Package: crmPack (via r-universe)

September 2, 2024

Title Object-Oriented Implementation of CRM Designs

Version 2.0.0.9158

Description Implements a wide range of model-based dose escalation designs, ranging from classical and modern continual reassessment methods (CRMs) based on dose-limiting toxicity endpoints to dual-endpoint designs taking into account a biomarker/efficacy outcome. The focus is on Bayesian inference, making it very easy to setup a new design with its own JAGS code. However, it is also possible to implement 3+3 designs for comparison or models with non-Bayesian estimation. The whole package is written in a modular form in the S4 class system, making it very flexible for adaptation to new models, escalation or stopping rules. Further details are presented in Sabanes Bove et al. (2019) <doi:10.18637/jss.v089.i10>.

License GPL (>= 2)

URL https://github.com/openpharma/crmPack

BugReports https://github.com/openpharma/crmPack/issues

Depends ggplot2 (>= 2.0.0), graphics, R (>= 3.5.0)

Imports checkmate (>= 2.2.0), dplyr, futile.logger, GenSA, gridExtra, kableExtra, knitr, lifecycle, magrittr, methods, mvtnorm, parallel, parallelly, rjags, rlang, survival, tibble, tidyselect (>= 1.2.0), tools, utils

Suggests bookdown, broom, covr, data.tree, ggmcmc, quarto (>= 1.4), rmarkdown, stringr, testthat (>= 3.0.0), tidyr, vdiffr, withr

VignetteBuilder knitr, quarto

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyLoad yes

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Collate 'CrmPackClass-class.R' 'CrmPackClass-methods.R'
'Data-validity.R' 'helpers.R' 'Data-class.R' 'helpers_data.R'
'Data-methods.R' 'Rules-validity.R' 'Rules-class.R'
'ModelParams-validity.R' 'ModelParams-class.R'
'Model-validity.R' 'helpers_jags.R' 'Model-class.R'
'Design-validity.R' 'Design-class.R' 'McmcOptions-validity.R'
'McmcOptions-class.R' 'McmcOptions-methods.R'
'Samples-validity.R' 'Samples-class.R' 'logger.R'
'helpers_covr.R' 'mcmc.R' 'Simulations-validity.R'
'Simulations-class.R' 'helpers_broom.R' 'helpers_rules.R'
'Model-methods.R' 'checkmate.R' 'Rules-methods.R'
'Design-methods.R' 'fromQuantiles.R' 'Samples-methods.R'
'Simulations-methods.R' 'crmPack-package.R' 'helpers_design.R'
'helpers_knitr.R' 'helpers_knitr_CohortSize.R'
'helpers_knitr_Design.R' 'helpers_knitr_GeneralData.R'
'helpers_knitr_GeneralModel.R' 'helpers_knitr_Increments.R'
'helpers_knitr_NextBest.R' 'helpers_knitr_SafetyWindow.R'
'helpers_knitr_Stopping.R' 'helpers_model.R'
'helpers_samples.R' 'helpers_simulations.R' 'utils-pipe.R'
'utils.R'

Repository https://openpharma.r-universe.dev

RemoteUrl https://github.com/openpharma/crmpack

RemoteRef HEAD

RemoteSha 75659a9bd9c856ac5ae33ddaed52eb1cef79df59

Contents

`crmPack-package` *Object-oriented implementation of CRM designs*

Description

Object-oriented implementation of CRM designs

References

Sabanes Bove D, Yeung WY, Palermo G, Jaki T (2019). "Model-Based Dose Escalation Designs in R with crmPack." Journal of Statistical Software, 89(10), 1-22. doi:10.18637/jss.v089.i10 (URL: http://doi.org/10.18637/jss.v089.i10).

`.DefaultCohortSize` `CohortSize`

Description

[Stable]

`CohortSize` is a class for cohort sizes.

Usage

```
.DefaultCohortSize()
```

```
.DefaultCohortSize()
```

Note

Typically, end users will not use the `DefaultCohortSize()` function.

Typically, end users will not use the `DefaultCohortSize()` function.

See Also

`CohortSizeRange`, `CohortSizeDLT`, `CohortSizeConst`, `CohortSizeParts`, `CohortSizeMin`, `CohortSizeMin`.

`approximate` *Approximate posterior with (log) normal distribution*

Description

To reproduce the resultant approximate model in the future exactly, include `seed = xxxx` in the call to `approximate`.

Usage

```
approximate(object, model, data, ...)

## S4 method for signature 'Samples'
approximate(
  object,
  model,
  data,
  points = seq(from = min(data@doseGrid), to = max(data@doseGrid), length = 5L),
  refDose = median(points),
  logNormal = FALSE,
  verbose = TRUE,
  create_plot = TRUE,
  ...
)
```

Arguments

object	the Samples object
model	the GeneralModel object
data	the Data object
...	additional arguments (see methods)
points	optional parameter, which gives the dose values at which the approximation should rely on (default: 5 values equally spaced from minimum to maximum of the dose grid)
refDose	the reference dose to be used (default: median of points)
logNormal	use the log-normal prior? (not default) otherwise, the normal prior for the logistic regression coefficients is used
verbose	be verbose (progress statements)? (default)
create_plot	add a ggplot2 object to the return value (default)

Value

a list containing the approximation model and, if requested, a ggplot2 object containing a graphical representation of the fitted model

Functions

- approximate(Samples): Here the ... argument can transport additional arguments for Quantiles2LogisticNormal, e.g. in order to control the approximation quality, etc.

Examples

```
# nolint start

# Create some data
data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  cohort = c(0, 1, 2, 3, 4, 5, 5, 5),
  doseGrid = c(
    0.1, 0.5, 1.5, 3, 6,
    seq(from = 10, to = 80, by = 2)
  )
)

# Initialize a model
model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 56
)

# Get posterior for all model parameters
options <- McmcOptions(
```

```
  burnin = 100,
  step = 2,
  samples = 2000
)
set.seed(94)
samples <- mcmc(data, model, options)

# Approximate the posterior distribution with a bivariate normal
# max.time and maxit are very small only for the purpose of showing the example. They
# should be increased for a real case.
set.seed(94)
approximation <- approximate(
  object = samples,
  model = model,
  data = data,
  logNormal = TRUE,
  control = list(
    threshold.stop = 0.1,
    max.time = 1,
    maxit = 1
  )
)

posterior <- approximation$model

# nolint end
```

`assertions` *Additional Assertions for* `checkmate`

Description

[Experimental]

We provide additional assertion functions that can be used together with the `checkmate` functions. These are described in individual help pages linked below.

Value

Depending on the function prefix.

- `assert_` functions return the object invisibly if successful, and otherwise throw an error message.
- `check_` functions return `TRUE` if successful, otherwise a string with the error message.
- `test_` functions just return `TRUE` or `FALSE`.

See Also

`assert_probabilities()`, `assert_probability()`, `assert_probability_range()`, `assert_length()`.

`biomarker` *Get the Biomarker Levels for a Given Dual-Endpoint Model, Given Dose Levels and Samples*

Description

[Experimental]

Usage

```
biomarker(xLevel, model, samples, ...)

## S4 method for signature 'integer,DualEndpoint,Samples'
biomarker(xLevel, model, samples, ...)
```

Arguments

`xLevel`	(`integer`) the levels for the doses the patients have been given w.r.t dose grid. See `Data` for more details.
`model`	(`DualEndpoint`) the model.
`samples`	(`Samples`) the samples of model's parameters that store the value of biomarker levels for all doses on the dose grid.
`...`	not used.

Details

This function simply returns a specific columns (with the indices equal to `xLevel`) of the biomarker samples matrix, which is included in the the `samples` object.

Value

The biomarker levels.

Functions

- `biomarker(xLevel = integer, model = DualEndpoint, samples = Samples)`:

Examples

```
# Create the data.
my_data <- DataDual(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10, 20, 20, 20, 40, 40, 40, 50, 50, 50),
  y = c(0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1),
  ID = 1:17,
```

```
  cohort = c(1L, 2L, 3L, 4L, 5L, 6L, 6L, 6L, 7L, 7L, 7L, 8L, 8L, 8L, 9L, 9L, 9L),
  w = c(
    0.31, 0.42, 0.59, 0.45, 0.6, 0.7, 0.55, 0.6, 0.52, 0.54,
    0.56, 0.43, 0.41, 0.39, 0.34, 0.38, 0.21
  ),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)

# Initialize the Dual-Endpoint model (in this case RW1).
my_model <- DualEndpointRW(
  mean = c(0, 1),
  cov = matrix(c(1, 0, 0, 1), nrow = 2),
  sigma2betaW = 0.01,
  sigma2W = c(a = 0.1, b = 0.1),
  rho = c(a = 1, b = 1),
  rw1 = TRUE
)

# Set-up some MCMC parameters and generate samples from the posterior.
my_options <- McmcOptions(
  burnin = 100,
  step = 2,
  samples = 500
)
my_samples <- mcmc(my_data, my_model, my_options)

# Obtain the biomarker levels (samples) for the second dose from the dose grid,
# which is 0.5.
biomarker(
  xLevel = 2L,
  model = my_model,
  samples = my_samples
)
```

---

check_equal *Check if All Arguments Are Equal*

Description

[Experimental] Elements of ... must be numeric vectors or scalars.

This function performs an element-by-element comparison of the first object provided in ... with every other object in ... and returns TRUE if all comparisons are equal within a given tolerance and FALSE otherwise.

[Experimental] Elements of ... must be numeric vectors or scalars.

This function performs an element-by-element comparison of the first object provided in ... with every other object in ... and throws an error if they are not.

Usage

```
check_equal(..., tol = sqrt(.Machine$double.eps))

assert_equal(
  ...,
  tol = sqrt(.Machine$double.eps),
  .var.name = vname(x),
  add = NULL
)
```

Arguments

`...`	(`numeric`) vectors to be compared
`tol`	(`numeric`) the maximum difference to be tolerated when judging equality
`.var.name`	[`character(1)`] Name of the checked object to print in assertions. Defaults to the heuristic implemented in `vname`.
`add`	[`AssertCollection`] Collection to store assertion messages. See `AssertCollection`.

Value

`TRUE` if all element-by-element differences are less than `tolerance` in magnitude, `FALSE` otherwise.

`list(...)`, invisibly.

Note

If there are any missing or infinite values in `...`, this function returns `FALSE`, regardless of the values of other elements in `....`

If elements in `...` are not all of the same length, `FALSE` is returned.

If there are any missing or infinite values in `...`, this function throws an error, regardless of the values of other elements in `....`

If elements in `...` are not all of the same length, an error is thrown.

See Also

`assertions` for more details.

`assertions` for more details.

Examples

```
check_equal(1:2, 1:2) # TRUE
check_equal(1:2, 2:3) # "Not all equal"
check_equal(Inf, Inf) # "Not all equal"
check_equal(0.01, 0.02) # "Not all equal"
```

```
check_equal(0.01, 0.02, tol = 0.05) # TRUE
check_equal(1, c(1, 1)) # "Not all equal"
assert_equal(1:2, 1:2) # no error
assert_equal(0.01, 0.02, tol = 0.05) # no error
```

`check_format` *Check that an argument is a valid format specification*

Description

[Stable]

Usage

```
check_format(x, len = NULL, min.len = NULL, max.len = NULL)

assert_format(
  x,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  .var.name = checkmate::vname(x),
  add = NULL
)

test_format(x, len = NULL, min.len = NULL, max.len = NULL)

expect_format(
  x,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  info = NULL,
  label = vname(x)
)
```

Arguments

`x`	[any] Object to check.
`len`	[`integer(1)`] Exact expected length of `x`.
`min.len`	[`integer(1)`] Minimal length of `x`.
`max.len`	[`integer(1)`] Maximal length of `x`.

.var.name	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.
add	[AssertCollection] Collection to store assertion messages. See AssertCollection.
info	[character(1)] Extra information to be included in the message for the testthat reporter. See expect_that.
label	[character(1)] Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value

TRUE if successful, otherwise a string with the error message.

See Also

assertions for more details.

Examples

```
check_format("%5.2f")
```

---

check_length *Check if vectors are of compatible lengths*

Description

[Stable]

Two vectors are of compatible size if and only if:

1. At least one vector has size 1
2. or both vectors are of the same size.

Usage

```
check_length(x, len)

assert_length(x, len, .var.name = checkmate::vname(x), add = NULL)

test_length(x, len)
```

Arguments

x	(any) the first vector, any object for which `length()` function is defined.
len	(count) the length of the second vector.
.var.name	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in `vname`.
add	[AssertCollection] Collection to store assertion messages. See `AssertCollection`.

Value

TRUE if successful, otherwise a string with the error message.

See Also

`assertions` for more details.

Examples

```
check_length(1:5, 1)
check_length(1:5, 6)
check_length(1:5, 5)
check_length(10, 1)
check_length(10, 9)
```

---

check_probabilities *Check if an argument is a probability vector*

Description

[Stable]

Check if every element in a given numerical vector or matrix represents a probability, that is a number within (0, 1) interval, that can optionally be closed at any side.

Usage

```
check_probabilities(
  x,
  bounds_closed = TRUE,
  len = NULL,
  unique = FALSE,
  sorted = FALSE
)
```

```
assert_probabilities(
  x,
  bounds_closed = TRUE,
  len = NULL,
  unique = FALSE,
  sorted = FALSE,
  .var.name = checkmate::vname(x),
  add = NULL
)

test_probabilities(
  x,
  bounds_closed = TRUE,
  len = NULL,
  unique = FALSE,
  sorted = FALSE
)

expect_probabilities(
  x,
  bounds_closed = TRUE,
  len = NULL,
  unique = FALSE,
  sorted = FALSE,
  info = NULL,
  label = vname(x)
)
```

Arguments

x	(numeric) vector or matrix with numerical values to check.
bounds_closed	(logical) should bounds be closed? This can be a scalar or vector of length two. If it is a scalar, then its value applies equally to lower bound 0 and upper bound 1. If this is a vector with two flags, the first flag corresponds to the lower bound 0 only, and the second to the upper bound 1 only.
len	[integer(1)] Exact expected length of x.
unique	[logical(1)] Must all values be unique? Default is FALSE.
sorted	[logical(1)] Elements must be sorted in ascending order. Missing values are ignored.
.var.name	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.
add	[AssertCollection] Collection to store assertion messages. See AssertCollection.

info	[character(1)] Extra information to be included in the message for the testthat reporter. See expect_that.
label	[character(1)] Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value

TRUE if successful, otherwise a string with the error message.

Note

If there are any missing or non-finite values in x, this function returns FALSE, regardless of the values of other elements in x.

See Also

assertions for more details.

Examples

```
x <- c(0, 0.2, 0.1, 0.3, 1)
check_probabilities(x)
check_probabilities(x, bounds_closed = FALSE)
check_probabilities(x, bounds_closed = c(FALSE, TRUE))
```

---

check_probability *Check if an argument is a single probability value*

Description

[Stable]

Check if a given value represents a probability, that is a number within (0, 1) interval, that can optionally be closed at any side.

Usage

```
check_probability(x, bounds_closed = TRUE)

assert_probability(
  x,
  bounds_closed = TRUE,
  .var.name = checkmate::vname(x),
  add = NULL
)
```

```
test_probability(x, bounds_closed = TRUE)

expect_probability(x, bounds_closed = TRUE, info = NULL, label = vname(x))
```

Arguments

x	(number) a single value to check.
bounds_closed	(logical) should bounds be closed? This can be a scalar or vector of length two. If it is a scalar, then its value applies equally to lower bound 0 and upper bound 1. If this is a vector with two flags, the first flag corresponds to the lower bound 0 only, and the second to the upper bound 1 only.
.var.name	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.
add	[AssertCollection] Collection to store assertion messages. See AssertCollection.
info	[character(1)] Extra information to be included in the message for the testthat reporter. See expect_that.
label	[character(1)] Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value

TRUE if successful, otherwise a string with the error message.

See Also

assertions for more details.

Examples

```
check_probability(0.5)
check_probability(0, bounds_closed = FALSE)
check_probability(0, bounds_closed = c(FALSE, TRUE))
```

---

check_probability_range

Check if an argument is a probability range

Description

[Stable]

Check if a given numerical interval represents a probability range, that is a sub-interval of (0, 1) interval, that can optionally be closed at any side.

Usage

```
check_probability_range(x, bounds_closed = TRUE)

assert_probability_range(
  x,
  bounds_closed = TRUE,
  .var.name = checkmate::vname(x),
  add = NULL
)

test_probability_range(x, bounds_closed = TRUE)

expect_probability_range(
  x,
  bounds_closed = TRUE,
  info = NULL,
  label = vname(x)
)
```

Arguments

x	(number) an interval to check.
bounds_closed	(logical) should bounds be closed? This can be a scalar or vector of length two. If it is a scalar, then its value applies equally to lower bound 0 and upper bound 1. If this is a vector with two flags, the first flag corresponds to the lower bound 0 only, and the second to the upper bound 1 only.
.var.name	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.
add	[AssertCollection] Collection to store assertion messages. See AssertCollection.
info	[character(1)] Extra information to be included in the message for the testthat reporter. See expect_that.
label	[character(1)] Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value

TRUE if successful, otherwise a string with the error message.

See Also

assertions for more details.

Examples

```
x <- c(0, 0.2)
check_probability_range(x)
check_probability_range(rev(x))
check_probability_range(x, bounds_closed = FALSE)
check_probability_range(x, bounds_closed = c(FALSE, TRUE))
```

---

check_range *Check that an argument is a numerical range*

Description

[Stable]

An argument x is a numerical range if and only if (all conditions must be met):

1. Is an object of type: integer or double.
2. Is a vector or length two such that the value of the first number is not less than the second number. Equalness is allowed if and only if unique flag is set to TRUE.
3. Lower bound of the interval is greater than or equal to lower and upper bound of the interval is less than or equal to upper.
4. It contains only finite (given that finite is TRUE) and non-missing values.

Usage

```
check_range(x, lower = -Inf, upper = Inf, finite = FALSE, unique = TRUE)

assert_range(
  x,
  lower = -Inf,
  upper = Inf,
  finite = FALSE,
  unique = TRUE,
  .var.name = checkmate::vname(x),
  add = NULL
)

test_range(x, lower = -Inf, upper = Inf, finite = FALSE, unique = TRUE)
```

```
expect_range(
  x,
  lower = -Inf,
  upper = Inf,
  finite = FALSE,
  unique = TRUE,
  info = NULL,
  label = vname(x)
)
```

Arguments

x	[any] Object to check.
lower	[numeric(1)] Lower value all elements of x must be greater than or equal to.
upper	[numeric(1)] Upper value all elements of x must be lower than or equal to.
finite	[logical(1)] Check for only finite values? Default is FALSE.
unique	[logical(1)] Must all values be unique? Default is FALSE.
.var.name	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.
add	[AssertCollection] Collection to store assertion messages. See AssertCollection.
info	[character(1)] Extra information to be included in the message for the testthat reporter. See expect_that.
label	[character(1)] Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value

TRUE if successful, otherwise a string with the error message.

See Also

assertions for more details.

Examples

```
check_range(c(1, 5))
check_range(c(-5, 1))
check_range(c(4, 1))
```

```
check_range(c(1, 1))
check_range(c(1, 1), unique = FALSE)
check_range(1:3)
```

`CohortSizeConst-class` `CohortSizeConst`

Description

[Stable]

`CohortSizeConst` is the class for fixed and constant size of cohort.

Usage

```
CohortSizeConst(size)

.DefaultCohortSizeConst()
```

Arguments

`size`	(`number`) see slot definition.

Slots

`size` (`integer`)
: cohort size.

Note

Typically, end users will not use the `.DefaultCohortSizeConst()` function.

Examples

```
# Cohort of size 3, constant along the study.
my_size <- CohortSizeConst(size = 3)
```

CohortSizeDLT-class CohortSizeDLT

Description

[Stable]

CohortSizeDLT is the class for cohort size based on number of DLTs.

Usage

```
CohortSizeDLT(intervals, cohort_size)

.DefaultCohortSizeDLT()
```

Arguments

intervals	(numeric) see slot definition.
cohort_size	(numeric) see slot definition.

Slots

`intervals` (integer)
: a vector with the left bounds of the relevant DLT intervals.

`cohort_size` (integer)
: a vector with the cohort sizes corresponding to the elements of `intervals`.

Note

Typically, end users will not use the `.DefaultCohortSizeDLT()` function.

Examples

```
# Rule for having cohort of size 1 until no DLT is observed and having cohort
# of size 3 as soon as 1 DLT is observed.
my_size <- CohortSizeDLT(intervals = c(0, 1), cohort_size = c(1, 3))
```

CohortSizeMax-class CohortSizeMax

Description

[Stable]

CohortSizeMax is the class for cohort size that is based on maximum of multiple cohort size rules. The cohort_sizes slot stores a set of cohort size rules, which are again the objects of class CohortSize. The maximum of these individual cohort sizes is taken to give the final cohort size.

Usage

```
.DefaultCohortSizeMax()

CohortSizeMax(cohort_sizes)
```

Arguments

cohort_sizes	(list) see slot definition.

Slots

cohort_sizes (list)
: a list of cohort size rules, i.e. objects of class CohortSize.

Note

Typically, end users will not use the .DefaultCohortSizeMax() function.

Examples

```
# Rule for cohort of size 1 for doses <30 and cohort of size 3 for doses >=30.
my_size1 <- CohortSizeRange(intervals = c(0, 10), cohort_size = c(1, 3))

# Rule for cohort of size 1 until no DLT were observed and cohort of size 3
# as soon as 1 DLT is observed.
my_size2 <- CohortSizeDLT(intervals = c(0, 1), cohort_size = c(1, 3))

# Cohort size rules of class 'CohortSizeMax' which will then be combined with
# the 'max' operation.
mySize <- CohortSizeMax(cohort_sizes = list(my_size1, my_size2))
```

CohortSizeMin-class CohortSizeMin

Description

[Stable]

CohortSizeMin is the class for cohort size that is based on minimum of multiple cohort size rules. The cohort_sizes slot stores a set of cohort size rules, which are again the objects of class CohortSize. The minimum of these individual cohort sizes is taken to give the final cohort size.

Usage

```
CohortSizeMin(cohort_sizes)

.DefaultCohortSizeMin()
```

Arguments

cohort_sizes	(list) see slot definition.

Slots

cohort_sizes (list)
: a list of cohort size rules, i.e. objects of class CohortSize.

Note

Typically, end users will not use the .DefaultCohortSizeMin() function.

Examples

```
# Rule for cohort of size 1 for doses <30 and cohort of size 3 for doses >=30.
my_size1 <- CohortSizeRange(intervals = c(0, 10), cohort_size = c(1, 3))

# Rule for cohort of size 1 until no DLT were observed and cohort of size 3
# as soon as 1 DLT is observed.
my_size2 <- CohortSizeDLT(intervals = c(0, 1), cohort_size = c(1, 3))

# Cohort size rules of class 'CohortSizeMin' which will then be combined with
# the 'min' operation.
my_size <- CohortSizeMin(cohort_sizes = list(my_size1, my_size2))
```

`CohortSizeOrdinal-class` `CohortSizeOrdinal`

Description

[Experimental]

`CohortSizeOrdinal` is the class for cohort size for an ordinal CRM trial.

Usage

```
CohortSizeOrdinal(grade, rule)

.DefaultCohortSizeOrdinal()
```

Arguments

`grade`	(`integer`) see slot definition.
`rule`	(`CohortSize`) see slot definition.

Slots

`grade` (`integer`)
: the grade at which the rule should be applied

`rule` (`CohortSize`)
: the `CohortSize` rule to apply.

Note

Typically, end users will not use the `.DefaultCohortSizeOrdinal()` function.

Examples

```
CohortSizeOrdinal(
  grade = 1L,
  rule = CohortSizeRange(intervals = c(0, 30), cohort_size = c(1L, 3L))
)
```

CohortSizeParts-class CohortSizeParts

Description

[Stable]

CohortSizeParts is the class for cohort size that changes for the second part of the dose escalation. It works only in conjunction with DataParts objects.

Usage

```
CohortSizeParts(cohort_sizes)

.DefaultCohortSizeParts()
```

Arguments

cohort_sizes	(numeric) see slot definition.

Slots

cohort_sizes (integer)
: a vector of length two with two sizes, one for part 1, and one for part 2 respectively.

Note

Typically, end users will not use the .DefaultCohortSizeParts() function.

Examples

```
# Part 1 cohort size = 1, Part 2 cohort size = 3.
my_size <- CohortSizeParts(cohort_sizes = c(1, 3))
```

CohortSizeRange-class CohortSizeRange

Description

[Stable]

CohortSizeRange is the class for cohort size based on dose range.

Usage

```
CohortSizeRange(intervals, cohort_size)

.DefaultCohortSizeRange()
```

Arguments

intervals	(numeric) see slot definition.
cohort_size	(numeric) see slot definition.

Slots

intervals (numeric)
: a vector with the left bounds of the relevant dose intervals.

cohort_size (integer)
: an integer vector with the cohort sizes corresponding to the elements of intervals.

Note

Typically, end users will not use the .DefaultCohortSizeRange() function.

Examples

```
# Example for the rule having cohort of size 1 for doses <30
# and having cohort of size 3 for doses >=30.

my_size <- CohortSizeRange(intervals = c(0, 30), cohort_size = c(1, 3))
```

CrmPackClass-class CrmPackClass

Description

[Experimental]

CrmPackClass is a virtual class, from which all other crmPack classes inherit.

---

crmPackExample *Open the example pdf for crmPack*

Description

Calling this helper function should open the example.pdf document, residing in the doc subfolder of the package installation directory.

Usage

```
crmPackExample()
```

Value

nothing

Author(s)

Daniel Sabanes Bove <sabanesd@roche.com>

---

crmPackHelp *Open the browser with help pages for crmPack*

Description

This convenience function opens your browser with the help pages for crmPack.

Usage

```
crmPackHelp()
```

Value

nothing

Author(s)

Daniel Sabanes Bove <sabanesd@roche.com>

DADesign-class DADesign

Description

[Stable]

This class has special requirements for the `model` and `data` slots in comparison to the parent class `Design`:

Usage

```
DADesign(model, data, safetyWindow, ...)

.DefaultDADesign()
```

Arguments

`model`	(`GeneralModel`) see slot definition.
`data`	(`DataDA`) see slot definition.
`safetyWindow`	(`SafetyWindow`) see slot definition.
`...`	Arguments passed on to `Design` `stopping` (`Stopping`) see slot definition. `increments` (`Increments`) see slot definition. `pl_cohort_size` (`CohortSize`) see slot definition.

Details

The `safetyWindow` slot should be an instance of the `SafetyWindow` class. It can be customized to specify the duration of the safety window for your trial. The safety window represents the time period required to observe toxicity data from the ongoing cohort before opening the next cohort. Note that even after opening the next cohort, further toxicity data will be collected and analyzed to make dose escalation decisions.

To specify a constant safety window, use the `SafetyWindowConst` constructor. For example:

```
mysafetywindow <- SafetyWindowConst(c(6, 2), 10, 20)
```

Slots

`model` (`GeneralModel`)
: the model to use, see in particular `DALogisticLogNormal` and `TITELogisticLogNormal` which make use of the time-to-DLT data.

`data` (`DataDA`)
: what is the dose grid, any previous data, etc.

`safetyWindow` (`SafetyWindow`)
: the safety window to apply between cohorts.

