>60
veteran$age60 <- factor(veteran$age60,labels=c("<=60",">60")) # convert to factor with labels
veteran$strata2 <- constStrata(veteran,c("celltype","karno","age60"))
table(veteran$strata2) # factor strata variable

veteran$strata2 <- constStrata(veteran,c("celltype","karno","age60"),as.numeric=TRUE)
table(veteran$strata2) # numeric strata variable
```

`Simulation function` *Simulation of data for the BuyseTest*

Description

Simulate binary, continuous or time to event data, possibly with strata.

Usage

```
simulBT(n.T, n.C = NULL, n.strata = NULL, format = "data.table",
  argsBin = list(), argsCont = list(), argsTTE = list())

simulBT_bin(n.T, n.C = NULL, p.T = 0.5, p.C = NULL, n.strata = NULL,
  format = "data.table")

simulBT_cont(n.T, n.C = NULL, mu.T = 0, sigma.T = 1, mu.C = NULL,
  sigma.C = NULL, n.strata = NULL, format = "data.table")

simulBT_TTE(n.T, n.C = NULL, rates.T = 2, rates.C = NULL,
  rates.Censor = 1, n.strata = NULL, sigma.C = NULL,
  format = "data.table")
```

Arguments

n.T	number of patients in the treatment arm
n.C	number of patients in the control arm
n.strata	number of strata. NULL indicates no strata
format	the format of the output. Can be "data.table", "data.frame" or "matrix"
argsBin	a list of arguments to be passed to simulBT_bin. They specify the distribution parameters of the binary endpoints
argsCont	a list of arguments to be passed to simulBT_continous. They specify the distribution parameters of the continuous endpoints
argsTTE	a list of arguments to be passed to simulBT_TTE. They specify the distribution parameters of the time to event endpoints
p.T	probability of event of each endpoint (binary endpoint, treatment group)
p.C	same as p.T but for the control group
mu.T	expected value of each endpoint (continuous endpoint, treatment group)
sigma.T	standard deviation of the values of each endpoint (continuous endpoint, treatment group)
mu.C	same as mu.C but for the control group
sigma.C	same as sigma.T but for the control group
rates.T	hazard corresponding to each endpoint (time to event endpoint, treatment group)
rates.C	same as rates.T but for the control group
rates.Censor	same as rates.T but for the censoring.

Details

Built on the lvm and sim functions from the lava package.

Examples

```
simulBT(1e3)
simulBT(1e3, argsBin = list(p.T = c(3:5/10)), argsCont = NULL, argsTTE = NULL)
simulBT(1e3, argsBin = NULL, argsCont = list(mu.T = c(3:5/10), sigma.T = rep(1,3)), argsTTE = NULL)
simulBT(1e3, argsBin = NULL, argsCont = NULL,
        argsTTE = list(rates.T = c(3:5/10), rates.Censor = rep(1,3)))
```

---

validFCTs *Check argument in functions*

Description

Check the validity of the arguments in functions

Usage

```
validCharacter(value1, name1 = as.character(substitute(value1)), validLength,
  validValues = "character", refuse.NULL = TRUE,
  refuse.duplicates = FALSE, method = NULL, addPP = TRUE)

validClass(value1, name1 = as.character(substitute(value1)), validClass,
  superClasses = TRUE, method = NULL, addPP = TRUE)

validDimension(value1, value2 = NULL,
  name1 = as.character(substitute(value1)),
  name2 = as.character(substitute(value2)), validDimension = NULL,
  type = c("NROW", "NCOL"), method = NULL, addPP = TRUE)

validInteger(value1, name1 = as.character(substitute(value1)), validLength,
  validValues = NULL, min = NULL, max = NULL, refuse.NA = TRUE,
  refuse.NULL = TRUE, refuse.duplicates = FALSE, method = NULL,
  addPP = TRUE)

validLogical(value1, name1 = as.character(substitute(value1)), validLength,
  refuse.NULL = TRUE, refuse.NA = TRUE, method = NULL, addPP = TRUE)

validNames(value1, name1 = as.character(substitute(value1)),
  refuse.NULL = TRUE, validLength = NULL, validValues = NULL,
  requiredValues = NULL, refuse.values = NULL, method = NULL,
  addPP = TRUE)

validNumeric(value1, name1 = as.character(substitute(value1)), validLength,
  validValues = NULL, min = NULL, max = NULL, refuse.NA = TRUE,
  refuse.NULL = TRUE, refuse.duplicates = FALSE, method = NULL,
  addPP = TRUE)
```

```
validPath(value1, name1 = as.character(substitute(value1)), type,
  method = NULL, addPP = TRUE, extension = NULL, checkFsep = FALSE)
```

Arguments

value1	the value of the (first) argument to be checked
name1	the name of the (first) argument.
validLength	the acceptable length(s) for the argument. If NULL no test is performed.
validValues	the acceptable value(s) for the argument. If NULL no test is performed. Can also be "character" or "character_or_logical".
refuse.NULL	should an error be output if value is NULL.
refuse.duplicates	should an error be output if value contains duplicated values.
method	the name of the function using the argument.
addPP	add ": " after the name of the function in the error message.
validClass	the acceptable classes(s) for the argument.
superClasses	uses the is function instead of class to test the class of the object.
value2	the second value of a second argument whose dimensions should be consistent with the first one
name2	the name of the second argument.
validDimension	the acceptable dimension for the argument. If NULL then name2 is used as a reference.
type	For validDimension: the type of operator used to check the dimensions. For validPath either "dir" or "file" to check whether to path points to an existing directory or file.
min	the minimum acceptable value
max	the maximum acceptable value
refuse.NA	should an error be output if value contains NA.
requiredValues	values that must appear in the argument
refuse.values	values that must not appear in the argument
extension	check that the file has the specified extension (excluding .).
checkFsep	check whether the path ends with a separator.

Value

An invisible TRUE or an error message.

Index