Note

Typically, end users will not use the `.DefaultDADesign()` function.

See Also

`SafetyWindowConst` for creating a constant safety window.

Examples

```
empty_data <- DataDA(doseGrid = c(
  0.1, 0.5, 1, 1.5, 3, 6,
  seq(from = 10, to = 80, by = 2)
), Tmax = 60)

npiece <- 10
t_max <- 60

lambda_prior <- function(k) {
  npiece / (t_max * (npiece - k + 0.5))
}

model <- DALogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 56,
  npiece = npiece,
  l = as.numeric(t(apply(as.matrix(c(1:npiece), 1, npiece), 2, lambda_prior))),
  c_par = 2
)

my_increments <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

my_next_best <- NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

my_size1 <- CohortSizeRange(
  intervals = c(0, 30),
  cohort_size = c(1, 3)
)

my_size2 <- CohortSizeDLT(
  intervals = c(0, 1),
  cohort_size = c(1, 3)
)

my_size <- maxSize(my_size1, my_size2)

my_stopping1 <- StoppingTargetProb(
  target = c(0.2, 0.35),
  prob = 0.5
)

my_stopping2 <- StoppingMinPatients(nPatients = 50)
```

```
my_stopping <- (my_stopping1 | my_stopping2)

my_safety_window <- SafetyWindowConst(c(6, 2), 7, 7)

design <- DADesign(
  model = model,
  increments = my_increments,
  nextBest = my_next_best,
  stopping = my_stopping,
  cohort_size = my_size,
  data = empty_data,
  safetyWindow = my_safety_window,
  startingDose = 3
)
```

DALogisticLogNormal-class

DALogisticLogNormal

Description

[Stable]

DALogisticLogNormal is the class for the logistic model with bivariate (log) normal prior and data augmentation. This class inherits from the LogisticLogNormal class.

Usage

```
DALogisticLogNormal(npiece = 3, l, c_par = 2, cond_pem = TRUE, ...)

.DefaultDALogisticLogNormal()
```

Arguments

npiece	(number) the number of pieces in the PEM.
l	(numeric) a vector used in the lambda prior.
c_par	(numeric) a parameter used in the lambda prior; according to Liu's paper, c_par = 2 is recommended.
cond_pem	(flag) is a conditional piecewise-exponential model used? (default). Otherwise an unconditional model is used.
...	Arguments passed on to LogisticLogNormal mean (numeric) the prior mean vector.

`cov` (`matrix`)
: the prior covariance matrix. The precision matrix `prec` is internally calculated as an inverse of `cov`.

`ref_dose` (`number`)
: the reference dose $x*$ (strictly positive number).

Slots

`npiece` (`number`)
: the number of pieces in the `PEM`.

`l` (`numeric`)
: a vector used in the lambda prior.

`c_par` (`numeric`)
: a parameter used in the lambda prior; according to Liu's paper, `c_par = 2` is recommended.

`cond_pem` (`flag`)
: is a conditional piecewise-exponential model used? (default). Otherwise an unconditional model is used.

Note

We still need to include here formula for the lambda prior.

Typically, end users will not use the `.DefaultDALogisticLogNormal()` function.

See Also

`ModelLogNormal`, `LogisticNormal`, `LogisticLogNormal`.

Examples

```
npiece <- 10
Tmax <- 60 # nolintr

lambda_prior <- function(k) {
  npiece / (Tmax * (npiece - k + 0.5))
}

model <- DALogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 56,
  npiece = npiece,
  l = as.numeric(t(apply(as.matrix(c(1:npiece), 1, npiece), 2, lambda_prior))),
  c_par = 2
)
```

dapply *Apply a Function to Subsets of Data Frame.*

Description

[Experimental]

dapply splits the data df into the subsets defined by f, and applies function FUN to each of the subset. All the results are row-binded and returned as data.frame object.

Usage

```
dapply(df, f, FUN, ...)
```

Arguments

df	(data frame) data set to be divided into groups.
f	(factor or formula or list) a factor in the sense that as.factor(f) defines the grouping, or a list of such factors in which case their interaction is used for the grouping. f can also be a formula of the form ~ g1 + ... + gk to split by the interaction of the variables g1, ..., gk. This parameter is passed directly into split() function.
FUN	(function) the function to be applied to each subset of df defined by f.
...	parameters passed to lapply(), which is used when applying a function FUN over groups defined by f.

Value

The data.frame object with results from FUN.

Examples

```
df <- data.frame(
  dose = c(0.1, 6, 6, 5, 0.1, 5, 6, 6),
  cohort = c("B", "B", "B", "A", "A", "A", "B", "B")
)

dapply(
  df,
  f = ~cohort,
  FUN = function(coh) {
    data.frame(my_cohort = coh$cohort[1], my_max = max(coh$dose))
  }
)

dapply(
```

```
  df,
  f = ~cohort,
  FUN = function(coh) {
    coh$dose <- sort(coh$dose, decreasing = TRUE)
    coh
  }
)
```

DASimulations *Initialization function for* DASimulations

Description

Initialization function for DASimulations

Usage

```
DASimulations(trialduration, ...)
```

Arguments

trialduration	see DASimulations
...	additional parameters from Simulations

Value

the DASimulations object

DASimulations-class *Class for the simulations output from DA based designs*

Description

This class captures the trial simulations from DA based designs. In comparison to the parent class Simulations, it contains additional slots to capture the time to DLT fits, additional parameters and the trial duration.

Usage

```
.DefaultDASimulations()
```

Slots

trialduration the vector of trial duration values for all simulations.

Note

Typically, end users will not use the .DASimulations() function. This function has a noticeable execution time.

Data-class Data

Description

[Stable]

Data is a class for the data input. It inherits from GeneralData.

Usage

```
Data(
  x = numeric(),
  y = integer(),
  ID = integer(),
  cohort = integer(),
  doseGrid = numeric(),
  placebo = FALSE,
  ...
)

.DefaultData()
```

Arguments

x	(numeric) the doses for the patients.
y	(integer) the vector of toxicity events (0 or 1). You can also supply numeric vectors, but these will then be converted to integer internally.
ID	(integer) unique patient IDs. You can also supply numeric vectors, but these will then be converted to integer internally.
cohort	(integer) the cohort (non-negative sorted) indices. You can also supply numeric vectors, but these will then be converted to integer internally.
doseGrid	(numeric) all possible doses.
placebo	(flag) if TRUE the first dose level in the doseGrid is considered as placebo.
...	not used.

Details

The cohort can be missing if and only if placebo is equal to FALSE.

Slots

`x (numeric)`
: the doses for the patients.

`y (integer)`
: the vector of toxicity events (0 or 1 integers).

`doseGrid (numeric)`
: the vector of all possible doses (sorted), i.e. the dose grid.

`nGrid (integer)`
: number of gridpoints.

`xLevel (integer)`
: the levels for the doses the patients have been given, w.r.t `doseGrid`.

`placebo (logical)`
: if `TRUE` the first dose level in the `doseGrid`is considered as PLACEBO.

Note

`ID` and `cohort` can be missing. Then a message will be issued and the variables will be filled with default IDs and best guesses cohort, i.e. a sorted (in ascending order) sequence of values from `{1, 2, ...}`.

Typically, end users will not use the `.DefaultData()` function.

Examples

```
my_data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  ID = as.integer(1:8),
  cohort = as.integer(c(1, 2, 3, 4, 5, 6, 6, 6)),
  doseGrid = c(
    0.1, 0.5, 1.5, 3, 6,
    seq(from = 10, to = 80, by = 2)
  )
)
my_data
```

`DataDA-class` `DataDA`

Description

[Stable]

`DataDA` is a class for the time-to-DLT augmented data. It inherits from `Data` and it contains additional DLT free survival times.

Usage

```
DataDA(
  u = numeric(),
  t0 = numeric(length(u)),
  Tmax = 0 + .Machine$double.xmin,
  ...
)

.DefaultDataDA()
```

Arguments

u	(numeric) the continuous vector of DLT free survival times.
t0	(numeric) time of initial dosing for each patient. Non-negative values sorted in ascending order. Default to vector of 0s of length equal to length of u.
Tmax	(number) the DLT observation period.
...	parameters passed to Data().

Slots

u (numeric)
: the continuous vector of DLT free survival times.

t0 (numeric)
: time of initial dosing for each patient. Non-negative values sorted in ascending order.

Tmax (number)
: the DLT observation period.

Note

survival time here refers to the time period for which the subject did not experience any DLT, and is not referring to deaths.

Typically, end users will not use the .DefaultDataDA() function.

Examples

```
my_data <- DataDA(
  u = c(42, 30, 15, 5, 20, 25, 30, 60),
  t0 = c(0, 15, 30, 40, 55, 70, 75, 85),
  Tmax = 60,
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 1, 1, 0, 0, 1, 0),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)

# Set up an empty data set.
```

```
empty_data <- DataDA(
  doseGrid = c(0.1, 0.5, 1, 1.5, 3, 6, seq(from = 10, to = 80, by = 2)),
  Tmax = 60
)
empty_data
```

DataDual-class DataDual

Description

[Stable]

DataDual is a class for the dual endpoint data. It inherits from Data and it contains additional biomarker information.

Usage

```
DataDual(w = numeric(), ...)

.DefaultDataDual()
```

Arguments

w	(numeric) the continuous vector of biomarker values.
...	parameters passed to Data().

Slots

w (numeric)
: the continuous vector of biomarker values.

Note

Typically, end users will not use the .DefaultDataDual() function.

Examples

```
my_data <- DataDual(
  w = rnorm(8),
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  doseGrid = c(
    0.1, 0.5, 1.5, 3, 6,
    seq(from = 10, to = 80, by = 2)
  )
)
my_data
```

DataGrouped-class DataGrouped

Description

[Stable]

DataGrouped is a class for a two groups dose escalation data set, comprised of a monotherapy (mono) and a combination therapy (combo) arm. It inherits from Data and it contains the additional group information.

Usage

```
DataGrouped(group = character(), ...)

.DefaultDataGrouped()
```

Arguments

group	(factor or character) whether mono or combo was used. If character then will be coerced to factor with the correct levels internally.
...	parameters passed to Data().

Slots

group (factor)
: whether mono or combo was used.

Note

Typically, end users will not use the .DefaultDataGrouped() function.

Examples

```
my_data <- DataGrouped(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 1, 1, 0, 0, 1, 0),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2)),
  group = c("mono", "mono", "mono", "mono", "mono", "mono", "combo", "combo")
)

# Set up an empty data set.
empty_data <- DataGrouped(
  doseGrid = c(0.1, 0.5, 1, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)
empty_data
```

`DataMixture-class` `DataMixture`

Description

[Stable]

`DataMixture` is a class for the data with mixture sharing. It inherits from `Data` and it contains additional information on the mixture sharing.

Usage

```
DataMixture(xshare = numeric(), yshare = integer(), ...)

.DefaultDataMixture()
```

Arguments

`xshare`	(`numeric`) the doses for the share patients.
`yshare`	(`integer`) the vector of toxicity events (0 or 1) for the share patients. You can also supply `numeric` vectors, but these will then be converted to `integer` internally.
`...`	parameters passed to `Data()`.

Slots

`xshare` (`numeric`)
: the doses for the share patients.

`yshare` (`integer`)
: the vector of toxicity events (0 or 1) for the share patients.

`nObsshare` (`count`)
: number of share patients.

Note

Typically, end users will not use the `.DefaultDataMixture()` function.

Examples

```
my_data <- DataMixture(
  xshare = c(12, 14, 16, 18.0),
  yshare = c(0L, 1L, 1L, 1L),
  nObsshare = 4L,
  x = c(0.1, 0.5, 1.5),
  y = c(0, 0, 0),
  ID = 1:3,
  cohort = 1:3,
```

```
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)
my_data
```

DataOrdinal-class DataOrdinal

Description

[Experimental]

DataOrdinal is a class for ordinal toxicity data. It inherits from GeneralData and it describes toxicity responses on an ordinal rather than binary scale.

Usage

```
DataOrdinal(
  x = numeric(),
  y = integer(),
  ID = integer(),
  cohort = integer(),
  doseGrid = numeric(),
  placebo = FALSE,
  yCategories = c(`No DLT` = 0L, DLT = 1L),
  ...
)

.DefaultDataOrdinal()
```

Arguments

x	(numeric) the doses for the patients.
y	(integer) the vector of toxicity events (0 or 1). You can also supply numeric vectors, but these will then be converted to integer internally.
ID	(integer) unique patient IDs. You can also supply numeric vectors, but these will then be converted to integer internally.
cohort	(integer) the cohort (non-negative sorted) indices. You can also supply numeric vectors, but these will then be converted to integer internally.
doseGrid	(numeric) all possible doses.
placebo	(flag) if TRUE the first dose level in the doseGrid is considered as placebo.

yCategories	(named integer) the names and codes for the toxicity categories used in the data. Category labels are taken from the names of the vector. The names of the vector must be unique and its values must be sorted and take the values 0, 1, 2, ...
...	not used.

Details

The cohort can be missing if and only if placebo is equal to FALSE.

Note

This class has been implemented as a sibling of the existing Data class (rather than as a parent or child) to minimise the risk of unintended side effects on existing classes and methods.

The default setting for the yCategories slot replicates the behaviour of the existing Data class.

Typically, end users will not use the .DefaultDataOrdinal() function.

Examples

```
DataOrdinal(
  x = c(10, 20, 30, 40, 50, 50, 50, 60, 60, 60),
  y = as.integer(c(0, 0, 0, 0, 0, 1, 0, 0, 1, 2)),
  ID = 1L:10L,
  cohort = as.integer(c(1:4, 5, 5, 5, 6, 6, 6)),
  doseGrid = c(seq(from = 10, to = 100, by = 10)),
  yCategories = c("No tox" = 0L, "Sub-tox AE" = 1L, "DLT" = 2L),
  placebo = FALSE
)
```

DataParts-class DataParts

Description

[Stable]

DataParts is a class for the data with two study parts. It inherits from Data and it contains additional information on the two study parts.

Usage

```
DataParts(part = integer(), nextPart = 1L, part1Ladder = numeric(), ...)

.DefaultDataParts()
```

Arguments

part	(integer) which part does each of the patients belong to?
nextPart	(count) what is the part for the next cohort (1 or 2)?
part1Ladder	(numeric) what is the escalation ladder for part 1? This shall be an ordered subset of the doseGrid.
...	parameters passed to Data().

Slots

part (integer)
: which part does each of the patients belong to?

nextPart (count)
: what is the part for the next cohort (1 or 2)?

part1Ladder (numeric)
: what is the escalation ladder for part 1? This shall be an ordered subset of the doseGrid.

Note

Typically, end users will not use the .DefaultDataParts() function.

Examples

```
my_data <- DataParts(
  x = c(0.1, 0.5, 1.5),
  y = c(0, 0, 0),
  ID = 1:3,
  cohort = 1:3,
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2)),
  part = c(1L, 1L, 1L),
  nextPart = 1L,
  part1Ladder = c(0.1, 0.5, 1.5, 3, 6, 10)
)
my_data
```

Design-class Design

Description

[Stable]

Design is the class for rule-based designs. The difference between this class and its parent RuleDesign class is that Design class contains additional model, stopping and increments slots.

Usage

```
Design(model, stopping, increments, pl_cohort_size = CohortSizeConst(0L), ...)

.DefaultDesign()
```

Arguments

model	(GeneralModel) see slot definition.
stopping	(Stopping) see slot definition.
increments	(Increments) see slot definition.
pl_cohort_size	(CohortSize) see slot definition.
...	Arguments passed on to RuleDesign nextBest (NextBest) see slot definition. cohort_size (CohortSize) see slot definition. data (Data) see slot definition. startingDose (number) see slot definition.

Slots

model (GeneralModel)
: the model to be used.

stopping (Stopping)
: stopping rule(s) for the trial.

increments (Increments)
: how to control increments between dose levels.

pl_cohort_size (CohortSize)
: rules for the cohort sizes for placebo, if any planned (defaults to constant 0 placebo patients).

Note

Typically, end users will not use the .DefaultDesign() function.

Examples

```
empty_data <- Data(doseGrid = c(1, 3, 5, 10, 15, 20, 25, 40, 50, 80, 100))

# Initialize the CRM model.
my_model <- LogisticLogNormal(
```

```
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 56
)

# Choose the rule for selecting the next dose.
my_next_best <- NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Choose the rule for the cohort-size.
my_size1 <- CohortSizeRange(
  intervals = c(0, 30),
  cohort_size = c(1, 3)
)
my_size2 <- CohortSizeDLT(
  intervals = c(0, 1),
  cohort_size = c(1, 3)
)
my_size <- maxSize(my_size1, my_size2)

# Choose the rule for stopping.
my_stopping1 <- StoppingMinCohorts(nCohorts = 3)
my_stopping2 <- StoppingTargetProb(
  target = c(0.2, 0.35),
  prob = 0.5
)
my_stopping3 <- StoppingMinPatients(nPatients = 20)
my_stopping <- (my_stopping1 & my_stopping2) | my_stopping3

# Choose the rule for dose increments.
my_increments <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

# Initialize the design.
design <- Design(
  model = my_model,
  nextBest = my_next_best,
  stopping = my_stopping,
  increments = my_increments,
  cohort_size = my_size,
  data = empty_data,
  startingDose = 3
)
```

DesignGrouped-class DesignGrouped

Description

[Experimental]

DesignGrouped combines two Design objects: one for the mono and one for the combo arm of a joint dose escalation design.

Usage

```
DesignGrouped(
  model,
  mono,
  combo = mono,
  first_cohort_mono_only = TRUE,
  same_dose_for_all = !same_dose_for_start,
  same_dose_for_start = FALSE,
  stop_mono_with_combo = FALSE,
  ...
)
```

Arguments

model	(LogisticLogNormalGrouped) see slot definition.
mono	(Design) see slot definition.
combo	(Design) see slot definition.
first_cohort_mono_only	(flag) see slot definition.
same_dose_for_all	(flag) see slot definition.
same_dose_for_start	(flag) see slot definition.
stop_mono_with_combo	(flag) whether the mono arm should be stopped when the combo arm is stopped (this makes sense when the only real trial objective is the recommended combo dose).
...	not used.

Details

- Note that the model slots inside the mono and combo parameters are ignored (because we don't fit separate regression models for the mono and combo arms). Instead, the model parameter is used to fit a joint regression model for the mono and combo arms together.

- same_dose_for_start = TRUE is useful as an option when we want to use same_dose_for_all = FALSE combined with first_cohort_mono_only = TRUE. This will allow to randomize patients to the mono and combo arms at the same dose as long as the selected dose for the cohorts stay the same. This can therefore further mitigate bias as long as possible between the mono and combo arms.

Slots

model (LogisticLogNormalGrouped)
: the model to be used, currently only one class is allowed.

mono (Design)
: defines the dose escalation rules for the mono arm, see details.

combo (Design)
: defines the dose escalation rules for the combo arm, see details.

first_cohort_mono_only (flag)
: whether first test one mono agent cohort, and then once its DLT data has been collected, we proceed from the second cohort onwards with concurrent mono and combo cohorts.

same_dose_for_all (flag)
: whether the lower dose of the separately determined mono and combo doses should be used as the next dose for both mono and combo in all cohorts.

same_dose_for_start (flag)
: indicates whether, when mono and combo are used in the same cohort for the first time, the same dose should be used for both. Note that this is different from same_dose_for_all which will always force them to be the same. If same_dose_for_all = TRUE, this is therefore ignored. See Details.

Note

Typically, end-users will not use the .DefaultDesignGrouped() function.

Examples

```
empty_data <- Data(doseGrid = c(1, 3, 5, 10, 15, 20, 25, 40, 50, 80, 100))

# Initialize the joint model.
my_model <- LogisticLogNormalGrouped(
  mean = c(-0.85, 0, 1, 0),
  cov = diag(1, 4),
  ref_dose = 56
)

# Choose the rule for selecting the next dose.
my_next_best <- NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Choose the rule for the cohort-size.
```

```
my_size1 <- CohortSizeRange(
  intervals = c(0, 30),
  cohort_size = c(1, 3)
)
my_size2 <- CohortSizeDLT(
  intervals = c(0, 1),
  cohort_size = c(1, 3)
)
my_size <- maxSize(my_size1, my_size2)

# Choose the rule for stopping.
my_stopping1 <- StoppingMinCohorts(nCohorts = 3)
my_stopping2 <- StoppingTargetProb(
  target = c(0.2, 0.35),
  prob = 0.5
)
my_stopping3 <- StoppingMinPatients(nPatients = 20)
my_stopping <- (my_stopping1 & my_stopping2) | my_stopping3

# Choose the rule for dose increments.
my_increments <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

# Rules to be used for both arms.
one_arm <- Design(
  model = .DefaultModelLogNormal(), # Ignored.
  nextBest = my_next_best,
  stopping = my_stopping,
  increments = my_increments,
  cohort_size = my_size,
  data = empty_data,
  startingDose = 3
)

# Initialize the design.
design <- DesignGrouped(
  model = my_model,
  mono = one_arm
)

# Alternative options: Here e.g.
# - use both mono in first cohort and afterwards have mono and combo in parallel,
# - in general allow different dose levels for the cohorts,
# - but for the start (i.e. second cohort) have the same dose for mono and combo.
# - Stop mono arm too, when combo arm is stopped.

design2 <- DesignGrouped(
  model = my_model,
  mono = one_arm,
  first_cohort_mono_only = TRUE,
  same_dose_for_all = FALSE,
```

```
  same_dose_for_start = TRUE,
  stop_mono_with_combo = TRUE
)
```

DesignOrdinal-class DesignOrdinal

Description

[Experimental]

DesignOrdinal is the class for rule-based ordinal designs. The difference between this class and its parent RuleDesignOrdinal class is that the DesignOrdinal class contains additional model, stopping, increments and pl_cohort_size slots.

Usage

```
DesignOrdinal(
  model,
  stopping,
  increments,
  pl_cohort_size = CohortSizeOrdinal(1L, CohortSizeConst(0L)),
  ...
)

.DefaultDesignOrdinal()
```

Arguments

model	(LogisticLogNormalOrdinal) see slot definition.
stopping	(StoppingOrdinal) see slot definition.
increments	(IncrementsOrdinal) see slot definition.
pl_cohort_size	(CohortSizeOrdinal) see slot definition.
...	Arguments passed on to RuleDesignOrdinal next_best (NextBestOrdinal) see slot definition. cohort_size (CohortSizeOrdinal) see slot definition. data (DataOrdinal) see slot definition. starting_dose (number) see slot definition.

Slots

`model` (`LogisticLogNormalOrdinal`)
: the model to be used.

`stopping` (`StoppingOrdinal`)
: stopping rule(s) for the trial.

`increments` (`IncrementsOrdinal`)
: how to control increments between dose levels.

`pl_cohort_size` (`CohortSizeOrdinal`)
: rules for the cohort sizes for placebo, if any planned (defaults to constant 0 placebo patients).

Note

Typically, end users will not use the `.DefaultDesignOrdinal()` function.

Examples

```
my_size1 <- CohortSizeRange(
  intervals = c(0, 30),
  cohort_size = c(1, 3)
)
my_size2 <- CohortSizeDLT(
  intervals = c(0, 1),
  cohort_size = c(1, 3)
)
my_size <- CohortSizeOrdinal(1L, maxSize(my_size1, my_size2))

my_stopping1 <- StoppingMinCohorts(nCohorts = 3)
my_stopping2 <- StoppingTargetProb(
  target = c(0.2, 0.35),
  prob = 0.5
)
my_stopping3 <- StoppingMinPatients(nPatients = 20)
my_stopping <- StoppingOrdinal(1L, (my_stopping1 & my_stopping2) | my_stopping3)

# Initialize the design.
design <- DesignOrdinal(
  model = LogisticLogNormalOrdinal(
    mean = c(-3, -4, 1),
    cov = diag(c(3, 4, 1)),
    ref_dose = 50
  ),
  next_best = NextBestOrdinal(
    1L,
    NextBestNCRM(
      target = c(0.2, 0.35),
      overdose = c(0.35, 1),
      max_overdose_prob = 0.25
    )
  ),
  stopping = my_stopping,
```

```
  increments = IncrementsOrdinal(
    1L,
    IncrementsRelative(
      intervals = c(0, 20),
      increments = c(1, 0.33)
    )
  ),
  cohort_size = my_size,
  data = DataOrdinal(
    doseGrid = c(1, 3, 5, 10, 15, 20, 25, 40, 50, 80, 100),
    yCategories = c("No tox" = 0L, "Sub-tox AE" = 1L, "DLT" = 2L)
  ),
  starting_dose = 3
)
```

---

dose *Computing the Doses for a given independent variable, Model and Samples*

Description

[Stable]

A function that computes the dose reaching a specific target value of a given variable that dose depends on. The meaning of this variable depends on the type of the model. For instance, for single agent dose escalation model or pseudo DLE (dose-limiting events)/toxicity model, this variable represents the a probability of the occurrence of a DLE. For efficacy models, it represents expected efficacy. The doses are computed based on the samples of the model parameters (samples).

Usage

```
dose(x, model, samples, ...)

## S4 method for signature 'numeric,LogisticNormal,Samples'
dose(x, model, samples)

## S4 method for signature 'numeric,LogisticLogNormal,Samples'
dose(x, model, samples)

## S4 method for signature 'numeric,LogisticLogNormalOrdinal,Samples'
dose(x, model, samples, grade)

## S4 method for signature 'numeric,LogisticLogNormalSub,Samples'
dose(x, model, samples)

## S4 method for signature 'numeric,ProbitLogNormal,Samples'
dose(x, model, samples)

## S4 method for signature 'numeric,ProbitLogNormalRel,Samples'
```

```
dose(x, model, samples)

## S4 method for signature 'numeric,LogisticLogNormalGrouped,Samples'
dose(x, model, samples, group)

## S4 method for signature 'numeric,LogisticKadane,Samples'
dose(x, model, samples)

## S4 method for signature 'numeric,LogisticKadaneBetaGamma,Samples'
dose(x, model, samples)

## S4 method for signature 'numeric,LogisticNormalMixture,Samples'
dose(x, model, samples)

## S4 method for signature 'numeric,LogisticNormalFixedMixture,Samples'
dose(x, model, samples)

## S4 method for signature 'numeric,LogisticLogNormalMixture,Samples'
dose(x, model, samples)

## S4 method for signature 'numeric,DualEndpoint,Samples'
dose(x, model, samples)

## S4 method for signature 'numeric,LogisticIndepBeta,Samples'
dose(x, model, samples)

## S4 method for signature 'numeric,LogisticIndepBeta,missing'
dose(x, model)

## S4 method for signature 'numeric,Effloglog,missing'
dose(x, model)

## S4 method for signature 'numeric,EffFlexi,Samples'
dose(x, model, samples)

## S4 method for signature 'numeric,OneParLogNormalPrior,Samples'
dose(x, model, samples)

## S4 method for signature 'numeric,OneParExpPrior,Samples'
dose(x, model, samples)
```

Arguments

x	(proportion or numeric) a value of an independent variable on which dose depends. The following recycling rule applies when samples is not missing: vectors of size 1 will be recycled to the size of the sample (i.e. size(samples)). Otherwise, x must have the same size as the sample.

model	(GeneralModel or ModelPseudo) the model.
samples	(Samples) the samples of model's parameters that will be used to compute the resulting doses. Can also be missing for some models.
...	model specific parameters when samples are not used.
grade	(integer) The toxicity grade for which probabilities are required
group	(character or factor) for LogisticLogNormalGrouped, indicating whether to calculate the dose for the mono or for the combo arm.

Details

The dose() function computes the doses corresponding to a value of a given independent variable, using samples of the model parameter(s). If you work with multivariate model parameters, then assume that your model specific dose() method receives a samples matrix where the rows correspond to the sampling index, i.e. the layout is then nSamples x dimParameter.

Value

A number or numeric vector with the doses. If non-scalar samples were used, then every element in the returned vector corresponds to one element of a sample. Hence, in this case, the output vector is of the same length as the sample vector. If scalar samples were used or no samples were used, e.g. for pseudo DLE/toxicity model, then the output is of the same length as the length of the prob.

Functions

- dose(x = numeric, model = LogisticNormal, samples = Samples): compute the dose level reaching a specific target probability of the occurrence of a DLE (x).
- dose(x = numeric, model = LogisticLogNormal, samples = Samples): compute the dose level reaching a specific target probability of the occurrence of a DLE (x).
- dose(x = numeric, model = LogisticLogNormalOrdinal, samples = Samples): compute the dose level reaching a specific target probability of the occurrence of a DLE (x).

  In the case of a LogisticLogNormalOrdinal model, dose returns only the probability of toxicity at the given grade or higher
- dose(x = numeric, model = LogisticLogNormalSub, samples = Samples): compute the dose level reaching a specific target probability of the occurrence of a DLE (x).
- dose(x = numeric, model = ProbitLogNormal, samples = Samples): compute the dose level reaching a specific target probability of the occurrence of a DLE (x).
- dose(x = numeric, model = ProbitLogNormalRel, samples = Samples): compute the dose level reaching a specific target probability of the occurrence of a DLE (x).
- dose(x = numeric, model = LogisticLogNormalGrouped, samples = Samples): method for LogisticLogNormalGrouped which needs group argument in addition.
- dose(x = numeric, model = LogisticKadane, samples = Samples): compute the dose level reaching a specific target probability of the occurrence of a DLE (x).

- dose(x = numeric, model = LogisticKadaneBetaGamma, samples = Samples): compute the dose level reaching a specific target probability of the occurrence of a DLE (x).
- dose(x = numeric, model = LogisticNormalMixture, samples = Samples): compute the dose level reaching a specific target probability of the occurrence of a DLE (x).
- dose(x = numeric, model = LogisticNormalFixedMixture, samples = Samples): compute the dose level reaching a specific target probability of the occurrence of a DLE (x).
- dose(x = numeric, model = LogisticLogNormalMixture, samples = Samples): compute the dose level reaching a specific target probability of the occurrence of a DLE (x).
- dose(x = numeric, model = DualEndpoint, samples = Samples): compute the dose level reaching a specific target probability of the occurrence of a DLE (x).
- dose(x = numeric, model = LogisticIndepBeta, samples = Samples): compute the dose level reaching a specific target probability of the occurrence of a DLE (x).
- dose(x = numeric, model = LogisticIndepBeta, samples = missing): compute the dose level reaching a specific target probability of the occurrence of a DLE (x). All model parameters (except x) should be present in the model object.
- dose(x = numeric, model = Effloglog, samples = missing): compute the dose level reaching a specific target probability of the occurrence of a DLE (x). All model parameters (except x) should be present in the model object.
- dose(x = numeric, model = EffFlexi, samples = Samples): compute the dose level reaching a specific target probability of the occurrence of a DLE (x). For this method x must be a scalar.
- dose(x = numeric, model = OneParLogNormalPrior, samples = Samples): compute the dose level reaching a specific target probability of the occurrence of a DLT (x).
- dose(x = numeric, model = OneParExpPrior, samples = Samples): compute the dose level reaching a specific target probability of the occurrence of a DLT (x).

Note

The dose() and prob() methods are the inverse of each other, for all dose() methods for which its first argument, i.e. a given independent variable that dose depends on, represents toxicity probability.

See Also

doseFunction(), prob(), efficacy().

Examples

```
# Create some data.
my_data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  cohort = c(0, 1, 2, 3, 4, 5, 5, 5),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)

# Initialize a model, e.g. 'LogisticLogNormal'.
```

```
my_model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 56
)

# Get samples from posterior.
my_options <- McmcOptions(burnin = 100, step = 2, samples = 20)
my_samples <- mcmc(data = my_data, model = my_model, options = my_options)

# Posterior for the dose achieving Prob(DLT) = 0.45.
dose(x = 0.45, model = my_model, samples = my_samples)

# Create data from the 'Data' (or 'DataDual') class.
dlt_data <- Data(
  x = c(25, 50, 25, 50, 75, 300, 250, 150),
  y = c(0, 0, 0, 0, 0, 1, 1, 0),
  doseGrid = seq(from = 25, to = 300, by = 25)
)

# Initialize a toxicity model using 'LogisticIndepBeta' model.
dlt_model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = dlt_data
)

# Get samples from posterior.
dlt_sample <- mcmc(data = dlt_data, model = dlt_model, options = my_options)

# Posterior for the dose achieving Prob(DLT) = 0.45.
dose(x = 0.45, model = dlt_model, samples = dlt_sample)
dose(x = c(0.45, 0.6), model = dlt_model)
data_ordinal <- .DefaultDataOrdinal()
model <- .DefaultLogisticLogNormalOrdinal()
options <- .DefaultMcmcOptions()
samples <- mcmc(data_ordinal, model, options)

dose(0.25, model, samples, grade = 2L)
```

`doseFunction` *Getting the Dose Function for a Given Model Type*

Description

[Experimental]

A function that returns a `dose()` method that computes the dose reaching a specific target value of a given independent variable, based on the model specific parameters.

Usage

```
doseFunction(model, ...)

## S4 method for signature 'GeneralModel'
doseFunction(model, ...)

## S4 method for signature 'ModelPseudo'
doseFunction(model, ...)

## S4 method for signature 'LogisticLogNormalOrdinal'
doseFunction(model, grade, ...)
```

Arguments

model	(GeneralModel or ModelPseudo) the model.
...	model specific parameters.
grade	(integer) the toxicity grade for which the dose function is required

Value

A dose() method that computes doses.

Functions

- doseFunction(GeneralModel):
- doseFunction(ModelPseudo):
- doseFunction(LogisticLogNormalOrdinal):

See Also

dose(), probFunction().

Examples

```
my_model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 50
)

dose_fun <- doseFunction(my_model, alpha0 = 2, alpha1 = 3)
dose_fun(0.6)
data_ordinal <- .DefaultDataOrdinal()
model <- .DefaultLogisticLogNormalOrdinal()
options <- .DefaultMcmcOptions()
suppressWarnings({
  samples <- mcmc(data_ordinal, model, options)
```

```
})

doseFunction(model, alpha1 = samples@data$alpha2, beta = samples@data$beta, grade = 1L)(x = 0.75)
doseFunction(model, alpha2 = samples@data$alpha2, beta = samples@data$beta, grade = 2L)(x = 0.25)
```

`dose_grid_range` *Getting the Dose Grid Range*

Description

[Stable]

A function that returns a vector of length two with the minimum and maximum dose in a grid. It returns `c(-Inf, Inf)` if the range cannot be determined, which happens when the dose grid is empty. User can choose whether the placebo dose (if any) should be counted or not.

[Experimental]

Usage

```
dose_grid_range(object, ...)

## S4 method for signature 'Data'
dose_grid_range(object, ignore_placebo = TRUE)

## S4 method for signature 'DataOrdinal'
dose_grid_range(object, ignore_placebo = TRUE)
```

Arguments

`object`	(`Data`) object with dose grid.
`...`	further arguments passed to class-specific methods.
`ignore_placebo`	(`flag`) should placebo dose (if any) not be counted?

Value

A `numeric` vector containing the minimum and maximum of all the doses in a grid or `c(-Inf, Inf)`.

Examples

```
my_data <- Data(
  x = c(10, 50, 90, 100, 0.001, 20, 30, 30),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  ID = 1:8,
  cohort = c(1L, 2L, 3L, 4L, 5L, 5L, 6L, 6L),
  doseGrid = c(0.001, seq(from = 10, to = 100, by = 10)),
```

```
  placebo = TRUE
)
dose_grid_range(my_data)
dose_grid_range(my_data, ignore_placebo = FALSE)
data <- DataOrdinal(
  x = c(10, 20, 30, 40, 50, 50, 50, 60, 60, 60),
  y = as.integer(c(0, 0, 0, 0, 0, 1, 0, 0, 1, 2)),
  ID = 1L:10L,
  cohort = as.integer(c(1:4, 5, 5, 5, 6, 6, 6)),
  doseGrid = c(seq(from = 10, to = 100, by = 10)),
  yCategories = c("No tox" = 0L, "Sub-tox AE" = 1L, "DLT" = 2L),
  placebo = FALSE
)

dose_grid_range(data)
```

`DualDesign-class` `DualDesign`

Description

[Stable]

`DualDesign` is the class for the dual-endpoint CRM design. This class has special requirements for the `model` and `data` slots in comparison to the parent class `Design`.

Usage

```
DualDesign(model, data, ...)

.DefaultDualDesign()
```

Arguments

`model`	(`DualEndpoint`) see slot definition.
`data`	(`DataDual`) see slot definition.
`...`	Arguments passed on to `Design` `stopping` (`Stopping`) see slot definition. `increments` (`Increments`) see slot definition. `pl_cohort_size` (`CohortSize`) see slot definition.

Slots

`model` (`DualEndpoint`)
: the model to be used.

`data` (`DataDual`)
: specifies dose grid, any previous data, etc.

Note

the `nextBest` slot can be of any class, this allows for easy comparison with recommendation methods that don't use the biomarker information.

Typically, end users will not use the `.DefaultDualDesign()` function.

Examples

```
empty_data <- DataDual(doseGrid = c(1, 3, 5, 10, 15, 20, 25, 40, 50, 80, 100))

# Initialize the CRM model.
my_model <- DualEndpointRW(
  mean = c(0, 1),
  cov = matrix(c(1, 0, 0, 1), nrow = 2),
  sigma2betaW = 0.01,
  sigma2W = c(a = 0.1, b = 0.1),
  rho = c(a = 1, b = 1),
  rw1 = TRUE
)

# Choose the rule for selecting the next dose.
my_next_best <- NextBestDualEndpoint(
  target = c(0.9, 1),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Choose the rule for the cohort-size.
my_size1 <- CohortSizeRange(
  intervals = c(0, 30),
  cohort_size = c(1, 3)
)
my_size2 <- CohortSizeDLT(
  intervals = c(0, 1),
  cohort_size = c(1, 3)
)
my_size <- maxSize(my_size1, my_size2)

# Choose the rule for stopping.
my_stopping1 <- StoppingTargetBiomarker(
  target = c(0.9, 1),
  prob = 0.5
)
my_stopping <- my_stopping1 | StoppingMinPatients(40)
```

```
# Choose the rule for dose increments.
my_increments <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

# Initialize the design.
design <- DualDesign(
  model = my_model,
  data = empty_data,
  nextBest = my_next_best,
  stopping = my_stopping,
  increments = my_increments,
  cohort_size = my_size,
  startingDose = 3
)
```

DualEndpoint-class DualEndpoint

Description

[Experimental]

DualEndpoint is the general class for the dual endpoint model.

Usage

```
DualEndpoint(mean, cov, ref_dose = 1, use_log_dose = FALSE, sigma2W, rho)

.DefaultDualEndpoint()
```

Arguments

mean	(numeric) for the probit toxicity model, the prior mean vector.
cov	(matrix) for the probit toxicity model, the prior covariance matrix. The precision matrix is internally calculated as an inverse of cov.
ref_dose	(number) for the probit toxicity model, the reference dose $x*$ (strictly positive number).
use_log_dose	(flag) for the probit toxicity model, whether a log transformation of the (standardized) dose should be used?
sigma2W	(numeric) the biomarker variance. Either a fixed value or Inverse-Gamma distribution parameters, i.e. vector with two elements named a and b.

`rho` (`numeric`)
either a fixed value for the correlation (between `-1` and `1`), or a named vector with two elements named `a` and `b` for the Beta prior on the transformation `kappa = (rho + 1) / 2`, which is in `(0, 1)`. For example, `a = 1, b = 1` leads to a uniform prior on `rho`.

Details

The idea of the dual-endpoint models is to model not only the dose-toxicity relationship, but also to model, at the same time, the relationship of a PD biomarker with the dose. The sub-classes of this class define how the dose-biomarker relationship is parametrized. This class here shall contain all the common features to reduce duplicate code. (This class however, must not be virtual as we need to create objects of it during the construction of subclass objects.)

The dose-toxicity relationship is modeled with probit regression model

$$probit[p(x)] = betaZ1 + betaZ2 * x/x*,$$

or

$$probit[p(x)] = betaZ1 + betaZ2 * log(x/x*),$$

in case when the option `use_log_dose` is `TRUE`. Here, $p(x)$ is the probability of observing a DLT for a given dose x and $x*$ is the reference dose. The prior

$$(betaZ1, log(betaZ2))\ Normal(mean, cov).$$

For the biomarker response w at a dose x, we assume

$$w(x)\ Normal(f(x), sigma2W),$$

where $f(x)$ is a function of the dose x, which is further specified in sub-classes. The biomarker variance $sigma2W$ can be fixed or assigned an Inverse-Gamma prior distribution; see the details below under slot `sigma2W`.

Finally, the two endpoints y (the binary DLT variable) and w (the biomarker) can be correlated, by assuming a correlation of level rho between the underlying continuous latent toxicity variable z and the biomarker w. Again, this correlation can be fixed or assigned a prior distribution from the scaled Beta family; see the details below under slot `rho`.

Please see the example vignette by typing `crmPackExample()` for a full example.

Slots

`betaZ_params` (`ModelParamsNormal`)
for the probit toxicity model, it contains the prior mean, covariance matrix and precision matrix which is internally calculated as an inverse of the covariance matrix.

`ref_dose` (`positive_number`)
for the probit toxicity model, the reference dose.

`use_log_dose` (`flag`)
for the probit toxicity model, whether a log transformation of the (standardized) dose should be used?

`sigma2W` (`numeric`)
: the biomarker variance. Either a fixed value or Inverse-Gamma distribution parameters, i.e. vector with two elements named `a` and `b`.

`rho` (`numeric`)
: either a fixed value for the correlation (between `-1` and `1`), or a named vector with two elements named `a` and `b` for the Beta prior on the transformation `kappa = (rho + 1) / 2`, which is in `(0, 1)`. For example, `a = 1, b = 1` leads to a uniform prior on `rho`.

`use_fixed` (`logical`)
: indicates whether a fixed value for `sigma2W` or `rho` (for each parameter separately) is used or not. This slot is needed for internal purposes and must not be touched by the user.

Note

Typically, end users will not use the `.DefaultDualEndpoint()` function.

See Also

`DualEndpointRW`, `DualEndpointBeta`, `DualEndpointEmax`.

DualEndpointBeta-class
DualEndpointBeta

Description

[Experimental]

`DualEndpointBeta` is the class for the dual endpoint model with beta function for dose-biomarker relationship.

Usage

```
DualEndpointBeta(E0, Emax, delta1, mode, ref_dose_beta = 1, ...)

.DefaultDualEndpointBeta()
```

Arguments

`E0`	(`numeric`) either a fixed number or the two uniform distribution parameters.
`Emax`	(`numeric`) either a fixed number or the two uniform distribution parameters.
`delta1`	(`numeric`) either a fixed positive number or the two parameters of the uniform distribution, that can take only positive values.

mode	(numeric) either a fixed positive number or the two parameters of the uniform distribution, that can take only positive values.
ref_dose_beta	(number) the reference dose $x*$ (strictly positive number). Note that this is different from the ref_dose in the inherited DualEndpoint model).
...	parameters passed to DualEndpoint().

Details

This class extends the DualEndpoint class so that the dose-biomarker relationship $f(x)$ is modelled by a parametric, rescaled beta density function:

$$f(x) = E0 + (Emax - E0) * Beta(delta1, delta2) * (x/x*)^{delta1} * (1 - x/x*)^{delta2},$$

where $x*$ is the maximum dose (end of the dose range to be considered), $delta1$ and $delta2$ are the two beta function parameters, and $E0$, $Emax$ are the minimum and maximum levels, respectively. For ease of interpretation, we use the parametrization based on $delta1$ and the mode, where

$$mode = delta1/(delta1 + delta2),$$

so that multiplying this by $x*$ gives the mode on the dose grid.

All parameters can currently be assigned uniform distributions or be fixed in advance. Note that E0 and Emax can have negative values or uniform distributions reaching into negative range, while delta1 and mode must be positive or have uniform distributions in the positive range.

Slots

E0 (numeric)
: either a fixed number or the two uniform distribution parameters.

Emax (numeric)
: either a fixed number or the two uniform distribution parameters.

delta1 (numeric)
: either a fixed positive number or the two parameters of the uniform distribution, that can take only positive values.

mode (numeric)
: either a fixed positive number or the two parameters of the uniform distribution, that can take only positive values.

ref_dose_beta (positive_number)
: the reference dose $x*$ (note that this is different from the ref_dose in the inherited DualEndpoint model).

Note

Typically, end users will not use the .DefaultDualEndpointBeta() function.

See Also

DualEndpoint, DualEndpointRW, DualEndpointEmax.

Examples

```
my_model <- DualEndpointBeta(
  mean = c(0, 1),
  cov = matrix(c(1, 0, 0, 1), nrow = 2),
  ref_dose = 10,
  use_log_dose = TRUE,
  sigma2W = c(a = 0.1, b = 0.1),
  rho = c(a = 1, b = 1),
  E0 = c(0, 100),
  Emax = c(0, 500),
  delta1 = c(0, 5),
  mode = c(1, 15),
  ref_dose_beta = 1000
)
```

DualEndpointEmax-class

DualEndpointEmax

Description

[Experimental]

DualEndpointEmax is the class for the dual endpoint model with Emax function for dose-biomarker relationship.

Usage

```
DualEndpointEmax(E0, Emax, ED50, ref_dose_emax = 1, ...)

.DefaultDualEndpointEmax()
```

Arguments

E0	(numeric) either a fixed number or the two uniform distribution parameters.
Emax	(numeric) either a fixed number or the two uniform distribution parameters.
ED50	(numeric) either a fixed number or the two uniform distribution parameters.
ref_dose_emax	(number) the reference dose $x*$ (strictly positive number). Note that this is different from the ref_dose in the inherited DualEndpoint model).
...	parameters passed to DualEndpoint().

Details

This class extends the DualEndpoint class so that the dose-biomarker relationship $f(x)$ is modelled by a parametric Emax function:

$$f(x) = E0 + [(Emax - E0) * (x/x*)]/[ED50 + (x/x*)],$$

where $x*$ is a reference dose, $E0$ and $Emax$ are the minimum and maximum levels for the biomarker, and $ED50$ is the dose achieving half of the maximum effect $0.5*Emax$. All parameters can currently be assigned uniform distributions or be fixed.

Slots

`E0` (`numeric`)
: either a fixed number or the two uniform distribution parameters.

`Emax` (`numeric`)
: either a fixed number or the two uniform distribution parameters.

`ED50` (`numeric`)
: either a fixed number or the two uniform distribution parameters.

`ref_dose_emax` (`positive_number`)
: the reference dose $x*$ (note that this is different from the `ref_dose` in the inherited DualEndpoint model).

Note

Typically, end users will not use the `.DefaultDualEndpointEmax()` function.

See Also

DualEndpoint, DualEndpointRW, DualEndpointBeta.

Examples

```
my_model <- DualEndpointEmax(
  mean = c(0, 1),
  cov = matrix(c(1, 0, 0, 1), nrow = 2),
  sigma2W = c(a = 0.1, b = 0.1),
  rho = c(a = 1, b = 1),
  E0 = c(0, 100),
  Emax = c(0, 500),
  ED50 = c(10, 200),
  ref_dose_emax = 1000
)
```

DualEndpointRW-class DualEndpointRW

Description

[Experimental]

DualEndpointRW is the class for the dual endpoint model with random walk prior for biomarker.

Usage

```
DualEndpointRW(sigma2betaW, rw1 = TRUE, ...)

.DefaultDualEndpointRW()
```

Arguments

sigma2betaW	(numeric) the prior variance factor of the random walk prior for the biomarker model. Either a fixed value or Inverse-Gamma distribution parameters, i.e. vector with two elements named a and b.
rw1	(flag) for specifying the random walk prior on the biomarker level. When TRUE, random walk of first order is used. Otherwise, the random walk of second order is used.
...	parameters passed to DualEndpoint().

Details

This class extends the DualEndpoint class so that the dose-biomarker relationship $f(x)$ is modelled by a non-parametric random walk of first or second order. That means, for the first order random walk we assume

$$betaW_i - betaW_i - 1\ Normal(0, (x_i - x_i - 1) * sigma2betaW),$$

where $betaW_i = f(x_i)$ is the biomarker mean at the i-th dose gridpoint x_i. For the second order random walk, the second-order differences instead of the first-order differences of the biomarker means follow the normal distribution with 0 mean and $2 * (x_i - x_i - 2) * sigma2betaW$ variance.

The variance parameter $sigma2betaW$ is important because it steers the smoothness of the function $f(x)$, i.e.: if it is large, then $f(x)$ will be very wiggly; if it is small, then $f(x)$ will be smooth. This parameter can either be a fixed value or assigned an inverse gamma prior distribution.

Slots

sigma2betaW (numeric)
: the prior variance factor of the random walk prior for the biomarker model. Either a fixed value or Inverse-Gamma distribution parameters, i.e. vector with two elements named a and b.

`rw1` (`flag`)
: for specifying the random walk prior on the biomarker level. When `TRUE`, random walk of first order is used. Otherwise, the random walk of second order is used.

Note

Non-equidistant dose grids can be used now, because the difference $x_i - x_i - 1$ is included in the modelling assumption above. Please note that due to impropriety of the random walk prior distributions, it is not possible to produce MCMC samples with empty data objects (i.e., sample from the prior). This is not a bug, but a theoretical feature of this model.

Typically, end users will not use the `.DefaultDualEndpointRW()` function.

See Also

`DualEndpoint`, `DualEndpointBeta`, `DualEndpointEmax`.

Examples

```
my_model <- DualEndpointRW(
  mean = c(0, 1),
  cov = matrix(c(1, 0, 0, 1), nrow = 2),
  sigma2W = c(a = 0.1, b = 0.1),
  rho = c(a = 1, b = 1),
  sigma2betaW = 0.01,
  rw1 = TRUE
)
```

DualResponsesDesign-class

DualResponsesDesign.R

Description

[Stable]

This is a class of design based on DLE responses using the `LogisticIndepBeta` model without DLE and efficacy samples. It contains all slots from the `RuleDesign` and `TDsamplesDesign` classes.

Usage

```
DualResponsesDesign(eff_model, data, ...)

.DefaultDualResponsesDesign()
```

Arguments

`eff_model`	(`ModelEff`) see slot definition.
`data`	(`DataDual`) see slot definition.
`...`	Arguments passed on to `TDDesign` `model` (`ModelTox`) see slot definition. `stopping` (`Stopping`) see slot definition. `increments` (`Increments`) see slot definition. `pl_cohort_size` (`CohortSize`) see slot definition.

Slots

`data` (`DataDual`)
: the data set.

`eff_model` (`ModelEff`)
: the pseudo efficacy model to be used.

Note

Typically, end users will not use the `.DefaultDualResponsesDesign()` function.

Examples

```
empty_data <- DataDual(doseGrid = seq(25, 300, 25))

tox_model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = empty_data
)

eff_model <- Effloglog(
  eff = c(1.223, 2.513),
  eff_dose = c(25, 300),
  nu = c(a = 1, b = 0.025),
  data = empty_data
)

my_next_best <- NextBestMaxGain(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3
)
```

```
my_increments <- IncrementsRelative(
  intervals = c(25, 300),
  increments = c(2, 2)
)

my_size <- CohortSizeConst(size = 3)
my_stopping <- StoppingMinPatients(nPatients = 36)

design <- DualResponsesDesign(
  nextBest = my_next_best,
  cohort_size = my_size,
  startingDose = 25,
  model = tox_model,
  eff_model = eff_model,
  data = empty_data,
  stopping = my_stopping,
  increments = my_increments
)
```

DualResponsesSamplesDesign-class
DualResponsesSamplesDesign

Description

[Stable]

This is a class of design based on DLE responses using the LogisticIndepBeta model with DLE and efficacy samples. It contain all slots in RuleDesign and TDsamplesDesign class objects.

Usage

```
DualResponsesSamplesDesign(eff_model, data, ...)

.DefaultDualResponsesSamplesDesign()
```

Arguments

eff_model	(ModelEff) see slot definition.
data	(DataDual) see slot definition.
...	Arguments passed on to TDsamplesDesign model (ModelTox) see slot definition. stopping (Stopping) see slot definition.

`increments` (`Increments`)
: see slot definition.

`pl_cohort_size` (`CohortSize`)
: see slot definition.

Slots

`data` (`DataDual`)
: the data set.

`eff_model` (`ModelEff`)
: the pseudo efficacy model to be used.

Note

Typically, end users will not use the `.DefaultDualResponsesSamplesDesign()` function.

Examples

```
empty_data <- DataDual(doseGrid = seq(25, 300, 25))

tox_model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = empty_data
)
options <- McmcOptions(burnin = 100, step = 2, samples = 200)
tox_samples <- mcmc(empty_data, tox_model, options)

eff_model <- Effloglog(
  eff = c(1.223, 2.513),
  eff_dose = c(25, 300),
  nu = c(a = 1, b = 0.025),
  data = empty_data
)
eff_samples <- mcmc(empty_data, eff_model, options)

my_next_best <- NextBestMaxGainSamples(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3,
  derive = function(samples) {
    as.numeric(quantile(samples, prob = 0.3))
  },
  mg_derive = function(mg_samples) {
    as.numeric(quantile(mg_samples, prob = 0.5))
  }
)

my_increments <- IncrementsRelative(
  intervals = c(25, 300),
  increments = c(2, 2)
)
```

```
my_size <- CohortSizeConst(size = 3)
my_stopping <- StoppingMinPatients(nPatients = 36)

design <- DualResponsesSamplesDesign(
  nextBest = my_next_best,
  cohort_size = my_size,
  startingDose = 25,
  model = tox_model,
  eff_model = eff_model,
  data = empty_data,
  stopping = my_stopping,
  increments = my_increments
)
```

`DualSimulations-class` `DualSimulations`

Description

[Stable]

This class captures the trial simulations from dual-endpoint model based designs. In comparison to the parent class `Simulations`, it contains additional slots to capture the dose-biomarker `fits`, and the `sigma2W` and `rho` estimates.

Usage

```
DualSimulations(rho_est, sigma2w_est, fit_biomarker, ...)

.DefaultDualSimulations()
```

Arguments

`rho_est`	(`numeric`) see `DualSimulations`
`sigma2w_est`	(`numeric`) `DualSimulations`
`fit_biomarker`	(`list`) see `DualSimulations`
`...`	additional parameters from `Simulations`

Slots

`rho_est` (`numeric`)
: vector of final posterior median rho estimates

`sigma2w_est` (`numeric`)
: vector of final posterior median sigma2W estimates

`fit_biomarker` (`list`)
: with the final dose-biomarker curve fits

Note

Typically, end users will not use the `.DefaultDualSimulations()` function.

Examples

```
data_list <- list(
  Data(
    x = 1:2,
    y = 0:1,
    doseGrid = 1:2,
    ID = 1L:2L,
    cohort = 1L:2L
  ),
  Data(
    x = 3:4,
    y = 0:1,
    doseGrid = 3:4,
    ID = 1L:2L,
    cohort = 1L:2L
  )
)

doses <- c(1, 2)
seed <- as.integer(123)

fit <- list(
  c(0.1, 0.2),
  c(0.3, 0.4)
)

stop_report <- matrix(c(TRUE, FALSE), nrow = 2)

stop_reasons <- list("A", "B")

additional_stats <- list(a = 1, b = 1)

dual_simulations_obj <- DualSimulations(
  rho_est = c(0.25, 0.35),
  sigma2w_est = c(0.15, 0.25),
  fit_biomarker = list(c(0.3, 0.4), c(0.4, 0.5)),
  fit = fit,
  stop_report = stop_report,
  stop_reasons = stop_reasons,
  additional_stats = additional_stats,
  data = data_list,
  doses = doses,
  seed = seed
)
```

DualSimulationsSummary-class DualSimulationsSummary

Description

[Stable] This class captures the summary of dual-endpoint simulations output. In comparison to its parent class `SimulationsSummary`, it has additional slots.

Usage

```
.DefaultDualSimulationsSummary()
```

Slots

`biomarker_fit_at_dose_most_selected` (`numeric`)
: fitted biomarker level at most often selected dose.

`mean_biomarker_fit` (`list`)
: list with average, lower (2.5%) and upper (97.5%) quantiles of mean fitted biomarker level at each dose

Note

Typically, end users will not use the `.DefaultDualSimulationsSummary()` function.

EffFlexi-class EffFlexi

Description

[Stable]

`EffFlexi` is the class for the efficacy model in flexible form of prior expressed in form of pseudo data. In this class, a flexible form is used to describe the relationship between the efficacy responses and the dose levels and it is specified as

$$(W|betaW, sigma2W)\ Normal(X * betaW, sigma2W * I),$$

where W is a vector of the efficacy responses, $betaW$ is a column vector of the mean efficacy responses for all dose levels, and X is the design matrix with entries I_i, j that are equal to 1 if subject i is allocated to dose j, and 0 otherwise. The $sigma2W$ is the variance of the efficacy responses which can be either a fixed number or a number from an inverse gamma distribution. This flexible form aims to capture different shapes of the dose-efficacy curve. In addition, the first (RW1) or second order (RW2) random walk model can be used for smoothing data. That is the random walk model is used to model the first or the second order differences of the mean efficacy responses to its neighboring dose levels of their mean efficacy responses.

The RW1 model is given as

$$betaW_j - betaW_j - 1)\ Normal(0, sigma2betaW),$$

and for RW2 as

$$betaW_j - 2 - 2 * betaW_j - 1 + beta_j\ Normal(0, sigma2betaW),$$

where $betaW_j$ is the vector of mean efficacy responses at dose j, and the $sigma2betaW$ is the prior variance which can be either a fixed number or a number from an inverse gamma distribution.

The `eff` and `eff_dose` are the pseudo efficacy responses and dose levels at which these pseudo efficacy responses are observed. Both, `eff` and `eff_dose` must be vectors of length at least 2. The positions of the elements specified in `eff` and `eff_dose` must correspond to each other between these vectors.

Usage

```
EffFlexi(eff, eff_dose, sigma2W, sigma2betaW, rw1 = TRUE, data)

.DefaultEffFlexi()
```

Arguments

`eff`	(`numeric`) the pseudo efficacy responses. Elements of `eff` must correspond to the elements of `eff_dose`.
`eff_dose`	(`numeric`) dose levels that correspond to pseudo efficacy responses in `eff`.
`sigma2W`	(`numeric`) the prior variance of the efficacy responses. This is either a fixed value or a named vector with two positive numbers, the shape (`a`), and the rate (`b`) parameters for the inverse gamma distribution.
`sigma2betaW`	(`numeric`) the prior variance of the random walk model used for smoothing. This is either a fixed value or a named vector with two positive numbers, the shape (`a`), and the rate (`b`) parameters for the inverse gamma distribution.
`rw1`	(`flag`) used for smoothing data for this efficacy model. If it is `TRUE`, the first-order random walk model is used for the mean efficacy responses. Otherwise, the random walk of second order is used.
`data`	(`DataDual`) observed data to update estimates of the model parameters.

Details

This model will output the updated value or the updated values of the parameters of the inverse gamma distributions for $sigma2W$ and $sigma2betaW$. The `EffFlexi` inherits all slots from `ModelEff` class.

Slots

`eff` (`numeric`)
: the pseudo efficacy responses. Each element here must represent responses treated based on one subject. It must be a vector of length at least 2 and the order of its elements must correspond to values specified in `eff_dose`.

`eff_dose` (`numeric`)
: the pseudo efficacy dose levels at which the pseudo efficacy responses are observed. It must be a vector of length at least 2 and the order of its elements must correspond to values specified in `eff`.

`sigma2W` (`numeric`)
: the prior variance of the flexible efficacy form. This is either a fixed value or a named vector with two positive numbers, the shape (`a`), and the rate (`b`) parameters for the gamma distribution.

`sigma2betaW` (`numeric`)
: the prior variance of the random walk model for the mean efficacy responses. This is either a fixed value or a named vector with two positive numbers, the shape (`a`), and the rate (`b`) parameters for the gamma distribution.

`use_fixed` (`logical`)
: indicates whether a fixed value for `sigma2W` and `sigma2betaW` (for each parameter separately) is used or not. This slot is needed for internal purposes and must not be touched by the user.

`rw1` (`flag`)
: used for smoothing data for this efficacy model. If it is `TRUE`, the first-order random walk model is used for the mean efficacy responses. Otherwise, the random walk of second order is used.

`X` (`matrix`)
: the design matrix for the efficacy responses. It is based on both the pseudo and the observed efficacy responses.

`RW` (`matrix`)
: the difference matrix for the random walk model. This slot is needed for internal purposes and must not be used by the user.

`RW_rank` (`integer`)
: is the rank of the difference matrix. This slot is needed for internal purposes and must not be used by the user.

Note

Typically, end users will not use the `.DefaultEffFlexi()` function.

Examples

```
# Obtain prior estimates for the efficacy model in flexible form, given the pseudo data.
# First define an empty data set by defining the dose levels used in the study.
# There are 12 dose levels used in the study, ranging from 25 to 300 mg with
# increments of 25.
emptydata <- DataDual(doseGrid = seq(25, 300, 25))

# Define the pseudo data, i.e.: fixed 2 dose levels 25 and 300 mg (`eff_dose`)
```

```
# and the efficacy responses 1.223 and 2.513 observed at these two dose levels (`eff`).
# The prior variance of the pseudo efficacy responses can be either a fixed value
# or two parameters for the inverse gamma distribution, the shape (a) and the
# rate (b) (`sigma2W`).
# The prior variance of the random walk model can be either a fixed value or two
# parameters for the inverse gamma distribution, the shape (a) and the rate (b)
# (`sigma2betaW`).
my_model <- EffFlexi(
  eff = c(1.223, 2.513),
  eff_dose = c(25, 300),
  sigma2W = c(a = 0.1, b = 0.1),
  sigma2betaW = c(a = 20, b = 50),
  rw1 = FALSE,
  data = emptydata
)

# Obtain estimates from the model given some observed data is available.
data <- DataDual(
  x = c(25, 50, 50, 75, 100, 100, 225, 300),
  y = c(0, 0, 0, 0, 1, 1, 1, 1),
  w = c(0.31, 0.42, 0.59, 0.45, 0.6, 0.7, 0.6, 0.52),
  doseGrid = emptydata@doseGrid
)

my_model1 <- EffFlexi(
  eff = c(1.223, 2.513),
  eff_dose = c(25, 300),
  sigma2W = c(a = 0.1, b = 0.1),
  sigma2betaW = c(a = 20, b = 50),
  rw1 = FALSE,
  data = data
)
```

`efficacy` *Computing Expected Efficacy for a Given Dose, Model and Samples*

Description

[Stable]

A function that computes the value of expected efficacy at a specified dose level, based on the model specific parameters. The model parameters (samples) are obtained based on prior specified in form of pseudo data combined with observed responses (if any).

Usage

```
efficacy(dose, model, samples, ...)

## S4 method for signature 'numeric,Effloglog,Samples'
efficacy(dose, model, samples)
```

```
## S4 method for signature 'numeric,Effloglog,missing'
efficacy(dose, model)

## S4 method for signature 'numeric,EffFlexi,Samples'
efficacy(dose, model, samples)
```

Arguments

dose	(numeric) the dose which is targeted. The following recycling rule applies when samples is not missing: vectors of size 1 will be recycled to the size of the sample (i.e. size(samples)). Otherwise, dose must have the same size as the sample.
model	(ModelEff) the efficacy model with pseudo data prior.
samples	(Samples) samples of model's parameters that will be used to compute expected efficacy values. Can also be missing for some models.
...	model specific parameters when samples are not used.

Details

The efficacy() function computes the expected efficacy for given doses, using samples of the model parameter(s). If you work with multivariate model parameters, then assume that your model specific efficacy() method receives a samples matrix where the rows correspond to the sampling index, i.e. the layout is then nSamples x dimParameter.

Value

A numeric vector with the values of expected efficacy. If non-scalar samples were used, then every element in the returned vector corresponds to one element of a sample. Hence, in this case, the output vector is of the same length as the sample vector. If scalar samples were used or no samples were used, e.g. for pseudo DLE/toxicity model, then the output is of the same length as the length of the dose.

Functions

- efficacy(dose = numeric, model = Effloglog, samples = Samples): compute the expected efficacy at a specified dose level, based on the samples of Effloglog model parameters.
- efficacy(dose = numeric, model = Effloglog, samples = missing): compute the expected efficacy at a specified dose level, based on the Effloglog model parameters. All model parameters (except dose) should be present in the model object.
- efficacy(dose = numeric, model = EffFlexi, samples = Samples): compute the expected efficacy at a specified dose level, based on the samples of EffFlexi model parameters. If a given dose in the dose vector is from outside of the dose grid range, the NA_real is returned for this dose and the warning is thrown.

See Also

dose(), prob().

Examples

```
# Obtain the expected efficacy value for a given dose, a given pseudo efficacy
# model (in flexible form for prior) and efficacy samples.

# Empty data (i.e. no observed data), dose grid only.
my_data <- DataDual(doseGrid = seq(25, 300, 25))

my_model <- EffFlexi(
  eff = c(1.223, 2.513),
  eff_dose = c(25, 300),
  sigma2W = c(a = 0.1, b = 0.1),
  sigma2betaW = c(a = 20, b = 50),
  rw1 = FALSE,
  data = my_data
)

my_options <- McmcOptions(
  burnin = 100, step = 2, samples = 200, rng_kind = "Mersenne-Twister", rng_seed = 94
)

my_samples <- mcmc(data = my_data, model = my_model, options = my_options)

# Efficacy for dose 75.
efficacy(dose = 75, model = my_model, samples = my_samples)

# Obtain the expected efficacy value for a given dose, a given pseudo efficacy
# model (linear log-log efficacy) and no samples.
my_model_ll <- Effloglog(
  eff = c(1.223, 2.513),
  eff_dose = c(25, 300),
  nu = c(a = 1, b = 0.025),
  data = my_data,
  const = 0
)

efficacy(dose = 75, model = my_model_ll)
```

---

efficacyFunction *Getting the Efficacy Function for a Given Model Type*

Description

[Experimental]

A function that returns an efficacy() function that computes expected efficacy for a given dose level, based on the model specific parameters.

Usage

```
efficacyFunction(model, ...)

## S4 method for signature 'ModelEff'
efficacyFunction(model, ...)
```

Arguments

model	(ModelEff) the model.
...	model specific parameters.

Value

A efficacy() function that computes expected efficacy.

Functions

- efficacyFunction(ModelEff):

See Also

efficacy().

Examples

```
my_data <- DataDual(
  doseGrid = c(0.001, seq(25, 300, 25)),
  placebo = TRUE
)

my_model <- Effloglog(
  eff = c(1.223, 2.513),
  eff_dose = c(25, 300),
  nu = c(a = 1, b = 0.025),
  data = my_data,
  const = 2
)

eff_fun <- efficacyFunction(my_model, theta1 = -4.8, theta2 = 3.7)
eff_fun(30)
```

`Effloglog-class` `Effloglog`

Description

[Stable]

`Effloglog` is the class for the linear log-log efficacy model using pseudo data prior. It describes the relationship between continuous efficacy responses and corresponding dose levels in log-log scale. This efficacy log-log model is given as

$$y_i = theta1 + theta2 * log(log(x_i)) + epsilon_i,$$

where y_i is the efficacy response for subject i, x_i is the dose level treated for subject i and $epsilon_i$ is the random error term of efficacy model at subject i. The error term $epsilon_i$ is a random variable that follows normal distribution with mean 0 and variance nu^{-1}, which is assumed to be the same for all subjects. There are three parameters in this model, the intercept $theta1$, the slope $theta2$ and the precision nu of the efficacy responses, also known as the inverse of the variance of the pseudo efficacy responses. It can be a fixed constant or having a gamma distribution. Therefore, a single scalar value or a vector with two positive numbers values must be specified for `nu` slot. If there are some observed efficacy responses available, in the output, `nu` will display the updated value of the precision or the updated values for the parameters of the gamma distribution. The `Effloglog` inherits all slots from `ModelEff` class.

Usage

```
Effloglog(eff, eff_dose, nu, data, const = 0)

.DefaultEffloglog()
```

Arguments

`eff`	(`numeric`) the pseudo efficacy responses. Elements of `eff` must correspond to the elements of `eff_dose`.
`eff_dose`	(`numeric`) dose levels that correspond to pseudo efficacy responses in `eff`.
`nu`	(`numeric`) the precision (inverse of the variance) of the efficacy responses. This is either a fixed value or a named vector with two positive numbers, the shape (`a`), and the rate (`b`) parameters for the gamma distribution.
`data`	(`DataDual`) observed data to update estimates of the model parameters.
`const`	(`number`) the constant value added to the dose level when the dose level value is less than or equal to 1 and a special form of the linear log-log has to applied (Yeung et al. (2015).).

Details

The prior of this model is specified in form of pseudo data. First, at least two dose levels are fixed. Then, using e.g. experts' opinion, the efficacy values that correspond to these dose levels can be obtained, The `eff` and `eff_dose` arguments represent the prior in form of the pseudo data. The `eff` represents the pseudo efficacy values. The `eff_dose` represents the dose levels at which these pseudo efficacy values are observed. Hence, the positions of the elements specified in `eff` and `eff_dose` must correspond to each other between these vectors. Since at least 2 pseudo efficacy values are needed to obtain modal estimates of the intercept and slope parameters, both `eff` and `eff_dose` must be vectors of length at least 2.

The joint prior distribution of the intercept $theta1$ and the slope $theta2$ of this model follows bivariate normal distribution with mean mu and covariance matrix $(nu * Q)^{-1}$. The mean mu is a $2x1$ column vector that contains the prior modal estimates of the intercept and the slope. Scalar nu is the precision of the pseudo efficacy responses and Q is the prior or posterior (given that observed, no DLT data is available) precision matrix. It is specified as $Q = X0^T * X0 + X^T * X$, where $X0$ is a design matrix that is based on pseudo dose levels only, and X is a design matrix that is based on dose levels corresponding to the no DLT efficacy responses observed only (if any). Hence, the $X0$ (or X) will be of size $rx2$, if there are $r >= 2$ pseudo efficacy responses specified (or if there are r no DLT efficacy responses observed in the `data`).

Slots

`eff` (`numeric`)
: the pseudo efficacy responses. Each element here must represent responses treated based on one subject. It must be a vector of length at least 2 and the order of its elements must correspond to values specified in `eff_dose`.

`eff_dose` (`numeric`)
: the pseudo efficacy dose levels at which the pseudo efficacy responses are observed. It must be a vector of length at least 2 and the order of its elements must correspond to values specified in `eff`.

`nu` (`numeric`)
: parameter of the prior precision of pseudo efficacy responses. This is either a fixed value or a named vector with two positive numbers, the shape (`a`), and the rate (`b`) parameters for the gamma distribution.

`use_fixed` (`flag`)
: indicates whether `nu` specified is a fixed value or a vector with two parameters for gamma distribution. This slot is for internal purposes only and must not be used by the user.

`theta1` (`number`)
: the intercept in this efficacy log-log model. This slot is used in output to display the resulting prior or posterior modal estimates obtained based on the pseudo and observed (if any) data.

`theta2` (`number`)
: the slope in this efficacy log-log model. This slot is used in output to display the resulting prior or posterior modal estimates obtained based on the pseudo and observed (if any) data.

`Pcov` (`matrix`)
: refers to the $2x2$ covariance matrix of the estimators of the intercept $theta1$ and the slope $theta2$ parameters in this model. This is used in output to display the resulting prior and posterior covariance matrix of $theta1$ and $theta2$ obtained, based on the pseudo and observed (if any) data. This slot is needed for internal purposes.

`X (matrix)`
: is the design matrix that is based on either the pseudo dose levels or observed dose levels (without DLT). This is used in the output to display the design matrix for the pseudo or the observed efficacy responses.

`Y (numeric)`
: is a vector that either contains the pseudo efficacy responses or observed efficacy responses (without DLT).

`mu (numeric)`
: a vector of the prior or the posterior modal estimates of the intercept ($theta1$) and the slope ($theta2$). This slot is used in output to display as the mean of the prior or posterior bivariate normal distribution for $theta1$ and $theta2$.

`Q (matrix)`
: is the prior or posterior (given that observed, no DLT data is available) precision matrix. It is specified as $Q = X0^T * X0 + X^T * X$, where $X0$ is a design matrix that is based on pseudo dose levels only, and X is a design matrix that is based on dose levels corresponding to the observed, no DLT efficacy values only (if any).

`const (number)`
: a non-negative number (default to 0), leading to the model form described above. In general, the model has the form $y_i = theta1 + theta2 * log(log(x_i + const)) + epsilon_i$, such that dose levels greater than $1 - const$ can be considered as described in Yeung et al. (2015).

Note

Typically, end users will not use the `.DefaultEffloglog()` function.

Examples

```
# Obtain prior modal estimates given the pseudo data.
# First we use an empty data set such that only the dose levels under
# investigations are given. In total, 12 dose levels are under investigation
# ranging from 25 to 300 mg with increments of 25 (i.e 25, 50, 75, ..., 300).
emptydata <- DataDual(doseGrid = seq(25, 300, 25), placebo = FALSE)

# Define the pseudo data as first by fixing two dose levels 25 and 300 mg (`eff_dose`).
# Then, the efficacy responses observed at these two dose levels are 1.223 and 2.513 (`eff`).
# We specify the prior precision of the pseudo efficacy responses (`nu`) as a vector
# with the shape (a) and the rate (b) parameters for the gamma distribution.
# Obtain modal estimates and other estimates from the model (no observations,
# only pseudo data).
my_model1 <- Effloglog(
  eff = c(1.223, 2.513),
  eff_dose = c(25, 300),
  nu = c(a = 1, b = 0.025),
  data = emptydata
)

# Observed data.
my_data <- DataDual(
  x = c(25, 50, 50, 75, 100, 100, 225, 300),
  y = c(0, 0, 0, 0, 1, 1, 1, 1),
```

```
  w = c(0.31, 0.42, 0.59, 0.45, 0.6, 0.7, 0.6, 0.52),
  doseGrid = emptydata@doseGrid
)

# Obtain posterior modal estimates and other estimates from the model given some
# observed data.
my_model2 <- Effloglog(
  eff = c(1.223, 2.513),
  eff_dose = c(25, 300),
  nu = c(a = 1, b = 0.025),
  data = my_data
)
```

---

enable_logging *Verbose Logging*

Description

[Experimental]

A family of wrappers of selected futile.logger functions that control the logging mechanism in crmPack. The crmPack uses futile.logger package for the logging purposes. All the messages logged in crmPack are logged into crmPack logger at the futile.logger::TRACE level. Hence, enabling verbose logging means that the logging threshold will be set to futile.logger::TRACE for the crmPack logger, and disabling verbose logging means that it will be set to futile.logger::FATAL.

Usage

```
enable_logging()

disable_logging()

is_logging_enabled()

log_trace(msg, ..., capture = FALSE)
```

Arguments

msg	The message to log
...	Optional arguments to populate the format string
capture	Capture print output of variables instead of interpolate

Functions

- enable_logging(): A simple wrapper of futile.logger::flog.threshold() that enables crmPack verbose logging by setting logging threshold to futile.logger::TRACE for crmPack logger.

- `disable_logging()`: A simple wrapper of `futile.logger::flog.threshold()` that disables `crmPack` verbose logging by setting logging threshold to `futile.logger::FATAL` for `crmPack` logger.
- `is_logging_enabled()`: A simple wrapper of `futile.logger::flog.logger()` that checks whether current threshold level for `crmPack` logger is verbose, which is `futile.logger::TRACE`. It returns `TRUE` if the current logging level is verbose, `FALSE` otherwise.
- `log_trace()`: A simple wrapper of `futile.logger::flog.trace()` that prints a log message in the `crmPack` logger.

`examine` *Obtain hypothetical trial course table for a design*

Description

This generic function takes a design and generates a dataframe showing the beginning of several hypothetical trial courses under the design. This means, from the generated dataframe one can read off:

- how many cohorts are required in the optimal case (no DLTs observed) in order to reach the highest dose of the specified dose grid (or until the stopping rule is fulfilled)
- assuming no DLTs are observed until a certain dose level, what the next recommended dose is for all possible number of DLTs observed
- the actual relative increments that will be used in these cases
- whether the trial would stop at a certain cohort Examining the "single trial" behavior of a dose escalation design is the first important step in evaluating a design, and cannot be replaced by studying solely the operating characteristics in "many trials". The cohort sizes are also taken from the design, assuming no DLTs occur until the dose listed.

Usage

```
examine(object, ..., maxNoIncrement = 100L)

## S4 method for signature 'Design'
examine(object, mcmcOptions = McmcOptions(), ..., maxNoIncrement)

## S4 method for signature 'RuleDesign'
examine(object, ..., maxNoIncrement = 100L)

## S4 method for signature 'DADesign'
examine(object, mcmcOptions = McmcOptions(), ..., maxNoIncrement)
```

Arguments

`object`	the design (`Design` or `RuleDesign` object) we want to examine
`...`	additional arguments (see methods)

maxNoIncrement	maximum number of contiguous next doses at 0 DLTs that are the same as before, i.e. no increment (default to 100)
mcmcOptions	object of class McmcOptions, giving the MCMC options for each evaluation in the trial. By default, the standard options are used

Value

The data frame

Functions

- examine(Design): Examine a model-based CRM
- examine(RuleDesign): Examine a rule-based design
- examine(DADesign): Examine a model-based CRM

Examples

```
# Define the dose-grid.
emptydata <- Data(doseGrid = c(1, 3, 5, 10, 15, 20, 25))


# Initialize the CRM model.
my_model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov =
    matrix(c(1, -0.5, -0.5, 1),
      nrow = 2
    ),
  ref_dose = 56
)

# Choose the rule for selecting the next dose.
my_next_best <- NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)


my_size1 <- CohortSizeRange(
  intervals = c(0, 30),
  cohort_size = c(1, 3)
)
my_size2 <- CohortSizeDLT(
  intervals = c(0, 1),
  cohort_size = c(1, 3)
)
my_size <- maxSize(my_size1, my_size2)

# Choose the rule for stopping.
my_stopping1 <- StoppingMinCohorts(nCohorts = 3)
```

```
my_stopping2 <- StoppingTargetProb(
  target = c(0.2, 0.35),
  prob = 0.5
)
my_stopping3 <- StoppingMinPatients(nPatients = 20)
my_stopping <- (my_stopping1 & my_stopping2) | my_stopping3

# Choose the rule for dose increments.
my_increments <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

# Initialize the design.
my_design <- Design(
  model = my_model,
  nextBest = my_next_best,
  stopping = my_stopping,
  increments = my_increments,
  cohort_size = my_size,
  data = emptydata,
  startingDose = 3
)

my_options <- McmcOptions(
  burnin = 10, step = 1, samples = 20, rng_kind = "Super-Duper",
  rng_seed = 94
)

examine(my_design, my_options)

# Example where examine stops because stopping rule already fulfilled.
my_stopping4 <- StoppingMinPatients(nPatients = 3)
my_stopping <- (my_stopping1 & my_stopping2) | my_stopping4

my_design <- Design(
  model = my_model,
  nextBest = my_next_best,
  stopping = my_stopping,
  increments = my_increments,
  cohort_size = my_size,
  data = emptydata,
  startingDose = 3
)

examine(my_design, mcmcOptions = my_options)

# Example where examine stops because infinite looping
# (note that here a very low threshold is used for the parameter
# "maxNoIncrement" in "examine" to keep the execution time short).
my_increments <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.00001)
```

```
)

my_stopping <- (my_stopping1 & my_stopping2) | StoppingMissingDose()

design <- Design(
  model = my_model,
  nextBest = my_next_best,
  stopping = my_stopping,
  increments = my_increments,
  cohort_size = my_size,
  data = emptydata,
  startingDose = 3
)

examine(my_design, mcmcOptions = my_options, maxNoIncrement = 2)
# Define the dose-grid
emptydata <- Data(doseGrid = c(5, 10, 15, 25, 35, 50, 80))

# inizialing a 3+3 design with constant cohort size of 3 and
# starting dose equal 5
myDesign <- RuleDesign(
  nextBest = NextBestThreePlusThree(),
  cohort_size = CohortSizeConst(size = 3L),
  data = emptydata,
  startingDose = 5
)

# Examine the design
set.seed(4235)
examine(myDesign)
# nolint start

# Define the dose-grid and PEM parameters
emptydata <- DataDA(doseGrid = c(
  0.1, 0.5, 1, 1.5, 3, 6,
  seq(from = 10, to = 80, by = 2)
), Tmax = 60)
# Initialize the mDA-CRM model
npiece_ <- 10
Tmax_ <- 60

lambda_prior <- function(k) {
  npiece_ / (Tmax_ * (npiece_ - k + 0.5))
}

model <- DALogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 56,
  npiece = npiece_,
  l = as.numeric(t(apply(as.matrix(c(1:npiece_), 1, npiece_), 2, lambda_prior))),
  c_par = 2
)
```

```
# Choose the rule for dose increments
myIncrements <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

myNextBest <- NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Choose the rule for the cohort-size
mySize1 <- CohortSizeRange(
  intervals = c(0, 30),
  cohort_size = c(1, 3)
)
mySize2 <- CohortSizeDLT(
  intervals = c(0, 1),
  cohort_size = c(1, 3)
)
mySize <- maxSize(mySize1, mySize2)

# Choose the rule for stopping
myStopping1 <- StoppingTargetProb(
  target = c(0.2, 0.35),
  prob = 0.5
)
myStopping2 <- StoppingMinPatients(nPatients = 50)

myStopping <- (myStopping1 | myStopping2)

# Choose the safety window
mysafetywindow <- SafetyWindowConst(c(6, 2), 7, 7)

# Initialize the design
design <- DADesign(
  model = model,
  increments = myIncrements,
  nextBest = myNextBest,
  stopping = myStopping,
  cohort_size = mySize,
  data = emptydata,
  safetyWindow = mysafetywindow,
  startingDose = 3
)

set.seed(4235)
# MCMC parameters are set to small values only to show this example. They should be
# increased for a real case.
# This procedure will take a while.
options <- McmcOptions(
  burnin = 10,
```

```
  step = 1,
  samples = 100,
  rng_kind = "Mersenne-Twister",
  rng_seed = 12
)
testthat::expect_warning(
  result <- examine(design, mcmcOptions = options, maxNoIncrement = 2),
  "Stopping because 2 times no increment"
)

# nolint end
```

`fit` *Fit method for the Samples class*

Description

Note this new generic function is necessary because the `fitted` function only allows the first argument `object` to appear in the signature. But we need also other arguments in the signature.

Usage

```
fit(object, model, data, ...)

## S4 method for signature 'Samples,GeneralModel,Data'
fit(
  object,
  model,
  data,
  points = data@doseGrid,
  quantiles = c(0.025, 0.975),
  middle = mean,
  ...
)

## S4 method for signature 'Samples,DualEndpoint,DataDual'
fit(object, model, data, quantiles = c(0.025, 0.975), middle = mean, ...)

## S4 method for signature 'Samples,LogisticIndepBeta,Data'
fit(
  object,
  model,
  data,
  points = data@doseGrid,
  quantiles = c(0.025, 0.975),
  middle = mean,
  ...
)
```

```
## S4 method for signature 'Samples,Effloglog,DataDual'
fit(
  object,
  model,
  data,
  points = data@doseGrid,
  quantiles = c(0.025, 0.975),
  middle = mean,
  ...
)

## S4 method for signature 'Samples,EffFlexi,DataDual'
fit(
  object,
  model,
  data,
  points = data@doseGrid,
  quantiles = c(0.025, 0.975),
  middle = mean,
  ...
)

## S4 method for signature 'Samples,LogisticLogNormalOrdinal,DataOrdinal'
fit(
  object,
  model,
  data,
  points = data@doseGrid,
  quantiles = c(0.025, 0.975),
  middle = mean,
  ...
)
```

Arguments

object	the Samples object
model	the GeneralModel object
data	the Data object
...	passed down to the prob() method.
points	at which dose levels is the fit requested? default is the dose grid
quantiles	the quantiles to be calculated (default: 0.025 and 0.975)
middle	the function for computing the middle point. Default: mean

Value

the data frame with required information (see method details)

Functions

- `fit(object = Samples, model = GeneralModel, data = Data)`: This method returns a data frame with dose, middle, lower and upper quantiles for the dose-toxicity curve
- `fit(object = Samples, model = DualEndpoint, data = DataDual)`: This method returns a data frame with dose, and middle, lower and upper quantiles, for both the dose-tox and dose-biomarker (suffix "Biomarker") curves, for all grid points (Note that currently only the grid points can be used, because the DualEndpointRW models only allow that)
- `fit(object = Samples, model = LogisticIndepBeta, data = Data)`: This method return a data frame with dose, middle lower and upper quantiles for the dose-DLE curve using DLE samples for "LogisticIndepBeta" model class
- `fit(object = Samples, model = Effloglog, data = DataDual)`: This method returns a data frame with dose, middle, lower, upper quantiles for the dose-efficacy curve using efficacy samples for "Effloglog" model class
- `fit(object = Samples, model = EffFlexi, data = DataDual)`: This method returns a data frame with dose, middle, lower and upper quantiles for the dose-efficacy curve using efficacy samples for "EffFlexi" model class
- `fit(object = Samples, model = LogisticLogNormalOrdinal, data = DataOrdinal)`: This method returns a data frame with dose, middle, lower and upper quantiles for the dose-efficacy curve using efficacy samples for the "LogisticLogNormalOrdinal" model class

Examples

```
# nolint start

# Create some data
data <- Data(x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
             y = c(0, 0, 0, 0, 0, 0, 1, 0),
             cohort = c(0, 1, 2, 3, 4, 5, 5, 5),
             doseGrid = c(0.1, 0.5, 1.5, 3, 6,
                          seq(from = 10, to = 80, by=2)))

# Initialize a model
model <- LogisticLogNormal(mean = c(-0.85, 1),
                           cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
                           ref_dose = 56)

# Get posterior for all model parameters
options <- McmcOptions(burnin = 100,
                       step = 2,
                       samples = 2000)
set.seed(94)
samples <- mcmc(data, model, options)

# Extract the posterior mean  (and empirical 2.5 and 97.5 percentile)
# for the prob(DLT) by doses
fitted <- fit(object = samples,
              model = model,
              data = data,
              quantiles=c(0.025, 0.975),
```

```
              middle=mean)


# ------------------------------------------------
# A different example using a different model
## we need a data object with doses >= 1:
data<-Data(x=c(25,50,50,75,150,200,225,300),
           y=c(0,0,0,0,1,1,1,1),
           doseGrid=seq(from=25,to=300,by=25))


model <- LogisticIndepBeta(binDLE=c(1.05,1.8),
                           DLEweights=c(3,3),
                           DLEdose=c(25,300),
                           data=data)
options <- McmcOptions(burnin=100,
                       step=2,
                       samples=200)
## samples must be from 'Samples' class (object slot in fit)
samples <- mcmc(data,model,options)

fitted <- fit(object=samples, model=model, data=data)

# nolint end
# nolint start

# Create some data
data <- DataDual(
  x=c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10,
      20, 20, 20, 40, 40, 40, 50, 50, 50),
  y=c(0, 0, 0, 0, 0, 0, 1, 0,
      0, 1, 1, 0, 0, 1, 0, 1, 1),
  w=c(0.31, 0.42, 0.59, 0.45, 0.6, 0.7, 0.55, 0.6,
      0.52, 0.54, 0.56, 0.43, 0.41, 0.39, 0.34, 0.38, 0.21),
  doseGrid=c(0.1, 0.5, 1.5, 3, 6,
             seq(from=10, to=80, by=2)))

# Initialize the Dual-Endpoint model (in this case RW1)
model <- DualEndpointRW(mean = c(0, 1),
                        cov = matrix(c(1, 0, 0, 1), nrow=2),
                        sigma2betaW = 0.01,
                        sigma2W = c(a=0.1, b=0.1),
                        rho = c(a=1, b=1),
                        rw1 = TRUE)

# Set-up some MCMC parameters and generate samples from the posterior
options <- McmcOptions(burnin=100,
                       step=2,
                       samples=500)
set.seed(94)
samples <- mcmc(data, model, options)

# Extract the posterior mean  (and empirical 2.5 and 97.5 percentile)
```

```
# for the prob(DLT) by doses and the Biomarker by doses
fitted <- fit(object = samples,
              model = model,
              data = data,
              quantiles=c(0.025, 0.975),
              middle=mean)

# nolint end
##Obtain the 'fit' the middle, uppper and lower quantiles for the dose-DLE curve
## at all dose levels using a DLE sample, a DLE model and the data
## samples must be from 'Samples' class (object slot)
## we need a data object with doses >= 1:
data<-Data(x=c(25,50,50,75,150,200,225,300),
           y=c(0,0,0,0,1,1,1,1),
           doseGrid=seq(from=25,to=300,by=25))
## model must be from 'Model' or 'ModelTox' class e.g using 'LogisticIbdepBeta' model class
model<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=data)
##options for MCMC
options<-McmcOptions(burnin=100,step=2,samples=200)
## samples must be from 'Samples' class (object slot in fit)
samples<-mcmc(data,model,options)

fit(object=samples, model=model,data=data)
##Obtain the 'fit' the middle, uppper and lower quantiles for the dose-efficacy curve
## at all dose levels using an efficacy sample, a pseudo efficacy model and the data
## data must be from 'DataDual' class
data<-DataDual(x=c(25,50,25,50,75,300,250,150),
               y=c(0,0,0,0,0,1,1,0),
               w=c(0.31,0.42,0.59,0.45,0.6,0.7,0.6,0.52),
               doseGrid=seq(25,300,25),
               placebo=FALSE)
## model must be from 'ModelEff' e.g using 'Effloglog' class
Effmodel<-Effloglog(c(1.223,2.513),c(25,300),nu=c(a=1,b=0.025),data=data,c=0)
## samples must be from 'Samples' class (object slot in fit)
options<-McmcOptions(burnin=100,step=2,samples=200)
Effsamples <- mcmc(data=data,model=Effmodel,options=options)
fit(object=Effsamples, model=Effmodel,data=data)
# nolint start

##Obtain the 'fit' the middle, uppper and lower quantiles for the dose-efficacy curve
## at all dose levels using an efficacy sample, the 'EffFlexi' efficacy model and the data
## data must be from 'DataDual' class
data<-DataDual(x=c(25,50,25,50,75,300,250,150),
               y=c(0,0,0,0,0,1,1,0),
               w=c(0.31,0.42,0.59,0.45,0.6,0.7,0.6,0.52),
               doseGrid=seq(25,300,25),
               placebo=FALSE)
## model must be from 'ModelEff' e.g using 'Effloglog' class
Effmodel<- EffFlexi(eff=c(1.223, 2.513),eff_dose=c(25,300),
                    sigma2W=c(a=0.1,b=0.1),sigma2betaW=c(a=20,b=50),rw1 = FALSE,data=data)

## samples must be from 'Samples' class (object slot in fit)
options<-McmcOptions(burnin=100,step=2,samples=200)
```

```
Effsamples <- mcmc(data=data,model=Effmodel,options=options)
fit(object=Effsamples, model=Effmodel,data=data)

# nolint end
model <- .DefaultLogisticLogNormalOrdinal()
ordinal_data <- .DefaultDataOrdinal()
options <- .DefaultMcmcOptions()
samples <- mcmc(ordinal_data, model, options)

grade1_fit <- fit(samples, model, ordinal_data, grade = 1L)
grade2_fit <- fit(samples, model, ordinal_data, grade = 2L)
```

`fitGain` *Get the fitted values for the gain values at all dose levels based on a given pseudo DLE model, DLE sample, a pseudo efficacy model, a Efficacy sample and data. This method returns a data frame with dose, middle, lower and upper quantiles of the gain value samples*

Description

Get the fitted values for the gain values at all dose levels based on a given pseudo DLE model, DLE sample, a pseudo efficacy model, a Efficacy sample and data. This method returns a data frame with dose, middle, lower and upper quantiles of the gain value samples

Usage

```
fitGain(DLEmodel, DLEsamples, Effmodel, Effsamples, data, ...)

## S4 method for signature 'ModelTox,Samples,ModelEff,Samples,DataDual'
fitGain(
  DLEmodel,
  DLEsamples,
  Effmodel,
  Effsamples,
  data,
  points = data@doseGrid,
  quantiles = c(0.025, 0.975),
  middle = mean,
  ...
)
```

Arguments

`DLEmodel`	the DLE pseudo model of `ModelTox` class object
`DLEsamples`	the DLE samples of `Samples` class object
`Effmodel`	the efficacy pseudo model of `ModelEff` class object
`Effsamples`	the efficacy samples of `Samples` class object

data	the data input of DataDual class object
...	additional arguments for methods
points	at which dose levels is the fit requested? default is the dose grid
quantiles	the quantiles to be calculated (default: 0.025 and 0.975)
middle	the function for computing the middle point. Default: mean

Functions

- fitGain(DLEmodel = ModelTox, DLEsamples = Samples, Effmodel = ModelEff, Effsamples = Samples, data = DataDual): This method returns a data frame with dose, middle, lower, upper quantiles for the gain values obtained given the DLE and the efficacy samples

Examples

```
##Obtain the 'fitGain' the middle, uppper and lower quantiles for the samples of gain values
## at all dose levels using a pseudo DLE model, a DLE sample, a pseudo Efficacy model and
## a efficacy sample
## data must be from 'DataDual' class
data<-DataDual(x=c(25,50,25,50,75,300,250,150),
               y=c(0,0,0,0,0,1,1,0),
               w=c(0.31,0.42,0.59,0.45,0.6,0.7,0.6,0.52),
               doseGrid=seq(25,300,25),
               placebo=FALSE)
## DLE model must be from 'ModelTox' class e.g using 'LogisticIndepBeta' model
DLEmodel<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=data)

## Efficacy model must be from 'ModelEff' class e.g using 'Effloglog' model
Effmodel<-Effloglog(c(1.223,2.513),c(25,300),nu=c(a=1,b=0.025),data=data,c=0)
## samples must be from 'Samples' class (object slot in fit)
options<-McmcOptions(burnin=100,step=2,samples=200)
##set up the same data set in class 'Data' for MCMC sampling for DLE
data1 <- Data(x=data@x,y=data@y,doseGrid=data@doseGrid)

DLEsamples <- mcmc(data=data1,model=DLEmodel,options=options)
Effsamples <- mcmc(data=data,model=Effmodel,options=options)

fitGain(DLEmodel=DLEmodel,DLEsamples=DLEsamples,
        Effmodel=Effmodel, Effsamples=Effsamples,data=data)
##Obtain the 'fitGain' the middle, uppper and lower quantiles for the samples of gain values
## at all dose levels using a pseudo DLE model, a DLE sample, a pseudo Efficacy model and
## a efficacy sample
## data must be from 'DataDual' class
data<-DataDual(x=c(25,50,25,50,75,300,250,150),
               y=c(0,0,0,0,0,1,1,0),
               w=c(0.31,0.42,0.59,0.45,0.6,0.7,0.6,0.52),
               doseGrid=seq(25,300,25),
               placebo=FALSE)
## DLE model must be from 'ModelTox' class e.g using 'LogisticIndepBeta' model
DLEmodel<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=data)

## Efficacy model must be from 'ModelEff' class e.g using 'Effloglog' model
```

```
Effmodel<-Effloglog(c(1.223,2.513),c(25,300),nu=c(a=1,b=0.025),data=data,c=0)
## samples must be from 'Samples' class (object slot in fit)
options<-McmcOptions(burnin=100,step=2,samples=200)
##set up the same data set in class 'Data' for MCMC sampling for DLE
data1 <- Data(x=data@x,y=data@y,doseGrid=data@doseGrid)

DLEsamples <- mcmc(data=data1,model=DLEmodel,options=options)
Effsamples <- mcmc(data=data,model=Effmodel,options=options)

fitGain(DLEmodel=DLEmodel,DLEsamples=DLEsamples,
        Effmodel=Effmodel, Effsamples=Effsamples,data=data)
```

`fitPEM` *Get the fitted DLT free survival (piecewise exponential model). This function returns a data frame with dose, middle, lower and upper quantiles for the* `PEM` *curve. If hazard=TRUE,*

Description

Get the fitted DLT free survival (piecewise exponential model). This function returns a data frame with dose, middle, lower and upper quantiles for the `PEM` curve. If hazard=TRUE,

Usage

```
fitPEM(
  object,
  model,
  data,
  quantiles = c(0.025, 0.975),
  middle = mean,
  hazard = FALSE,
  ...
)

## S4 method for signature 'Samples,DALogisticLogNormal,DataDA'
fitPEM(
  object,
  model,
  data,
  quantiles = c(0.025, 0.975),
  middle = mean,
  hazard = FALSE,
  ...
)
```

Arguments

object	mcmc samples
model	the mDA-CRM model
data	the data input, a DataDA class object
quantiles	the quantiles to be calculated (default: 0.025 and 0.975)
middle	the function for computing the middle point. Default: mean
hazard	should the the hazard over time be plotted based on the PEM? (not default) Otherwise ...
...	additional arguments for methods

Functions

- fitPEM(object = Samples, model = DALogisticLogNormal, data = DataDA): This method works for the DALogisticLogNormal model class.

Examples

```
# nolint start

# Create the data
data <- DataDA(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 1, 1, 0, 0, 1, 0),
  ID = 1L:8L,
  cohort = as.integer(c(1:5, 6, 6, 6)),
  doseGrid =
    c(
      0.1, 0.5, 1.5, 3, 6,
      seq(from = 10, to = 80, by = 2)
    ),
  u = c(42, 30, 15, 5, 20, 25, 30, 60),
  t0 = c(0, 15, 30, 40, 55, 70, 75, 85),
  Tmax = 60
)

# Initialize the CRM model used to model the data
npiece_ <- 10
lambda_prior <- function(k) {
  npiece_ / (data@Tmax * (npiece_ - k + 0.5))
}

model <- DALogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 56,
  npiece = npiece_,
  l = as.numeric(t(apply(as.matrix(c(1:npiece_), 1, npiece_), 2, lambda_prior))),
  c_par = 2
)
```

```
# Obtain the posterior

options <- McmcOptions(
  burnin = 10,
  step = 2,
  samples = 1e2
)

set.seed(94)
samples <- mcmc(data, model, options)


# Extract the posterior mean hazard (and empirical 2.5 and 97.5 percentile)
# for the piecewise exponential model
# If hazard=FALSE, the posterior PEM will be plot
fitted <- fitPEM(
  object = samples,
  model = model,
  data = data,
  middle = mean,
  hazard = TRUE,
  quantiles = c(0.25, 0.75)
)

# nolint end
```

`FractionalCRM-class` `FractionalCRM`

Description

[Stable]

`FractionalCRM` is the class for a fractional CRM model based on a one parameter CRM (with normal prior on the log-power parameter) as well as Kaplan-Meier based estimation of the conditional probability to experience a DLT for non-complete observations.

This fractional CRM model follows the paper and code by Guosheng Yin et al.

Usage

```
FractionalCRM(...)

.DefaultFractionalCRM()
```

Arguments

`...`	Arguments passed on to `OneParLogNormalPrior`

`skel_probs` (`numeric`)
: skeleton prior probabilities. This is a vector of unique and sorted probability values between 0 and 1.

`dose_grid` (`numeric`)
: dose grid. It must be must be a sorted vector of the same length as `skel_probs`.

`sigma2` (`number`)
: prior variance of log power parameter alpha.

Note

Typically, end users will not use the `.DefaultTITELogisticLogNormal()` function.

See Also

`TITELogisticLogNormal`.

Examples

```
my_model <- FractionalCRM(
  skel_probs = c(0.1, 0.2, 0.3, 0.4),
  dose_grid = c(10, 30, 50, 100),
  sigma2 = 2
)
```

`gain` *Compute Gain Values based on Pseudo DLE and a Pseudo Efficacy Models and Using Optional Samples.*

Description

[Stable]

Usage

```
gain(dose, model_dle, samples_dle, model_eff, samples_eff, ...)

## S4 method for signature 'numeric,ModelTox,Samples,ModelEff,Samples'
gain(dose, model_dle, samples_dle, model_eff, samples_eff, ...)

## S4 method for signature 'numeric,ModelTox,missing,Effloglog,missing'
gain(dose, model_dle, samples_dle, model_eff, samples_eff, ...)
```

Arguments

`dose` (`number` or `numeric`)
: the dose which is targeted. The following recycling rule applies when samples are not missing: vectors of size 1 will be recycled to the size of the sample. Otherwise, `dose` must have the same size as the sample.

model_dle	(ModelTox) pseudo DLE (dose-limiting events)/toxicity model.
samples_dle	(Samples) the samples of model's parameters that will be used to compute toxicity probabilities. Can also be missing for some models.
model_eff	(ModelEff) the efficacy model with pseudo data prior.
samples_eff	(Samples) samples of model's parameters that will be used to compute expected efficacy values. Can also be missing for some models.
...	not used.

Details

This function computes the gain values for a given dose level, pseudo DLE and Efficacy models as well as a given DLE and Efficacy samples.

Value

The gain values.

Functions

- gain(dose = numeric, model_dle = ModelTox, samples_dle = Samples, model_eff = ModelEff, samples_eff = Samples):
- gain(dose = numeric, model_dle = ModelTox, samples_dle = missing, model_eff = Effloglog, samples_eff = missing): Compute the gain value for a given dose level, pseudo DLE and Efficacy models without DLE and the Efficacy samples.

Examples

```
# Obtain the gain value for a given dose, a pseudo DLE and efficacy models
# as well as DLE and efficacy samples.
emptydata <- DataDual(doseGrid = seq(25, 300, 25), placebo = FALSE)
mcmc_opts <- McmcOptions(burnin = 100, step = 2, samples = 200)

# DLE model and samples.
model_dle <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = emptydata
)

samples_dle <- mcmc(emptydata, model_dle, mcmc_opts)

# Efficacy model (Effloglog) and samples.
model_effloglog <- Effloglog(
  eff = c(1.223, 2.513),
```

```
  eff_dose = c(25, 300),
  nu = c(a = 1, b = 0.025),
  data = emptydata
)

samples_effloglog <- mcmc(emptydata, model_effloglog, mcmc_opts)

# Gain values for dose level 75 and Effloglog efficacy model.
gain(
  dose = 75,
  model_dle = model_dle,
  samples_dle = samples_dle,
  model_eff = model_effloglog,
  samples_eff = samples_effloglog
)

# Efficacy model (EffFlexi) and samples.
model_effflexi <- EffFlexi(
  eff = c(1.223, 2.513),
  eff_dose = c(25, 300),
  sigma2W = c(a = 0.1, b = 0.1),
  sigma2betaW = c(a = 20, b = 50),
  rw1 = FALSE,
  data = emptydata
)

samples_effflexi <- mcmc(emptydata, model_effflexi, mcmc_opts)

# Gain values for dose level 75 and EffFlexi efficacy model.
gain(
  dose = 75,
  model_dle = model_dle,
  samples_dle = samples_dle,
  model_eff = model_effflexi,
  samples_eff = samples_effflexi
)

# Obtain the gain value for a given dose, a pseudo DLE and efficacy models
# without DLE and efficacy samples.
emptydata <- DataDual(doseGrid = seq(25, 300, 25), placebo = FALSE)
data <- Data(doseGrid = seq(25, 300, 25), placebo = FALSE)
mcmc_opts <- McmcOptions(burnin = 100, step = 2, samples = 200)

# DLE model and samples.
model_dle <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = data
)

# Efficacy model and samples.
model_eff <- Effloglog(
```

```
  eff = c(1.223, 2.513),
  eff_dose = c(25, 300),
  nu = c(a = 1, b = 0.025),
  data = emptydata
)

# Gain value for dose level 75.
gain(
  dose = 75,
  model_dle = model_dle,
  model_eff = model_eff
)
```

`GeneralData-class` `GeneralData`

Description

[Stable]

`GeneralData` is a class for general data input.

Usage

```
.DefaultDataGeneral()
```

Slots

`ID (integer)`
: unique patient IDs.

`cohort (integer)`
: the cohort (non-negative sorted) indices.

`nObs (integer)`
: number of observations, a single value.

Note

Typically, end users will not use the `.DefaultDataGeneral()` function.

GeneralModel-class GeneralModel

Description

[Stable]

GeneralModel is a general model class, from which all other specific model-like classes inherit.

Usage

```
.DefaultGeneralModel()
```

Slots

datamodel (function)
: a function representing the JAGS data model specification.

priormodel (function)
: a function representing the JAGS prior specification.

modelspecs (function)
: a function computing the list of the data model and prior model specifications that are required to be specified completely (e.g. prior parameters, reference dose, etc.), based on the data slots that are required as arguments of this function. Apart of data arguments, this function can be specified with one additional (optional) argument from_prior of type logical and length one. This from_prior flag can be used to differentiate the output of the modelspecs, as its value is taken directly from the from_prior argument of the mcmc method that invokes modelspecs function. That is, when from_prior is TRUE, then only priormodel JAGS model is used (datamodel is not used) by the mcmc, and hence modelspecs function should return all the parameters that are required by the priormodel only. If the value of from_prior is FALSE, then both JAGS models datamodel and priormodel are used in the MCMC sampler, and hence modelspecs function should return all the parameters required by both datamodel and priormodel.

init (function)
: a function computing the list of starting values for parameters required to be initialized in the MCMC sampler, based on the data slots that are required as arguments of this function.

datanames (character)
: the names of all data slots that are used by datamodel JAGS function. No other names should be specified here.

datanames_prior (character)
: the names of all data slots that are used by priormodel JAGS function. No other names should be specified here.

sample (character)
: names of all parameters from which you would like to save the MCMC samples.

Note

The datamodel must obey the convention that the data input is called exactly in the same way as in the corresponding data class. All prior distributions for parameters should be contained in the model function priormodel. The background is that this can be used to simulate from the prior distribution, before obtaining any data.

Typically, end users will not use the .DefaultGeneralModel() function.

See Also

ModelPseudo.

GeneralSimulations-class

GeneralSimulations @*description* **[Stable]** *This class captures trial simulations. Here also the random generator state before starting the simulation is saved, in order to be able to reproduce the outcome. For this just use* set.seed *with the* seed *as argument before running* simulate,Design-method.

Description

GeneralSimulations @description **[Stable]** This class captures trial simulations. Here also the random generator state before starting the simulation is saved, in order to be able to reproduce the outcome. For this just use set.seed with the seed as argument before running simulate,Design-method.

Usage

```
GeneralSimulations(data, doses, seed)

.DefaultGeneralSimulations()
```

Arguments

data	(list) see slot definition.
doses	(numeric) see slot definition.
seed	(integer) see slot definition.

Slots

data (list)
: produced Data objects.

doses (numeric)
: final dose recommendations.

seed (integer)
: random generator state before starting the simulation.

Note

Typically, end users will not use the `.DefaultGeneralSimulations()` function.

Examples

```
data <- list(
  Data(x = 1:3, y = c(0, 1, 0), doseGrid = 1:3, ID = 1L:3L, cohort = 1L:3L),
  Data(x = 4:6, y = c(0, 1, 0), doseGrid = 4:6, ID = 1L:3L, cohort = 1L:3L)
)

doses <- c(1, 2)

seed <- 123L

simulations <- GeneralSimulations(data, doses, seed)
```

GeneralSimulationsSummary-class
GeneralSimulationsSummary

Description

[Stable]

This class captures the summary of general simulations output. Note that objects should not be created by users, therefore no initialization function is provided for this class.

Usage

```
.DefaultGeneralSimulationsSummary()

.DefaultPseudoSimulationsSummary()
```

Slots

`target` (`numeric`)
: target toxicity interval

`target_dose_interval` (`numeric`)
: corresponding target dose interval

`nsim` (`integer`)
: number of simulations

`prop_dlts` (`ANY`)
: A numeric array (multi-dimensional) or list representing proportions of DLTs in the trials

`mean_tox_risk` (`numeric`)
: mean toxicity risks for the patients

`dose_selected` (`numeric`)
: doses selected as MTD

`tox_at_doses_selected` (numeric)
: true toxicity at doses selected

`prop_at_target` (numeric)
: Proportion of trials selecting target MTD

`dose_most_selected` (numeric)
: dose most often selected as MTD

`obs_tox_rate_at_dose_most_selected` (numeric)
: observed toxicity rate at dose most often selected

`n_obs` (ANY)
: A numeric array (multi-dimensional) or list representing number of patients overall.

`n_above_target` (integer)
: number of patients treated above target tox interval

`dose_grid` (numeric)
: the dose grid that has been used

`placebo` (logical)
: set to TRUE (default is FALSE) for a design with placebo

Note

Typically, end users will not use the `.DefaultGeneralSimulationsSummary()` function.

Typically, end users will not use the `.DefaultPseudoSimulationsSummary()` function.

`get,Samples,character-method`
Get specific parameter samples and produce a data.frame

Description

Here you have to specify with `pos` which parameter you would like to extract from the `Samples` object

Usage

```
## S4 method for signature 'Samples,character'
get(x, pos = -1L, envir = NULL, mode = NULL, inherits = NULL)
```

Arguments

`x`	the `Samples` object
`pos`	the name of the parameter
`envir`	for vectorial parameters, you can give the indices of the elements you would like to extract. If `NULL`, the whole vector samples will be returned
`mode`	not used
`inherits`	not used

Value

the data frame suitable for use with ggmcmc

Examples

```
# nolint start

# Create some data
data <- Data(x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
             y = c(0, 0, 0, 0, 0, 0, 1, 0),
             cohort = c(0, 1, 2, 3, 4, 5, 5, 5),
             doseGrid = c(0.1, 0.5, 1.5, 3, 6,
                          seq(from = 10, to = 80, by=2)))

# Initialize a model
model <- LogisticLogNormal(mean = c(-0.85, 1),
                           cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
                           ref_dose = 56)

# Get posterior for all model parameters
options <- McmcOptions(burnin = 100,
                       step = 2,
                       samples = 2000)
set.seed(94)
samples <- mcmc(data, model, options)

# now extract the alpha0 samples (intercept of the regression model)
alpha0samples <- get(samples, "alpha0")

# nolint end
```

---

getEff *Extracting Efficacy Responses for Subjects Categorized by the DLT*

Description

[Stable]

A method that extracts efficacy responses for subjects and categorizes it with respect to DLT, i.e. DLT or no DLT. The efficacy responses are reported together with their corresponding dose levels.

Usage

```
getEff(object, ...)

## S4 method for signature 'DataDual'
getEff(object, no_dlt = FALSE)
```

Arguments

object	(DataDual) object from which the responses and dose levels are extracted.
...	further arguments passed to class-specific methods.
no_dlt	(flag) should only no DLT responses be returned? Otherwise, all responses are returned.

Value

list with efficacy responses categorized by the DLT value.

Examples

```
# Example data.
data <- DataDual(
  x = c(25, 50, 25, 50, 75, 300, 250, 150),
  y = c(0, 0, 0, 0, 0, 1, 1, 0),
  w = c(0.31, 0.42, 0.59, 0.45, 0.6, 0.7, 0.6, 0.52),
  doseGrid = seq(25, 300, 25)
)

# Get the efficacy response and their corresponding dose levels
# categorized by the DLT.
getEff(data)
```

---

h_all_equivalent *Comparison with Numerical Tolerance and Without Name Comparison*

Description

[Experimental]

This helper function ensures a default tolerance level equal to 1e-10, and ignores names and other attributes. In contrast to all.equal(), it always returns a logical type object.

Usage

```
h_all_equivalent(target, current, tolerance = 1e-10)
```

Arguments

target	(numeric) target values.
current	(numeric) current values.
tolerance	(number) relative differences smaller than this are not reported.

Value

TRUE when target and current do not differ up to desired tolerance and without looking at names or other attributes, FALSE otherwise.

---

h_blind_plot_data *Helper Function to Blind Plot Data*

Description

Helper Function to Blind Plot Data

Usage

```
h_blind_plot_data(df, blind, has_placebo, pbo_dose)
```

Arguments

df	(GeneralData) The data to be blinded
blind	(flag) Should the data be blinded?
has_placebo	(flag) Does the data contain a placebo dose?
pbo_dose	(positive_number) The dose to be taken as placebo. Ignored if has_placebo is FALSE

Value

The blinded data

---

h_calc_report_label_percentage

Helper function to calculate percentage of true stopping rules for report label output calculates true column means and converts output into percentages before combining the output with the report label; output is passed to show() *and output with cat to console*

Description

Helper function to calculate percentage of true stopping rules for report label output calculates true column means and converts output into percentages before combining the output with the report label; output is passed to show() and output with cat to console

Usage

```
h_calc_report_label_percentage(stop_report)
```

Arguments

stop_report	object from summary method

Value

named list with label and percentage of rule activation

---

h_check_fun_formals *Checking Formals of a Function*

Description

[Experimental]

This helper function checks whether a given function fun has required or allowed arguments. The argument check is based only on the names of the arguments. No any further logic is verified here.

Usage

```
h_check_fun_formals(fun, mandatory = NULL, allowed = NULL)
```

Arguments

fun	(function) a function name whose argument names will be checked.
mandatory	(character or NULL) the names of the arguments which must be present in fun. If mandatory is specified as NULL (default) this requirement is ignored.
allowed	(character or NULL) the names of the arguments which are allowed in fun. Names that do not belong to allowed are simply not allowed. The allowed parameter is independent from the mandatory, in a sense that if mandatory is specified as a character vector, it does not have to be repeated in allowed. If allowed is specified as NULL (default), then it means that there must be no any arguments in fun (except these ones which are specified in mandatory).

h_convert_ordinal_data

Convert a Ordinal Data to the Equivalent Binary Data for a Specific Grade

Description

[Experimental]

A simple helper function that takes a `DataOrdinal` object and an integer grade and converts them to the equivalent `Data` object.

Usage

```
h_convert_ordinal_data(data_ord, grade)
```

Arguments

`data_ord`	(`DataOrdinal`) the `DataOrdinal` object to covert
`grade`	(`integer`) the toxicity grade for which the equivalent data is required.

Value

A `Data` object.

h_convert_ordinal_model

Convert an ordinal CRM model to the Equivalent Binary CRM Model for a Specific Grade

Description

[Experimental]

A simple helper function that takes a `LogisticLogNormalOrdinal` and an integer grade and converts them to the equivalent `LogisticLogNormal` model.

Usage

```
h_convert_ordinal_model(x, grade)
```

Arguments

x	(LogisticLogNormalOrdinal) the LogisticLogNormalOrdinal model to covert
grade	(integer) the toxicity grade for which the equivalent model is required.

Value

A LogisticLogNormal model.

---

h_convert_ordinal_samples

Convert a Samples Object from an ordinal Model to the Equivalent Samples Object from a Binary Model

---

Description

[Experimental]

A simple helper function that converts a Samples object from the fit of an ordinal CRM model to that which would have been obtained from fitting a binary CRM model for toxicities of a specified grade to the same observed data.

Usage

```
h_convert_ordinal_samples(obj, grade)
```

Arguments

obj	(Samples) the Samples object to covert
grade	(integer) the toxicity grade for which the equivalent data is required.

Value

A Samples object.

h_default_if_empty *Getting the default value for an empty object*

Description

[Stable]

A simple helper function that sets a default value for an empty or missing object, that is an object for which `length()` function returns `0L` or it has length 1 and `is.na()` returns `TRUE`.

Usage

```
h_default_if_empty(x, default)
```

Arguments

`x`	(any) an object to handle. It can be any object for which `length()` function is defined.
`default`	(any) a default value for `x` object.

Examples

```
h_default_if_empty(character(0), default = "default label")
h_default_if_empty("custom label", default = "default label")
h_default_if_empty(NA, default = "default label")
```

h_find_interval *Find Interval Numbers or Indices and Return Custom Number For 0.*

Description

[Stable]

A simple wrapper of `findInterval()` function that invokes `findInterval()`, takes its output and replaces all the elements with 0 value to a custom number as specified in `replacement` argument.

Usage

```
h_find_interval(..., replacement = -Inf)
```

Arguments

...	Arguments passed on to base::findInterval x numeric. vec numeric, sorted (weakly) increasingly, of length N, say. rightmost.closed logical; if true, the rightmost interval, vec[N-1] .. vec[N] is treated as *closed*, see below. all.inside logical; if true, the returned indices are coerced into 1,...,N-1, i.e., 0 is mapped to 1 and N to N-1. left.open logical; if true all the intervals are open at left and closed at right; in the formulas below, $\leq$ should be swapped with $<$ (and $>$ with $\geq$), and rightmost.closed means 'leftmost is closed'. This may be useful, e.g., in survival analysis computations.
replacement	(number) a custom number to be used as a replacement for 0. Default to -Inf.

Examples

```
h_find_interval(1, c(2, 4, 6))
h_find_interval(3, c(2, 4, 6))
h_find_interval(1, c(2, 4, 6), replacement = -1)
```

---

h_format_number *Conditional Formatting Using C-style Formats*

Description

[Experimental]

This helper function conditionally formats a number with formatC() function using "E" format and specific number of digits as given by the user. A number is formatted if and only if its absolute value is less than 0.001 or greater than 10000. Otherwise, the number is not formatted. Additionally, custom prefix or suffix can be appended to character string with formatted number, so that the changes are marked.

Usage

```
h_format_number(x, digits = 5, prefix = "", suffix = "")
```

Arguments

x	(number) a number to be formatted.
digits	(function) the desired number of significant digits.
prefix	(string) a prefix to be added in front of the formatted number.
suffix	(string) a suffix to be appended after the formatted number.

Value

Either formatted x as string or unchanged x if the formatting condition is not met.

Note

This function was primarily designed as a helper for h_jags_write_model() function.

Examples

```
h_format_number(50000)
h_format_number(50000, prefix = "P", suffix = "S")
```

---

h_info_theory_dist *Calculating the Information Theoretic Distance*

Description

[Experimental]

Helper function which provides the value of the divergence as given by equation in (7) in the reference at https://doi.org/10.1002/sim.8450.

Usage

```
h_info_theory_dist(prob, target, asymmetry)
```

Arguments

prob	(numeric) vector or matrix with probabilities of a DLT occurring.
target	(number) single target probability of a DLT.
asymmetry	(number) describes the rate of penalization for overly toxic does, range 0 to 2.

Examples

```
h_info_theory_dist(c(0.5, 0.2), 0.4, 1.2)
```

h_in_range *Check which elements are in a given range*

Description

[Stable]

A simple helper function that tests whether elements of a given vector or matrix are within specified interval.

Usage

```
h_in_range(x, range = c(0, 1), bounds_closed = TRUE)
```

Arguments

x	(numeric) vector or matrix with elements to test.
range	(numeric) an interval, i.e. sorted two-elements vector.
bounds_closed	(logical) should bounds in the range be treated as closed? This can be a scalar or vector of length two. If it is a scalar, then its value applies to lower bound range[1] and upper bound range[2]. If this is a vector with two flags, the first flag corresponds to the lower bound only, and the second to the upper bound only.

Value

A logical vector or matrix of length equal to the length of x, that for every element of x, indicates whether a given element of x is in the range.

Examples

```
x <- 1:4
h_in_range(x, range = c(1, 3))
h_in_range(x, range = c(1, 3), bounds_closed = FALSE)
h_in_range(x, range = c(1, 3), bounds_closed = c(FALSE, TRUE))
mat <- matrix(c(2, 5, 3, 10, 4, 9, 1, 8, 7), nrow = 3)
h_in_range(mat, range = c(1, 5))
```

h_is_positive_definite *Testing Matrix for Positive Definiteness*

Description

[Experimental]

This helper function checks whether a given numerical matrix x is a positive-definite square matrix of a given size, without any missing values. This function is used to test if a given matrix is a covariance matrix, since every symmetric positive semi-definite matrix is a covariance matrix.

Usage

```
h_is_positive_definite(x, size = 2, tol = 1e-08)
```

Arguments

x	(matrix) a matrix to be checked.
size	(integer) a size of the square matrix x to be checked against for.
tol	(number) a given tolerance number used to check whether an eigenvalue is positive or not. An eigenvalue is considered as positive if and only if it is greater than the tol.

Details

The positive definiteness test implemented in this function is based on the following characterization valid for real matrices: A symmetric matrix is positive-definite if and only if all of its eigenvalues are posi
In this function an eigenvalue is considered as positive if and only if it is greater than the tol.

Value

TRUE if a given matrix is a positive-definite, FALSE otherwise.

h_jags_add_dummy *Appending a Dummy Number for Selected Slots in Data*

Description

[Experimental]

A helper function that appends a dummy value to a given slots in GeneralData class object, if and only if the total number of observations (as indicated by object@nObs) equals to 1. Otherwise, the object is not changed.

Usage

```
h_jags_add_dummy(object, where, dummy = 0)
```

Arguments

object	(GeneralData) object into which dummy values will be added.
where	(character) names of slots in object to which a dummy number will be appended.
dummy	(number) a dummy number that will be appended to selected slots in object. Default to 0.

Value

A GeneralData object with slots updated with dummy number.

Note

The main motivation behind this function is related to the JAGS. If there is only one observation, the data is not passed correctly to JAGS, i.e. e.g. x and y are treated like scalars in the data file. Therefore it is necessary to add dummy values to the vectors in this case As we don't change the number of observations (nObs), this addition of zeros doesn't affect the results of JAGS computations.

Examples

```
# Create some data of class 'Data'
my_data <- Data(
  x = 0.1,
  y = 0,
  doseGrid = c(0.1, 0.5)
)

my_data_2 <- Data(
  x = c(0.1, 0.5),
  y = c(0, 1),
  doseGrid = c(0.1, 0.5)
)

# Append dummy to `x` and `y`.
h_jags_add_dummy(my_data, where = c("x", "y"))

# Append dummy to `x` and `y`. No effect as `my_data_2@nObs != 1`.
h_jags_add_dummy(my_data_2, where = c("x", "y"))
```

h_jags_extract_samples *Extracting Samples from* JAGS mcarray *Object*

Description

[Stable]

A simple helper function that extracts a sample from rjags::mcarray.object S3 class object. The rjags::mcarray.object object is used by the rjags::jags.samples() function to represent MCMC output from a JAGS model.

Usage

```
h_jags_extract_samples(x)
```

Arguments

x	an rjags::mcarray.object object.

h_jags_get_data *Getting Data for* JAGS

Description

[Experimental]

A simple helper function that prepares an object for data argument of rjags::jags.model(), which is invoked by mcmc() method.

Usage

```
h_jags_get_data(model, data, from_prior)
```

Arguments

model	(GeneralModel) an input model.
data	(GeneralData) an input data.
from_prior	(flag) sample from the prior only? In this case data will not be appended to the output, i.e. only the variables required by the model@priormodel model will be returned in data.

Examples

```
# Create some data from the class `Data`.
my_data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)

# Initialize the CRM model.
my_model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 56
)

jags_data <- h_jags_get_data(my_model, my_data, from_prior = FALSE)
jags_data
```

---

h_jags_get_model_inits

Setting Initial Values for JAGS *Model Parameters*

Description

[Experimental]

A simple helper function that prepares an object for `inits` argument of `rjags::jags.model()`, which is invoked by `mcmc()` method. The `inits` argument specifies initial values for model parameters.

Usage

```
h_jags_get_model_inits(model, data)
```

Arguments

`model`	(`GeneralModel`) an input model.
`data`	(`GeneralData`) an input data.

Value

A `list` of starting values for parameters required to be initialized in the MCMC JAGS sampler.

Examples

```
# Create some data from the class `Data`.
my_data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)

# Initialize the CRM model.
my_model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 56
)

h_jags_get_model_inits(model = my_model, data = my_data)
```

---

h_jags_join_models *Joining* JAGS *Models*

Description

[Stable]

This helper function joins two JAGS models in the way that the body of the second model is appended to the body of the first model (in this order). After that, the first, body-extended model is returned. The arguments of `model1`, `model2` model functions (if any) are not combined in any way.

Usage

```
h_jags_join_models(model1, model2)
```

Arguments

`model1`	(`function`) the first model to join.
`model2`	(`function`) the second model to join.

Value

joined models.

Note

`model1` and `model2` functions must have a multi-expression body, i.e. braced expression(s). Environments or any attributes of the function bodies are not preserved in any way after joining.

h_jags_write_model *Writing JAGS Model to a File*

Description

[Stable]

This function converts a R function with JAGS model into a text and then writes it into a given file. During the "model into text" conversion, the format of numbers of which absolute value is less than `0.001` or greater than `10000` is changed. These numbers will be converted into scientific format with specified number of significant digits using `formatC()` function.

Usage

```
h_jags_write_model(model, file = NULL, digits = 5)
```

Arguments

`model`	(`function`) function containing the JAGS model.
`file`	(`string` or `NULL`) the name of the file (including the optional path) where the model will be saved. If `NULL`, the file will be created in a `R_crmPack` folder placed under temporary directory indicated by `tempdir()` function.
`digits`	(`count`) a desired number of significant digits for for numbers used in JAGS input, see `formatC()`.

Value

The name of the file where the model was saved.

Note

JAGS syntax allows truncation specification like `dnorm(...) I(...)`, which is illegal in R. To overcome this incompatibility, use dummy operator `\%_\%` before `I(...)`, i.e. `dnorm(...) \%_\% I(...)` in the model's code. This dummy operator `\%_\%` will be removed just before saving the JAGS code into a file. Due to technical issues related to conversion of numbers to scientific format, it is required that the body of a model function does not contain `TEMP_NUM_PREF_` or `_TEMP_NUM_SUF` character constants in its body.

Examples

```
# Some model function
my_model <- function() {
  alpha0 <- mean(1:10)
  alpha1 <- 600000
}
```

```
h_jags_write_model(my_model, digits = 5)
```

`h_model_dual_endpoint_beta` *Update certain components of* `DualEndpoint` *model with regard to parameters of the function that models dose-biomarker relationship defined in the* `DualEndpointBeta` *class.*

Description

[Stable]

A simple helper function that takes `DualEndpoint` object and updates `use_fixed`, `priormodel`, `modelspecs`, `init`, `sample` slots with regard to a given parameter of the dose-biomarker relationship $f(x)$ defined in the `DualEndpointBeta` class. This update solely depends on whether a given parameter's value `param` is a fixed-valued scalar or two-elements numeric vector. In the later case, it is assumed that `param` represents two parameters of a probability distribution that will be used in `priormodel` function to generate values for the `param_name` parameter of $f(x)$. See the help page for `DualEndpointBeta` class for more details.

Usage

```
h_model_dual_endpoint_beta(
  param,
  param_name,
  param_suffix = c("_low", "_high"),
  priormodel = NULL,
  de
)
```

Arguments

`param`	(`numeric`) the value of a given `param_name` parameter of the dose-biomarker relationship function $f(x)$. Either a fixed-valued scalar or vector with two elements that are the parameters of a probability distribution that will be used in `priormodel` function to generate values for the `param_name` parameter of $f(x)$.
`param_name`	(`string`) the name of the parameter of $f(x)$, whose value depends on `param`.
`param_suffix`	(`character`) the two suffixes to be appended to the elements of `param_name` and then used when updating `modelspecs`. The value of this argument is ignored when `param` is a scalar.
`priormodel`	(`function` or `NULL`) a function representing the `JAGS` prior specification that will be appended to existing `de@priormodel` specification if `param` is not a scalar. Otherwise, `de@priormodel` remains unchanged.

de (DualEnpoint)
dual endpoint model whose slots will be updated.

Value

A DualEndpoint model with updated use_fixed, priormodel, modelspecs, init, sample slots.

---

h_model_dual_endpoint_rho *Update DualEndpoint class model components with regard to DLT and biomarker correlation.*

Description

[Stable]

A simple helper function that takes DualEndpoint model existing components (priormodel, modelspecs, init, sample), and updates them with regard to DLT and biomarker correlation rho.

Usage

```
h_model_dual_endpoint_rho(use_fixed, rho, comp)
```

Arguments

use_fixed (flag)
indicates whether a fixed value for DLT and biomarker correlation rho should be used or not. If rho is not supposed to be a fixed value, a prior distribution from the scaled Beta family will be used. See the details below, under rho argument.

rho (numeric)
DLT and biomarker correlation. It must be either a fixed value (between -1 and 1), or a named vector with two elements, named a and b for the Beta prior on the transformation kappa = (rho + 1) / 2, which is in (0, 1). For example, a = 1, b = 1 leads to a uniform prior on rho.

comp (list)
a named list with model components that will be updated. The names should be: priormodel, modelspecs, init, sample. For definitions of the components, see GeneralModel class. The modelspecs and init components on comp list are specified up to the body of corresponding GeneralModel@modelspecs and GeneralModel@init functions. These bodies are simply a lists itself.

Value

A list with updated model components.

h_model_dual_endpoint_sigma2betaW *Update certain components of* DualEndpoint *model with regard to prior variance factor of the random walk.*

Description

[Stable]

A simple helper function that takes DualEndpoint object and updates priormodel, modelspecs, init, sample slots according to the random walk variance.

Usage

```
h_model_dual_endpoint_sigma2betaW(use_fixed, sigma2betaW, de)
```

Arguments

use_fixed	(flag) indicates whether a fixed value for sigma2betaW should be used or not. If sigma2betaW is not supposed to be a fixed value, a prior distribution from the Inverse-Gamma distribution will be used. See the details below, under sigma2betaW argument.
sigma2betaW	(numeric) the prior variance factor of the random walk prior for the biomarker model. Either a fixed value or Inverse-Gamma distribution parameters, i.e. vector with two elements named a and b.
de	(DualEnpoint) dual endpoint model whose slots will be updated.

Value

A DualEndpoint model with updated priormodel, modelspecs, init, sample slots.

See Also

DualEndpointRW.

`h_model_dual_endpoint_sigma2W`

Update `DualEndpoint` *class model components with regard to biomarker regression variance.*

Description

[Stable]

A simple helper function that takes `DualEndpoint` model existing components (`priormodel`, `modelspecs`, `init`, `sample`), and updates them with regard to to biomarker regression variance `sigma2W`.

Usage

```
h_model_dual_endpoint_sigma2W(use_fixed, sigma2W, comp)
```

Arguments

`use_fixed`	(`flag`) indicates whether a fixed value for the biomarker regression variance `sigma2W` should be used or not. If `sigma2W` is not supposed to be a fixed value, a prior distribution from the Inverse-Gamma distribution will be used. See the details below, under `sigma2W` argument.
`sigma2W`	(`numeric`) the biomarker variance. Either a fixed value or Inverse-Gamma distribution parameters, i.e. vector with two elements named `a` and `b`.
`comp`	(`list`) a named list with model components that will be updated. The names should be: `priormodel`, `modelspecs`, `init`, `sample`. For definitions of the components, see `GeneralModel` class. The `modelspecs` and `init` components on `comp` list are specified up to the body of corresponding `GeneralModel@modelspecs` and `GeneralModel@init` functions. These bodies are simply a lists itself.

Value

`list` with updated model components.

`h_next_best_eligible_doses`

Get Eligible Doses from the Dose Grid.

Description

[Experimental]

Helper function that gets the eligible doses from the dose grid. The eligible doses are the doses which do not exceed a given `doselimit`. For placebo design, if safety allows (i.e. if there is at least one non-placebo dose which does not exceed the dose limit), the placebo dose it then excluded from the eligible doses.

Usage

```
h_next_best_eligible_doses(dose_grid, doselimit, placebo, levels = FALSE)
```

Arguments

`dose_grid`	(`numeric`) all possible doses.
`doselimit`	(`number`) the maximum allowed next dose.
`placebo`	(`flag`) if `TRUE` the first dose level in the `dose_grid` is considered as placebo.
`levels`	(`flag`) if `TRUE` the levels of eligible doses are returned, otherwise, the doses (default).

Value

A numeric vector with eligible doses or eligible dose levels if `levels` flag is `TRUE`.

Examples

```
dose_grid <- c(0.001, seq(25, 200, 25))
h_next_best_eligible_doses(dose_grid, 79, TRUE)
h_next_best_eligible_doses(dose_grid, 24, TRUE)
```

---

h_next_best_mgsamples_plot

Building the Plot for `nextBest-NextBestMaxGainSamples` *Method.*

Description

[Experimental]

Helper function which creates the plot for `nextBest-NextBestMaxGainSamples()` method.

Usage

```
h_next_best_mgsamples_plot(
  prob_target_drt,
  dose_target_drt,
  prob_target_eot,
  dose_target_eot,
  dose_mg,
  dose_mg_samples,
  next_dose,
  doselimit,
  dose_grid_range
)
```

Arguments

`prob_target_drt`
(`proportion`)
target DLT probability during the trial.

`dose_target_drt`
(`number`)
target dose estimate during the trial.

`prob_target_eot`
(`proportion`)
target DLT probability at the end of the trial.

`dose_target_eot`
(`number`)
target dose estimate at the end of the trial.

`dose_mg` (`number`)
the dose corresponding to the maximum gain.

`dose_mg_samples`
(`numeric`)
for every sample, the dose (from the dose grid) that gives the maximum gain value.

`next_dose` (`number`)
next best dose.

`doselimit` (`number`)
the maximum allowed next dose.

`dose_grid_range`
(`numeric`)
dose grid range.

h_next_best_mg_ci *Credibility Intervals for Max Gain and Target Doses at* nextBest-NextBestMaxGain *Method.*

Description

[Experimental]

Helper function for nextBest-NextBestMaxGain() method. It computes a 95% credibility intervals for given target dose and max gain dose. It also returns a ratio of upper and lower bounds of the interval.

Usage

```
h_next_best_mg_ci(dose_target, dose_mg, prob_target, placebo, model, model_eff)
```

Arguments

dose_target	(number) target dose estimate.
dose_mg	(number) the dose corresponding to the maximum gain.
prob_target	(proportion) target DLT probability.
placebo	(flag) if TRUE the first dose level in the dose grid used is considered as placebo. This is needed to adjust the max gain dose using efficacy constant value. If the placebo was used, then the model_eff@const is added to dose_mg.
model	(ModelTox) the DLT model.
model_eff	(Effloglog) the efficacy model.

References

Yeung, W.Y., Whitehead, J., Reigner, B., Beyer, U., Diack, Ch., Jaki, T. (2015), Bayesian adaptive dose-escalation procedures for binary and continuous responses utilizing a gain function, *Pharmaceutical Statistics*, doi:10.1002/pst.1706

h_next_best_mg_doses_at_grid *Get Closest Grid Doses for a Given Target Doses for* nextBest-NextBestMaxGain *Method.*

Description

[Experimental]

Helper function that for a given target doses finds the dose in grid that is closest and below the target. There are four different targets in the context of nextBest-NextBestMaxGain() method: $min(`dose_mg`, `dose_target_drt`)$, dose_mg, dose_target_drt or dose_target_eot.

Usage

```
h_next_best_mg_doses_at_grid(
  dose_target_drt,
  dose_target_eot,
  dose_mg,
  dose_grid,
  doselimit,
  placebo
)
```

Arguments

dose_target_drt	(number) target dose estimate during the trial.
dose_target_eot	(number) target dose estimate at the end of the trial.
dose_mg	(number) the dose corresponding to the maximum gain.
dose_grid	(numeric) all possible doses.
doselimit	(number) the maximum allowed next dose.
placebo	(flag) if TRUE the first dose level in the dose_grid is considered as placebo.

`h_next_best_mg_plot` *Building the Plot for* `nextBest-NextBestMaxGain` *Method.*

Description

[Experimental]

Helper function which creates the plot for `nextBest-NextBestMaxGain()` method.

Usage

```
h_next_best_mg_plot(
  prob_target_drt,
  dose_target_drt,
  prob_target_eot,
  dose_target_eot,
  dose_mg,
  max_gain,
  next_dose,
  doselimit,
  data,
  model,
  model_eff
)
```

Arguments

`prob_target_drt` (`proportion`)
target DLT probability during the trial.

`dose_target_drt` (`number`)
target dose estimate during the trial.

`prob_target_eot` (`proportion`)
target DLT probability at the end of the trial.

`dose_target_eot` (`number`)
target dose estimate at the end of the trial.

`dose_mg` (`number`)
the dose corresponding to the maximum gain.

`max_gain` (`number`)
the maximum gain estimate.

`next_dose` (`number`)
next best dose.

doselimit (number)
the maximum allowed next dose.

data (DataDual)
the data object from which the dose grid will be fetched.

model (ModelTox)
the DLT model.

model_eff (Effloglog)
the efficacy model.

---

h_next_best_ncrm_loss_plot *Building the Plot for* nextBest-NextBestNCRMLoss *Method.*

Description

[Experimental]

Helper function which creates the plot for nextBest-NextBestNCRMLoss() method.

Usage

```
h_next_best_ncrm_loss_plot(
  prob_mat,
  posterior_loss,
  max_overdose_prob,
  dose_grid,
  max_eligible_dose_level,
  doselimit,
  next_dose,
  is_unacceptable_specified
)
```

Arguments

prob_mat (numeric)
matrix with probabilities of a grid doses to be in a given interval. If is_unacceptable_specified is TRUE, there must be 4 intervals (columns) in prob_mat: underdosing, target, excessive, unacceptable. Otherwise, there must be 3 intervals (columns): underdosing, target, overdose. Number of rows must be equal to number of doses in a grid.

posterior_loss (numeric)
posterior losses.

max_overdose_prob (number)
maximum overdose posterior probability that is allowed.

`dose_grid` (`numeric`)
dose grid.

`max_eligible_dose_level` (`number`)
maximum eligible dose level in the `dose_grid`.

`doselimit` (`number`)
the maximum allowed next dose.

`next_dose` (`number`)
next best dose.

`is_unacceptable_specified` (`flag`)
is unacceptable interval specified?

`h_next_best_tdsamples_plot` *Building the Plot for* `nextBest-NextBestTDsamples` *Method.*

Description

[Experimental]

Helper function which creates the plot for `nextBest-NextBestTDsamples()` method.

Usage

```
h_next_best_tdsamples_plot(
  dose_target_drt_samples,
  dose_target_eot_samples,
  dose_target_drt,
  dose_target_eot,
  dose_grid_range,
  nextBest,
  doselimit,
  next_dose
)
```

Arguments

`dose_target_drt_samples` (`numeric`)
vector of in-trial samples.

`dose_target_eot_samples` (`numeric`)
vector of end-of-trial samples.

`dose_target_drt` (`number`)
target in-trial estimate.

dose_target_eot
: (number)
target end-of-trial estimate.

dose_grid_range
: (numeric)
range of dose grid.

nextBest
: (NextBestTDsamples)
the rule for the next best dose.

doselimit
: (number)
the maximum allowed next dose.

next_dose
: (number)
next best dose.

---

h_next_best_td_plot *Building the Plot for* nextBest-NextBestTD *Method.*

Description

[Experimental]

Helper function which creates the plot for nextBest-NextBestTD() method.

Usage

```
h_next_best_td_plot(
  prob_target_drt,
  dose_target_drt,
  prob_target_eot,
  dose_target_eot,
  data,
  prob_dlt,
  doselimit,
  next_dose
)
```

Arguments

prob_target_drt
: (proportion)
target DLT probability during the trial.

dose_target_drt
: (number)
target dose estimate during the trial.

prob_target_eot
: (proportion)
target DLT probability at the end of the trial.

dose_target_eot
: (number)
target dose estimate at the end of the trial.

data
: (Data)
the data object from which the dose grid will be fetched.

prob_dlt
: (numeric)
DLT probabilities for doses in grid.

doselimit
: (number)
the maximum allowed next dose.

next_dose
: (number)
next best dose.

`h_null_if_na` *Getting* `NULL` *for* `NA`

Description

[Stable]

A simple helper function that replaces `NA` object by `NULL` object.

Usage

```
h_null_if_na(x)
```

Arguments

x
: (any)
atomic object of length `1`. For the definition of "atomic", see `is.atomic()`.

Value

`NULL` if `x` is `NA`, otherwise, `x`.

Examples

```
h_null_if_na(NA)
```

h_obtain_dose_grid_range

Helper Function Containing Common Functionality

Description

Used by dose_grid_range-Data and dose_grid_range-DataOrdinal

Usage

```
h_obtain_dose_grid_range(object, ignore_placebo)
```

Arguments

object	(Data or DataOrdinal) the object for which the dose grid range is required
ignore_placebo	(flag) should placebo dose (if any) not be counted?

h_plot_data_cohort_lines

Preparing Cohort Lines for Data Plot

Description

[Experimental]

This helper function prepares a ggplot geom with reference lines separating different cohorts on the plot of Data class object. Lines are either vertical or horizontal of green color and longdash type.

Usage

```
h_plot_data_cohort_lines(cohort, placebo, vertical = TRUE)
```

Arguments

cohort	(integer) the cohort indices.
placebo	(flag) is placebo included in the doses? If it so, this function returns NULL object as in this case all doses in a given cohort are equal and there is no need to separate them.
vertical	(flag) should the line be vertical? Otherwise it is horizontal.

Details

The geom object is returned if and only if `placebo` is equal to `TRUE` and there are more than one unique values in `cohort`. Otherwise, this function returns `NULL` object.

`h_plot_data_dataordinal` *Helper Function for the Plot Method of the Data and DataOrdinal Classes*

Description

[Stable]

A method that creates a plot for `Data` and `DataOrdinal` objects.

[Stable]

A method that creates a plot for `Data` object.

[Experimental]

A method that creates a plot for `DataOrdinal` object.

Usage

```
h_plot_data_dataordinal(
  x,
  blind = FALSE,
  legend = TRUE,
  tox_labels = c(Yes = "red", No = "black"),
  tox_shapes = c(Yes = 17L, No = 16L),
  ...
)

## S4 method for signature 'Data,missing'
plot(x, y, blind = FALSE, legend = TRUE, ...)

## S4 method for signature 'DataOrdinal,missing'
plot(
  x,
  y,
  blind = FALSE,
  legend = TRUE,
  tox_labels = NULL,
  tox_shapes = NULL,
  ...
)
```

Arguments

x	(DataOrdinal) object we want to plot.
blind	(flag) indicates whether to blind the data. If TRUE, then placebo subjects are reported at the same level as the active dose level in the corresponding cohort, and DLTs are always assigned to the first subjects in a cohort.
legend	(flag) whether the legend should be added.
tox_labels	(named list of character) the labels of the toxicity categories.
tox_shapes	(names list of integers) the symbols used to identify the toxicity categories.
...	not used.
y	(missing) missing object, for compatibility with the generic function.

Value

The ggplot2 object.

The ggplot2 object.

The ggplot2 object.

Note

The default values of tox_shapes and tox_labels result in DLTs being displayed as red triangles and other responses as black circles.

With more than 9 toxicity categories, toxicity symbols must be specified manually.
With more than 5 toxicity categories, toxicity labels must be specified manually.

Examples

```
# Create some data of class 'Data'.
my_data <- Data(
  x = c(0.001, 0.1, 0.1, 0.5, 0.001, 3, 3, 0.001, 10, 10, 10),
  y = c(0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0),
  cohort = c(1, 1, 1, 2, 3, 3, 3, 4, 4, 4, 4),
  doseGrid = c(0.001, 0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2)),
  placeb = TRUE
)

# Plot the data.
plot(my_data)
data <- DataOrdinal(
  x = c(10, 20, 30, 40, 50, 50, 50, 60, 60, 60),
  y = as.integer(c(0, 0, 0, 0, 0, 1, 0, 0, 1, 2)),
  ID = 1L:10L,
```

```
  cohort = as.integer(c(1:4, 5, 5, 5, 6, 6, 6)),
  doseGrid = c(seq(from = 10, to = 100, by = 10)),
  yCategories = c("No tox" = 0L, "Sub-tox AE" = 1L, "DLT" = 2L),
  placebo = FALSE
)

plot(data)
```

---

h_plot_data_df *Preparing Data for Plotting*

Description

[Experimental]

This helper function prepares a `data.frame` object based on `Data` class object. The resulting data frame is used by the plot function for `Data` class objects.

[Experimental]

A method that transforms `GeneralData` objects into a `tibble` suitable for plotting with `ggplot2` methods

Usage

```
h_plot_data_df(data, ...)

h_plot_data_df(data, ...)

## S4 method for signature 'Data'
h_plot_data_df(data, blind = FALSE, legend = TRUE, ...)

## S4 method for signature 'DataOrdinal'
h_plot_data_df(data, blind = FALSE, legend = TRUE, ...)
```

Arguments

`data`	(`Data`) object from which data is extracted and converted into a data frame.
`...`	further arguments passed to `data.frame` constructor. It can be e.g. an extra `column_name = value` pair based on a slot from `x` (which in this case might be a subclass of `Data`) which does not appear in `Data`.
`blind`	(`flag`) should data be blinded? If `TRUE`, then for each cohort, all DLTs are assigned to the first subjects in the cohort. In addition, the placebo (if any) is set to the active dose level for that cohort.
`legend`	(`flag`) Display the legend for the toxicity categories

Value

A data.frame object with values to plot.

data.frame containing columns for patient, cohort, dose and toxicity grade

A data.frame object with columns patient, ID, cohort, dose and toxicity.

Methods (by class)

- h_plot_data_df(Data): method for Data.
- h_plot_data_df(DataOrdinal): Class specific method for DataOrdinal

---

h_rapply *Recursively Apply a Function to a List*

Description

[Experimental]

This helper function recursively iterates through a "list-like" object and it checks whether an element is of a given class. If it so, then it replaces that element by the result of an execution of a given function. Otherwise, and if the element is of length greater than 1 (i.e. not a scalar), it replaces that element by the result of h_rapply(), recursively called for that element. In the remaining case, that is, the element is not of a given class and is a scalar, then that element remains unchanged.

Usage

```
h_rapply(x, fun, classes, ...)
```

Arguments

x	(any) "list-like" object for which subsetting operator [[is defined.
fun	(function) a function of one "principal" argument, passing further arguments via ....
classes	(character) class names.
...	further arguments passed to function fun.

Value

"list-like" object of similar structure as x.

Note

This helper function is conceptually similar the same as rapply() function. However, it differs from rapply() in two major ways. First, the h_rapply() is not limited to objects of type list or expression only. It can be any "list-like" object of any type for which subsetting operator [[is defined. This can be, for example, an object of type language, often obtained from the body() function. The second difference is that the flexibility of rapply() on how the result is structured is not available with h_rapply() for the user. That is, with h_rapply() each element of x, which has a class included in classes, is replaced by the result of applying fun to the element. This behavior corresponds to rapply() when invoked with fixed how = replace. This function was primarily designed as a helper for h_jags_write_model() function.

Examples

```
# Some model function.
my_model <- function() {
  alpha0 <- mean(1:10)
  alpha1 <- 600000
}

# Replace format of numbers using `formatC` function.
h_rapply(
  x = body(my_model),
  fun = formatC,
  classes = c("integer", "numeric"),
  digits = 3,
  format = "E"
)
```

---

h_slots *Getting the Slots from a S4 Object*

Description

[Experimental]

This helper function extracts requested slots from the S4 class object. It is a simple wrapper of methods::slot() function.

Usage

```
h_slots(object, names, simplify = FALSE)
```

Arguments

object	(S4) an object from a formally defined S4 class.
names	(character) a vector with names of slots to be fetched. This function assumes that for every element in names, there exists a slot of the same name in the object.

simplify (flag)
should an output be simplified? This has an effect if and only if a single slot is about to be extracted, i.e. names is just a single string.

Value

list with the slots extracted from object according to names, or single slot if simplification is required and possible.

---

h_summarize_add_stats *Helper function to calculate average across iterations for each additional reporting parameter extracts parameter names as specified by user and averaged the values for each specified parameter to show() and output with cat to console*

Description

Helper function to calculate average across iterations for each additional reporting parameter extracts parameter names as specified by user and averaged the values for each specified parameter to show() and output with cat to console

Usage

```
h_summarize_add_stats(stats_list)
```

Arguments

stats_list object from simulation with nested parameter values (sublist for each parameter)

Value

list of parameter names and averaged values for console output

---

h_test_named_numeric *Check that an argument is a named vector of type numeric*

Description

[Stable]

A simple helper function that tests whether an object is a named numerical vector.

Usage

```
h_test_named_numeric(
  x,
  subset.of = NULL,
  must.include = NULL,
  permutation.of = NULL,
  identical.to = NULL,
  disjunct.from = NULL,
  lower = 0 + .Machine$double.xmin,
  finite = TRUE,
  any.missing = FALSE,
  len = 2,
  ...
)
```

Arguments

x	(any) object to check.
subset.of	[character] Names provided in x must be subset of the set subset.of.
must.include	[character] Names provided in x must be a superset of the set must.include.
permutation.of	[character] Names provided in x must be a permutation of the set permutation.of. Duplicated names in permutation.of are stripped out and duplicated names in x thus lead to a failed check. Use this argument instead of identical.to if the order of the names is not relevant.
identical.to	[character] Names provided in x must be identical to the vector identical.to. Use this argument instead of permutation.of if the order of the names is relevant.
disjunct.from	[character] Names provided in x must may not be present in the vector disjunct.from.
lower	[numeric(1)] Lower value all elements of x must be greater than or equal to.
finite	[logical(1)] Check for only finite values? Default is FALSE.
any.missing	[logical(1)] Are vectors with missing values allowed? Default is TRUE.
len	[integer(1)] Exact expected length of x.
...	further parameters passed to checkmate::test_numeric().

Value

TRUE if x is a named vector of type numeric, otherwise FALSE.

Note

This function is based on checkmate::test_numeric() and checkmate::test_names() functions.

Examples

```
h_test_named_numeric(1:2, permutation.of = c("a", "b"))
h_test_named_numeric(c(a = 1, b = 2), permutation.of = c("a", "b"))
h_test_named_numeric(c(a = 1, b = 2), permutation.of = c("b", "a"))
```

---

h_unpack_stopit *Helper function to recursively unpack stopping rules and return lists with logical value and label given*

Description

Helper function to recursively unpack stopping rules and return lists with logical value and label given

Usage

```
h_unpack_stopit(stopit_tree)
```

Arguments

stopit_tree	object from simulate method

Value

named list

---

h_validate_combine_results *Combining S4 Class Validation Results*

Description

[Experimental]

A simple helper function that combines two outputs from calls to `result()` function which is placed in a slot of Validate() reference class.

Usage

```
h_validate_combine_results(v1, v2)
```

Arguments

v1	(logical or character) an output from result() function from Validate() reference class, to be combined with v2.
v2	(logical or character) an output from result() function from Validate() reference class, to be combined with v1.

Examples

```
h_validate_combine_results(TRUE, "some_message")
```

---

h_validate_common_data_slots

Helper Function performing validation Common to Data and DataOrdinal

---

Description

Helper Function performing validation Common to Data and DataOrdinal

Usage

```
h_validate_common_data_slots(object)
```

Arguments

object	(Data or DataOrdinal) the object to be validated

Value

a Validate object containing the result of the validation

Increments-class Increments

Description

[Stable]

Increments is a virtual class for controlling increments, from which all other specific increments classes inherit.

Usage

```
.DefaultIncrements()
```

Note

Typically, end users will not use the `.DefaultIncrements()` function.

See Also

IncrementsRelative, IncrementsRelativeDLT, IncrementsDoseLevels, IncrementsHSRBeta, IncrementsMin.

IncrementsDoseLevels-class
IncrementsDoseLevels

Description

[Stable]

IncrementsDoseLevels is the class for increments control based on the number of dose levels.

Usage

```
IncrementsDoseLevels(levels = 1L, basis_level = "last")

.DefaultIncrementsDoseLevels()
```

Arguments

`levels`	(count) see slot definition.
`basis_level`	(string) see slot definition.

Slots

`levels` (count)
: maximum number of dose levels to increment for the next dose. It defaults to 1, which means that no dose skipping is allowed, i.e. the next dose can be maximum one level higher than the current base dose. The current base dose level is the dose level used to increment from (see `basis_level` parameter).

`basis_level` (string)
: defines the current base dose level. It can take one out of two possible values: `last` or `max`. If `last` is specified (default), the current base dose level is set to the last dose given. If `max` is specified, then the current base dose level is set to the maximum dose level given.

Note

Typically, end users will not use the `.DefaultIncrementsDoseLevels()` function.

Examples

```
# The rule for dose increments which allows for maximum skip one dose level,
# that is 2 dose levels higher than the last dose given.
my_increments <- IncrementsDoseLevels(levels = 2, basis_level = "last")
```

`IncrementsHSRBeta-class` `IncrementsHSRBeta`

Description

[Experimental]

`IncrementsHSRBeta` is a class for limiting further increments using a Hard Safety Rule based on the Bin-Beta model. Increment control is based on the number of observed DLTs and number of subjects at each dose level. The probability of toxicity is calculated using a Bin-Beta model with prior (a,b). If the probability exceeds the threshold for a given dose, that dose and all doses above are excluded from further escalation. This is a hard safety rule that limits further escalation based on the observed data per dose level, independent from the underlying model.

Usage

```
IncrementsHSRBeta(target = 0.3, prob = 0.95, a = 1, b = 1)

.DefaultIncrementsHSRBeta()
```

Arguments

`target`	(`proportion`) see slot definition.
`prob`	(`proportion`) see slot definition.
`a`	(`number`) see slot definition.
`b`	(`number`) see slot definition.

Slots

`target` (`proportion`)
: the target toxicity, except 0 or 1.

`prob` (`proportion`)
: the threshold probability (except 0 or 1) for a dose being toxic.

`a` (`number`)
: shape parameter $a > 0$ of probability distribution Beta (a,b).

`b` (`number`)
: shape parameter $b > 0$ of probability distribution Beta (a,b).

Note

Typically, end users will not use the `.DefaultIncrementsHSRBeta()` function.

Examples

```
# Limit the escalation with a hard safety criteria to the doses that are below
# the first dose that is toxic with a probability of 0.95.
my_increments <- IncrementsHSRBeta(target = 0.3, prob = 0.95)
```

IncrementsMin-class IncrementsMin

Description

[Stable]

`IncrementsMin` is the class that combines multiple increment rules with the `minimum` operation. Slot `increments_list` contains all increment rules, which are itself the objects of class `Increments`. The minimum of these individual increments is taken to give the final maximum increment.

Usage

```
IncrementsMin(increments_list)

.DefaultIncrementsMin()
```

Arguments

`increments_list`
: (`list`)
see slot definition.

Slots

`increments_list` (`list`)
: list with increment rules.

Note

Typically, end users will not use the `.DefaultIncrementsMin()` function.

Examples

```
# As example, here we are combining 2 different increment rules.

# The first rule is the following:
# maximum doubling the dose if no DLTs were observed at the current dose
# or maximum increasing the dose by 1.33 if 1 or 2 DLTs were observed at the current dose
# or maximum increasing the dose by 1.22 if 3 or more DLTs were observed.
```

```
my_increments_1 <- IncrementsRelativeDLT(
  intervals = c(0, 1, 3),
  increments = c(1, 0.33, 0.2)
)

# The second rule is the following:
# maximum doubling the dose if the current dose is <20
# or only maximum increasing the dose by 1.33 if the current dose is >=20.
my_increments_2 <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

# Now we combine the 2 rules.
comb_increments <- IncrementsMin(
  increments_list = list(my_increments_1, my_increments_2)
)
```

`IncrementsOrdinal-class` `IncrementsOrdinal`

Description

[Experimental]

`IncrementsOrdinal` is the class for applying a standard `Increments` rule to the results of an ordinal CRM trial.

Usage

```
IncrementsOrdinal(grade, rule)

.DefaultIncrementsOrdinal()
```

Arguments

`grade`	(`numeric`) see slot definition.
`rule`	(`Increments`) see slot definition.

Slots

`grade` (`integer`)
: the toxicity grade to which the `rule` should be applied.

`rule` (`Increments`)
: the standard `Increments` rule to be applied

Note

Typically, end users will not use the `.DefaultIncrementsOrdinal()` function.

Examples

```
IncrementsOrdinal(
  grade = 1L,
  rule = IncrementsRelative(
    intervals = c(0, 20),
    increments = c(1, 0.33)
  )
)
```

IncrementsRelative-class
IncrementsRelative

Description

[Stable]

`IncrementsRelative` is the class for increments control based on relative differences in intervals.

Usage

```
IncrementsRelative(intervals, increments)

.DefaultIncrementsRelative()
```

Arguments

`intervals`	(`numeric`) see slot definition.
`increments`	(`numeric`) see slot definition.

Slots

`intervals` (`numeric`)
: a vector with the left bounds of the relevant intervals. For example, `intervals = c(0, 50, 100)` specifies three intervals: $(0, 50)$, $[50, 100)$ and $[100, +Inf)$. That means, the right bound of the intervals are exclusive to the interval and the last interval goes from the last value to infinity.

`increments` (`numeric`)
: a vector of the same length with the maximum allowable relative increments in the `intervals`.

Note

Typically, end users will not use the `.DefaultIncrementsRelative()` function.

Examples

```
# This is the example of a rule for:
# maximum doubling the dose if the current dose is <20
# or only maximum increasing the dose by 1.33 if the current dose is >=20.
my_increments <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)
```

IncrementsRelativeDLT-class
IncrementsRelativeDLT

Description

[Stable]

IncrementsRelativeDLT is the class for increments control based on relative differences in terms of DLTs.

Usage

```
IncrementsRelativeDLT(intervals, increments)

.DefaultIncrementsRelativeDLT()
```

Arguments

intervals	(numeric) see slot definition.
increments	(numeric) see slot definition.

Slots

`intervals` (integer)
: a vector with the left bounds of the relevant DLT intervals. For example, `intervals = c(0, 1, 3)` specifies three intervals (sets of DLTs: first, 0 DLT; second 1 or 2 DLTs; and the third one, at least 3 DLTs. That means, the right bound of the intervals are exclusive to the interval and the last interval goes from the last value to infinity.

`increments` (numeric)
: a vector of maximum allowable relative increments corresponding to `intervals`. IT must be of the same length as the length of `intervals`.

Note

This considers all DLTs across all cohorts observed so far.

Typically, end users will not use the `.DefaultIncrementsRelativeDLT()` function.

See Also

IncrementsRelativeDLTCurrent which only considers the DLTs in the current cohort.

Examples

```
# This is the example of a rule for:
# maximum doubling the dose if no DLTs were observed in the whole study so far
# or maximum increasing the dose by 1.33 if 1 or 2 DLTs were observed so far
# or maximum increasing the dose by 1.22 if 3 or more DLTs were observed so far.
my_increments <- IncrementsRelativeDLT(
  intervals = c(0, 1, 3),
  increments = c(1, 0.33, 0.2)
)
```

IncrementsRelativeDLTCurrent-class
IncrementsRelativeDLTCurrent

Description

[Experimental]

IncrementsRelativeDLTCurrent is the class for increments control based on relative differences and current DLTs. The class is based on the number of DLTs observed in the current cohort, but not cumulatively over all cohorts so far.

Usage

```
IncrementsRelativeDLTCurrent(intervals = c(0L, 1L), increments = c(2L, 1L))

.DefaultIncrementsRelativeDLTCurrent()
```

Arguments

intervals	(numeric) see slot definition.
increments	(numeric) see slot definition.

Note

Typically, end users will not use the .DefaultIncrementsRelativeDLTCurrent() function.

See Also

IncrementsRelativeDLT.

Examples

```
# As example, here is the rule for:
# maximum doubling the dose if no DLTs were observed at the current dose
# or maximum increasing the dose by 1.33 if 1 or 2 DLTs were observed at the current dose
# or maximum increasing the dose by 1.22 if 3 or more DLTs were observed.

my_increments <- IncrementsRelativeDLTCurrent(
  intervals = c(0, 1, 3),
  increments = c(1, 0.33, 0.2)
)
```

IncrementsRelativeParts-class

IncrementsRelativeParts

Description

[Stable]

IncrementsRelativeParts is the class for increments control based on relative differences in intervals, with special rules for part 1 and beginning of part 2.

Usage

```
IncrementsRelativeParts(dlt_start, clean_start, ...)

.DefaultIncrementsRelativeParts()
```

Arguments

dlt_start	(count) see slot definition.
clean_start	(count) see slot definition.
...	Arguments passed on to IncrementsRelative intervals (numeric) see slot definition. increments (numeric) see slot definition.

Details

This class works only in conjunction with DataParts objects. If part 2 will just be started in the next cohort, then the next maximum dose will be either dlt_start (e.g. -1) shift of the last part 1 dose in case of a DLT in part 1, or clean_start shift (e.g. -1) in case of no DLTs in part 1, given that clean_start <= 0 (see description of clean_start slot for more details). If part 1 will still be on in the next cohort, then the next dose level will be the next higher dose level in the part1Ladder slot of the data object. If part 2 has been started before, the usual relative increment rules apply, see IncrementsRelative.

Slots

`dlt_start (integer)`
: a scalar, the dose level increment for starting part 2 in case of at least one DLT event in part 1.

`clean_start (integer)`
: a scalar, the dose level increment for starting part 2 in case of no DLTs in part 1. If `clean_start <= 0`, then the part 1 ladder will be used to find the maximum next dose. Otherwise, the relative increment rules will be applied to find the next maximum dose level.

Note

We require that `clean_start >= dlt_start`. However, this precondition is not a prerequisite for any function (except of the class' validation function) that works with objects of this class. It is rather motivated by the semantics. That is, if we observe a DLT in part 1, we cannot be more aggressive than in case of a clean part 1 without DLT.

Typically, end users will not use the `.DefaultIncrementsRelativeParts()` function.

Examples

```
my_increments <- IncrementsRelativeParts(dlt_start = 0, clean_start = 1)
```

---

knit_print *Render a* `CohortSizeConst` *Object*

Description

[Experimental]

We provide additional utility functions to allow human-friendly rendition of crmPack objects in Markdown and Quarto files

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

We provide additional utility functions to allow human-friendly rendition of crmPack objects in Markdown and Quarto files. This file contains methods for all design classes, not just those that are direct descendants of `Design`.

[Experimental]

[Experimental]

[Experimental]

[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]
[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

We provide additional utility functions to allow human-friendly rendition of crmPack objects in Markdown and Quarto files

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

[Experimental]

Usage

```
## S3 method for class 'CohortSizeConst'
knit_print(x, ..., asis = TRUE, label = c("participant", "participants"))

## S3 method for class 'CohortSizeRange'
knit_print(x, ..., asis = TRUE)

## S3 method for class 'CohortSizeDLT'
knit_print(x, ..., tox_label = "toxicity", asis = TRUE)
```

```
## S3 method for class 'CohortSizeParts'
knit_print(x, ..., asis = TRUE, label = c("participant", "participants"))

## S3 method for class 'CohortSizeMax'
knit_print(x, ..., asis = TRUE)

## S3 method for class 'CohortSizeMin'
knit_print(x, ..., asis = TRUE)

## S3 method for class 'CohortSizeOrdinal'
knit_print(x, ..., tox_label = "toxicity", asis = TRUE)

## S3 method for class 'StartingDose'
knit_print(x, ..., asis = TRUE)

## S3 method for class 'RuleDesign'
knit_print(x, ..., level = 2L, title = "Design", sections = NA, asis = TRUE)

## S3 method for class 'Design'
knit_print(x, ..., level = 2L, title = "Design", sections = NA, asis = TRUE)

## S3 method for class 'DualDesign'
knit_print(x, ..., level = 2L, title = "Design", sections = NA, asis = TRUE)

## S3 method for class 'DADesign'
knit_print(x, ..., level = 2L, title = "Design", sections = NA, asis = TRUE)

## S3 method for class 'TDDesign'
knit_print(x, ..., level = 2L, title = "Design", sections = NA, asis = TRUE)

## S3 method for class 'DualResponsesDesign'
knit_print(x, ..., level = 2L, title = "Design", sections = NA, asis = TRUE)

## S3 method for class 'DesignOrdinal'
knit_print(x, ..., level = 2L, title = "Design", sections = NA, asis = TRUE)

## S3 method for class 'DesignGrouped'
knit_print(
  x,
  ...,
  level = 2L,
  title = "Design",
  sections = c(model = "Dose toxicity model", mono = "Monotherapy rules", combo =
    "Combination therapy rules", other = "Other details"),
  asis = TRUE
)
```

```
## S3 method for class 'TDsamplesDesign'
knit_print(x, ..., level = 2L, title = "Design", sections = NA, asis = TRUE)

## S3 method for class 'DualResponsesDesign'
knit_print(x, ..., level = 2L, title = "Design", sections = NA, asis = TRUE)

## S3 method for class 'DualResponsesSamplesDesign'
knit_print(x, ..., level = 2L, title = "Design", sections = NA, asis = TRUE)

## S3 method for class 'RuleDesignOrdinal'
knit_print(x, ..., level = 2L, title = "Design", sections = NA, asis = TRUE)

## S3 method for class 'GeneralData'
knit_print(
  x,
  ...,
  asis = TRUE,
  label = c("participant", "participants"),
  full_grid = FALSE,
  summarise = c("none", "dose", "cohort"),
  summarize = summarise,
  units = NA,
  format_func = function(x) x
)

## S3 method for class 'DataParts'
knit_print(
  x,
  ...,
  asis = TRUE,
  label = c("participant", "participants"),
  full_grid = FALSE,
  summarise = c("none", "dose", "cohort"),
  summarize = summarise,
  units = NA,
  format_func = function(x) x
)

## S3 method for class 'DualEndpoint'
knit_print(
  x,
  ...,
  asis = TRUE,
  use_values = TRUE,
  fmt = "%5.2f",
  units = NA,
  tox_label = "toxicity",
  biomarker_label = "PD biomarker"
```

```
)

## S3 method for class 'ModelParamsNormal'
knit_print(
  x,
  use_values = TRUE,
  fmt = "%5.2f",
  params = c("alpha", "beta"),
  preamble = "The prior for &theta; is given by\\n",
  asis = TRUE,
  theta = "\\theta",
  ...
)

## S3 method for class 'GeneralModel'
knit_print(
  x,
  ...,
  params = c("alpha", "beta"),
  asis = TRUE,
  use_values = TRUE,
  fmt = "%5.2f",
  units = NA
)

## S3 method for class 'LogisticKadane'
knit_print(
  x,
  ...,
  asis = TRUE,
  use_values = TRUE,
  fmt = "%5.2f",
  units = NA,
  tox_label = "toxicity"
)

## S3 method for class 'LogisticKadaneBetaGamma'
knit_print(
  x,
  ...,
  asis = TRUE,
  use_values = TRUE,
  fmt = "%5.2f",
  tox_label = "toxicity",
  units = NA
)

## S3 method for class 'LogisticLogNormal'
```

```
knit_print(
  x,
  ...,
  use_values = TRUE,
  fmt = "%5.2f",
  params = c(`\\alpha` = "alpha", `log(\\beta)` = "beta"),
  preamble = "The prior for &theta; is given by\\n",
  asis = TRUE
)

## S3 method for class 'LogisticLogNormalMixture'
knit_print(x, ..., asis = TRUE, use_values = TRUE, fmt = "%5.2f", units = NA)

## S3 method for class 'LogisticLogNormalSub'
knit_print(
  x,
  ...,
  use_values = TRUE,
  fmt = "%5.2f",
  params = c(`\\alpha` = "alpha", `log(\\beta)` = "beta"),
  preamble = "The prior for &theta; is given by\\n",
  asis = TRUE
)

## S3 method for class 'LogisticNormalMixture'
knit_print(x, ..., asis = TRUE, use_values = TRUE, fmt = "%5.2f", units = NA)

## S3 method for class 'LogisticNormalFixedMixture'
knit_print(x, ..., asis = TRUE, use_values = TRUE, fmt = "%5.2f", units = NA)

## S3 method for class 'OneParLogNormalPrior'
knit_print(
  x,
  ...,
  tox_label = "toxicity",
  asis = TRUE,
  use_values = TRUE,
  fmt = "%5.2f"
)

## S3 method for class 'OneParExpPrior'
knit_print(x, ..., asis = TRUE)

## S3 method for class 'LogisticLogNormalGrouped'
knit_print(
  x,
  ...,
  use_values = TRUE,
```

```
  fmt = "%5.2f",
  params = c(`\\alpha` = "alpha", `\\beta` = "beta", `log(\\delta_0)` = "delta_0",
    `log(\\delta_1)` = "delta_1"),
  preamble = "The prior for &theta; is given by\\n",
  asis = TRUE
)

## S3 method for class 'LogisticLogNormalOrdinal'
knit_print(
  x,
  ...,
  use_values = TRUE,
  fmt = "%5.2f",
  params = NA,
  preamble = "The prior for &theta; is given by\\n",
  asis = TRUE
)

## S3 method for class 'LogisticIndepBeta'
knit_print(
  x,
  ...,
  use_values = TRUE,
  fmt = "%5.2f",
  params = NA,
  tox_label = "DLAE",
  preamble = "The prior for &theta; is given by\\n",
  asis = TRUE
)

## S3 method for class 'Effloglog'
knit_print(
  x,
  ...,
  use_values = TRUE,
  fmt = "%5.2f",
  params = NA,
  tox_label = "DLAE",
  eff_label = "efficacy",
  label = "participant",
  preamble = "The prior for &theta; is given by\\n",
  asis = TRUE
)

## S3 method for class 'IncrementsRelative'
knit_print(x, ..., asis = TRUE)

## S3 method for class 'IncrementsRelativeDLT'
```

```
knit_print(x, ..., asis = TRUE)

## S3 method for class 'IncrementsDoseLevels'
knit_print(x, ..., asis = TRUE)

## S3 method for class 'IncrementsHSRBeta'
knit_print(x, ..., asis = TRUE)

## S3 method for class 'IncrementsMin'
knit_print(x, ..., asis = TRUE)

## S3 method for class 'IncrementsOrdinal'
knit_print(x, ..., asis = TRUE)

## S3 method for class 'IncrementsRelativeParts'
knit_print(x, ..., asis = TRUE, tox_label = c("toxicity", "toxicities"))

## S3 method for class 'IncrementsRelativeDLTCurrent'
knit_print(x, ..., asis = TRUE, tox_label = c("DLT", "DLTs"))

## S3 method for class 'NextBestMTD'
knit_print(
  x,
  ...,
  target_label = "the 25th centile",
  tox_label = "toxicity",
  asis = TRUE
)

## S3 method for class 'NextBestNCRM'
knit_print(x, ..., tox_label = "toxicity", asis = TRUE)

## S3 method for class 'NextBestThreePlusThree'
knit_print(
  x,
  ...,
  tox_label = c("toxicity", "toxicities"),
  label = "participant",
  asis = TRUE
)

## S3 method for class 'NextBestDualEndpoint'
knit_print(
  x,
  ...,
  tox_label = "toxicity",
  biomarker_label = "the biomarker",
  biomarker_units = ifelse(x@target_relative, "%", ""),
```

```
  asis = TRUE
)

## S3 method for class 'NextBestMinDist'
knit_print(x, ..., tox_label = "toxicity", asis = TRUE)

## S3 method for class 'NextBestInfTheory'
knit_print(
  x,
  ...,
  tox_label = "toxicity",
  citation_text = "Mozgunov & Jaki (2019)",
  citation_link = "https://doi.org/10.1002/sim.8450",
  asis = TRUE
)

## S3 method for class 'NextBestTD'
knit_print(x, ..., tox_label = "toxicity", asis = TRUE)

## S3 method for class 'NextBestMaxGain'
knit_print(x, ..., tox_label = "toxicity", asis = TRUE)

## S3 method for class 'NextBestProbMTDLTE'
knit_print(x, ..., tox_label = "toxicity", asis = TRUE)

## S3 method for class 'NextBestProbMTDMinDist'
knit_print(x, ..., tox_label = "toxicity", asis = TRUE)

## S3 method for class 'NextBestNCRMLoss'
knit_print(
  x,
  ...,
  tox_label = "toxicity",
  asis = TRUE,
  format_func = function(x) {
     kableExtra::kable_styling(x, bootstrap_options =
    c("striped", "hover", "condensed"))
 }
)

## S3 method for class 'NextBestTDsamples'
knit_print(x, ..., tox_label = "toxicity", asis = TRUE)

## S3 method for class 'NextBestMaxGainSamples'
knit_print(x, ..., tox_label = "toxicity", asis = TRUE)

## S3 method for class 'NextBestOrdinal'
knit_print(x, ..., tox_label = "toxicity", asis = TRUE)
```

```
## S3 method for class 'SafetyWindow'
knit_print(x, ..., asis = TRUE, time_unit = "day", label = "participant")

## S3 method for class 'SafetyWindowConst'
knit_print(
  x,
  ...,
  asis = TRUE,
  label = "participant",
  ordinals = c("first", "second", "third", "fourth", "fifth", "sixth", "seventh",
    "eighth", "ninth", "tenth"),
  time_unit = "day"
)

## S3 method for class 'SafetyWindowSize'
knit_print(
  x,
  ...,
  asis = TRUE,
  ordinals = c("first", "second", "third", "fourth", "fifth", "sixth", "seventh",
    "eighth", "ninth", "tenth"),
  label = "participant",
  time_unit = "day",
  level = 2L
)

## S3 method for class 'StoppingOrdinal'
knit_print(x, ..., asis = TRUE)

## S3 method for class 'StoppingMaxGainCIRatio'
knit_print(x, ..., asis = TRUE)

## S3 method for class 'StoppingList'
knit_print(x, ..., preamble, indent = 0L, asis = TRUE)

## S3 method for class 'StoppingAny'
knit_print(x, ..., preamble, asis = TRUE)

## S3 method for class 'StoppingAll'
knit_print(x, ..., preamble, asis = TRUE)

## S3 method for class 'StoppingTDCIRatio'
knit_print(
  x,
  ...,
  dose_label = "the next best dose",
  tox_label = "toxicity",
```

```
  fmt_string =
   paste0("%sIf, at %s, the ratio of the upper to the lower limit of the posterior ",
    "95%% credible interval for %s (targetting %2.0f%%) is less than or equal to "),
  asis = TRUE
)

## S3 method for class 'StoppingTargetBiomarker'
knit_print(
  x,
  ...,
  dose_label = "the next best dose",
  biomarker_label = "the target biomarker",
  fmt_string =
    paste0("%sIf, at %s, the posterior probability that %s is in the range ",
    "(%.2f, %.2f)%s is %.0f%% or more.\n\n"),
  asis = TRUE
)

## S3 method for class 'StoppingLowestDoseHSRBeta'
knit_print(
  x,
  ...,
  tox_label = "toxicity",
  fmt_string =
   paste0("%sIf, using a Hard Stopping Rule with a prior of Beta(%.0f, %.0f), the ",
    "lowest dose in the dose grid has a posterior probability of %s of ",
    "%.0f%% or more.\n\n"),
  asis = TRUE
)

## S3 method for class 'StoppingMTDCV'
knit_print(
  x,
  ...,
  fmt_string =
   paste0("%sIf the posterior estimate of the robust coefficient of variation of ",
    "the MTD (targetting %2.0f%%), is than or equal to %.0f%%.\n\n"),
  asis = TRUE
)

## S3 method for class 'StoppingMTDdistribution'
knit_print(
  x,
  ...,
  fmt_string =
   "%sIf the mean posterior probability of %s at %.0f%% of %s is at least %4.2f.\n\n",
  dose_label = "the next best dose",
  tox_label = "toxicity",
```

```
  asis = TRUE
)

## S3 method for class 'StoppingHighestDose'
knit_print(
  x,
  ...,
  dose_label = "the highest dose in the dose grid",
  asis = TRUE
)

## S3 method for class 'StoppingSpecificDose'
knit_print(x, ..., dose_label = as.character(x@dose), asis = TRUE)

## S3 method for class 'StoppingTargetProb'
knit_print(
  x,
  ...,
  fmt_string =
    paste0("%sIf the probability of %s at %s is in the range [%4.2f, %4.2f] ",
    "is at least %4.2f.\n\n"),
  dose_label = "the next best dose",
  tox_label = "toxicity",
  asis = TRUE
)

## S3 method for class 'StoppingMinCohorts'
knit_print(x, ..., asis = TRUE)

## S3 method for class 'StoppingMinPatients'
knit_print(x, ..., label = "participant", asis = TRUE)

## S3 method for class 'StoppingPatientsNearDose'
knit_print(
  x,
  ...,
  dose_label = "the next best dose",
  label = "participants",
  asis = TRUE
)

## S3 method for class 'StoppingCohortsNearDose'
knit_print(x, ..., dose_label = "the next best dose", asis = TRUE)

## S3 method for class 'StoppingMissingDose'
knit_print(x, ..., asis = TRUE)
```

Arguments

x	(ModelParamsNormal) the object to be rendered
...	passed to knitr::kable()
asis	(flag) Not used at present
label	(character) the term used to label participants
tox_label	(character) the term used to describe toxicity
level	(count) the markdown level at which the headings for cohort size will be printed. An integer between 1 and 6
title	(character) The text of the heading of the section describing the design
sections	(character) a named vector of length at least 4 defining the headings used to define the sections corresponding to the design's slots. The element names must match the Design's slot names.
full_grid	(flag) Should the full dose grid appear in the output table or simply those doses for whom at least one evaluable participant is available? Ignored unless summarise == "dose".
summarise	(character) How to summarise the observed data. The default, "none", lists observed data at the participant level. "dose" presents participant counts by dose and "cohort" by cohort.
summarize	(character) Synonym for summarise
units	(character) The units in which the values in doseGrid are
format_func	(function) The function used to format the range table.
use_values	(flag) print the values associated with hyperparameters, or the symbols used to define the hyper-parameters. That is, for example, mu or 1.
fmt	(character) the sprintf format string used to render numerical values. Ignored if use_values is FALSE.
biomarker_label	(character) the term used to describe the biomarker
params	(character) The names of the model parameters. See Usage Notes below.
preamble	(character) the text that introduces the list of rules

theta (character)
the LaTeX representation of the theta vector

eff_label (character)
the term used to describe efficacy

target_label (character)
the term used to describe the target toxicity rate

biomarker_units
(character)
the units in which the biomarker is measured

citation_text (character)
the text used to cite Mozgunov & Jaki

citation_link (character)
the link to Mozgunov & Jaki

time_unit (character)
the word used to describe units of time. See Usage Notes below.

ordinals (character)
a character vector whose nth defines the word used as the written representation of the nth ordinal number.

indent (integer)
the indent level of the current stopping rule list. Spaces with length `indent * 4` will be prepended to the beginning of the rendered stopping rule list.

dose_label (character)
the term used to describe the target dose

fmt_string (character)
the character string that defines the format of the output

Value

a character string that represents the object in markdown.

The markdown representation of the object, as a character string

a character string that represents the object in markdown.

A character string containing a LaTeX rendition of the object.

a character string that represents the object in markdown.

Usage Notes

`label` describes the trial's participants.

It should be a character vector of length 1 or 2. If of length 2, the first element describes a `cohort_size` of 1 and the second describes all other `cohort_sizes`. If of length 1, the character `s` is appended to the value when `cohort_size` is not 1.

The default value of `col.names` is `c("Lower", "Upper", "Cohort size")` and that of `caption` is `"Defined by the dose to be used in the next cohort"`. These values can be overridden by passing `col.names` and `caption` in the function call.

The by default, the columns are labelled Lower, Upper and Cohort size. The table's caption is Defined by the number of <tox_label[2]> so far observed. These values can be overridden by passing col.names and caption in the function call.

label describes the trial's participants.

It should be a character vector of length 1 or 2. If of length 2, the first element describes a single participant and the second describes all other situations. If of length 1, the character s is appended to the value when the number of participants is not 1. The default values of col.names and caption vary depending on the summary requested. The default values can be overridden by passing col.names and caption in the function call.

params must be a character vector of length equal to that of x@mean (and x@cov). Its values represent the parameters of the model as entries in the vector theta, on the left-hand side of "~" in the definition of the prior. If named, names should be valid LaTeX, escaped as usual for R character variables. For example, "\\alpha" or "\\beta_0". If unnamed, names are constructed by prepending an escaped backslash to each value provided.

The default value of col.names is c("Min", "Max", "Increment") and that of caption is "Defined by highest dose administered so far". These values can be overridden by passing col.names and caption in the function call.

The default value of col.names is c("Min", "Max", "Increment") and that of caption is "Defined by number of DLTs reported so far". These values can be overridden by passing col.names and caption in the function call.

label defines how toxicities are described.

It should be a character vector of length 1 or 2. If of length 2, the first element describes a single toxicity and the second describes all other toxicity counts. If of length 1, the character s is appended to the value describing a single toxicity.

The default value of col.names is c("Min", "Max", "Increment") and that of caption is "Defined by number of DLTs in the current cohort". These values can be overridden by passing col.names and caption in the function call.

tox_label defines how toxicities are described.

It should be a character vector of length 1 or 2. If of length 2, the first element describes a single toxicity and the second describes all other toxicity counts. If of length 1, the character s is appended to the value describing a single toxicity.

This section describes the use of label and tox_label, collectively referred to as labels. A label should be a scalar or a vector of length 2. If a scalar, it is converted by adding a second element that is equal to the first, suffixed by s. For example, tox_label = "DLT" becomes tox_label = c("DLT", "DLTs"). The first element of the vector is used to describe a count of 1. The second is used in all other cases.

To use a BibTeX-style citation, specify (for example) citation_text = "@MOZGUNOV", citation_link = "".

label should be a character vector of length 1 or 2. If of length 2, the first element describes a count of 1 and the second describes all other counts. If of length 1, the character s is appended to the value when the count is not 1.

label and time_unit are, collectively, labels.

A label should be a character vector of length 1 or 2. If of length 2, the first element describes a count of 1 and the second describes all other counts. If of length 1, the character s is appended to the value when the count is not 1.

label describes the trial's participants.

It should be a character vector of length 1 or 2. If of length 2, the first element describes a cohort_size of 1 and the second describes all other cohort_sizes. If of length 1, the character s is appended to the value when cohort_size is not 1.

The default value of col.names is c("Lower", "Upper", "Cohort size") and that of caption is "Defined by the dose to be used in the next cohort". These values can be overridden by passing col.names and caption in the function call.

The by default, the columns are labelled Lower, Upper and Cohort size. The table's caption is Defined by the number of <tox_label[2]> so far observed. These values can be overridden by passing col.names and caption in the function call.

label describes the trial's participants.

It should be a character vector of length 1 or 2. If of length 2, the first element describes a single participant and the second describes all other situations. If of length 1, the character s is appended to the value when the number of participants is not 1. The default values of col.names and caption vary depending on the summary requested. The default values can be overridden by passing col.names and caption in the function call.

params must be a character vector of length equal to that of x@mean (and x@cov). Its values represent the parameters of the model as entries in the vector theta, on the left-hand side of "~" in the definition of the prior. If named, names should be valid LaTeX, escaped as usual for R character variables. For example, "\\alpha" or "\\beta_0". If unnamed, names are constructed by prepending an escaped backslash to each value provided.

The default value of col.names is c("Min", "Max", "Increment") and that of caption is "Defined by highest dose administered so far". These values can be overridden by passing col.names and caption in the function call.

The default value of col.names is c("Min", "Max", "Increment") and that of caption is "Defined by number of DLTs reported so far". These values can be overridden by passing col.names and caption in the function call.

label defines how toxicities are described.

It should be a character vector of length 1 or 2. If of length 2, the first element describes a single toxicity and the second describes all other toxicity counts. If of length 1, the character s is appended to the value describing a single toxicity.

The default value of col.names is c("Min", "Max", "Increment") and that of caption is "Defined by number of DLTs in the current cohort". These values can be overridden by passing col.names and caption in the function call.

tox_label defines how toxicities are described.

It should be a character vector of length 1 or 2. If of length 2, the first element describes a single toxicity and the second describes all other toxicity counts. If of length 1, the character s is appended to the value describing a single toxicity.

This section describes the use of label and tox_label, collectively referred to as labels. A label should be a scalar or a vector of length 2. If a scalar, it is converted by adding a second element that is equal to the first, suffixed by s. For example, tox_label = "DLT" becomes tox_label = c("DLT", "DLTs"). The first element of the vector is used to describe a count of 1. The second is used in all other cases.

To use a BibTeX-style citation, specify (for example) citation_text = "@MOZGUNOV", citation_link = "".

`label` should be a character vector of length 1 or 2. If of length 2, the first element describes a count of 1 and the second describes all other counts. If of length 1, the character `s` is appended to the value when the count is not 1.

`label` and `time_unit` are, collectively, labels.

A label should be a character vector of length 1 or 2. If of length 2, the first element describes a count of 1 and the second describes all other counts. If of length 1, the character `s` is appended to the value when the count is not 1.

See Also

`knit_print` for more details.

`LogisticIndepBeta-class`

`LogisticIndepBeta`

Description

[Stable]

`LogisticIndepBeta` is the class for the two-parameters logistic regression dose-limiting events (DLE) model with prior expressed in form of pseudo data. This model describes the relationship between the binary DLE responses and the dose levels. More specifically, it represents the relationship of the probabilities of the occurrence of a DLE for corresponding dose levels in log scale. This model is specified as

$$p(x) = exp(phi1 + phi2 * log(x))/(1 + exp(phi1 + phi2 * log(x)))$$

where $p(x)$ is the probability of the occurrence of a DLE at dose x. The two parameters of this model are the intercept $phi1$ and the slope $phi2$. The `LogisticIndepBeta` inherits all slots from `ModelTox` class.

In the context of pseudo data, the following three arguments are used, `binDLE`, `DLEdose` and `DLEweights`. The `DLEdose` represents fixed dose levels at which the pseudo DLE responses `binDLE` are observed. `DLEweights` represents total number of subjects treated per each dose level in `DLEdose`. The `binDLE` represents the number of subjects observed with DLE per each dose level in `DLEdose`. Hence, all these three vectors must be of the same length and the order of the elements in any of the vectors `binDLE`, `DLEdose` and `DLEweights` must be kept, so that an element of a given vector corresponds to the elements of the remaining two vectors (see the example for more insight). Finally, since at least two DLE pseudo responses are needed to obtain prior modal estimates (same as the maximum likelihood estimates) for the model parameters, the `binDLE`, `DLEdose` and `DLEweights` must all be vectors of at least length 2.

Usage

```
LogisticIndepBeta(binDLE, DLEdose, DLEweights, data)

.DefaultLogisticIndepBeta()
```

Arguments

binDLE	(numeric) the number of subjects observed with a DLE, the pseudo DLE responses, depending on dose levels DLEdose. Elements of binDLE must correspond to the elements of DLEdose and DLEweights.
DLEdose	(numeric) dose levels for the pseudo DLE responses. Elements of DLEdose must correspond to the elements of binDLE and DLEweights.
DLEweights	(numeric) the total number of subjects treated at each of the dose levels DLEdose, pseudo weights. Elements of DLEweights must correspond to the elements of binDLE and DLEdose.
data	(Data) the input data to update estimates of the model parameters.

Details

The pseudo data can be interpreted as if we obtain some observations before the trial starts. It can be used to express our prior, i.e. the initial beliefs for the model parameters. The pseudo data is expressed in the following way. First, fix at least two dose levels, then ask for experts' opinion on how many subjects are to be treated at each of these dose levels and on the number of subjects observed with a DLE. At each dose level, the number of subjects observed with a DLE, divided by the total number of subjects treated, is the probability of the occurrence of a DLE at that particular dose level. The probabilities of the occurrence of a DLE based on this pseudo data are independent and they follow Beta distributions. Therefore, the joint prior probability density function of all these probabilities can be obtained. Hence, by a change of variable, the joint prior probability density function of the two parameters in this model can also be obtained. In addition, a conjugate joint prior density function of the two parameters in the model is used. For details about the form of all these joint prior and posterior probability density functions, please refer to Whitehead and Willamson (1998).

Slots

binDLE (numeric)
: a vector of total numbers of DLE responses. It must be at least of length 2 and the order of its elements must correspond to values specified in DLEdose and DLEweights.

DLEdose (numeric)
: a vector of the dose levels corresponding to It must be at least of length 2 and the order of its elements must correspond to values specified in binDLE and DLEweights.

DLEweights (integer)
: total number of subjects treated at each of the pseudo dose level DLEdose. It must be at least of length 2 and the order of its elements must correspond to values specified in binDLE and DLEdose.

phi1 (number)
: the intercept of the model. This slot is used in output to display the resulting prior or posterior modal estimate of the intercept obtained based on the pseudo data and (if any) observed data/responses.

`phi2` (number)
: the slope of the model. This slot is used in output to display the resulting prior or posterior modal estimate of the slope obtained based on the pseudo data and (if any) the observed data/responses.

`Pcov` (matrix)
: refers to the 2x2 covariance matrix of the intercept ($phi1$) and the slope parameters ($phi2$) of the model. This is used in output to display the resulting prior and posterior covariance matrix of $phi1$ and $phi2$ obtained, based on the pseudo data and (if any) the observed data and responses. This slot is needed for internal purposes.

Note

Typically, end users will not use the `.DefaultLogisticIndepBeta()` function.

Examples

```
# Obtain prior modal estimates given the pseudo data.
# First we used an empty data set such that only the dose levels under
# investigations are given. In total, 12 dose levels are under investigation
# ranging from 25 to 300 mg with increments of 25 (i.e 25, 50, 75, ..., 300).
emptydata <- Data(doseGrid = seq(25, 300, 25))

# Fix two dose levels 25 and 300 mg (DLEdose).
# Total number of subjects treated in each of these levels is 3, (DLEweights).
# The number of subjects observed with a DLE is 1.05 at dose 25 mg and 1.8 at dose 300 mg (binDLE).
my_model1 <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEdose = c(25, 300),
  DLEweights = c(3, 3),
  data = emptydata
)

# Use observed DLE responses to obtain posterior modal estimates.
my_data <- Data(
  x = c(25, 50, 50, 75, 100, 100, 225, 300),
  y = c(0, 0, 0, 0, 1, 1, 1, 1),
  doseGrid = emptydata@doseGrid
)

my_model2 <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEdose = c(25, 300),
  DLEweights = c(3, 3),
  data = my_data
)
```

`LogisticKadane-class` `LogisticKadane`

Description

[Stable]

LogisticKadane is the class for the logistic model in the parametrization of Kadane et al. (1980).

Usage

```
LogisticKadane(theta, xmin, xmax)

.DefaultLogisticKadane()
```

Arguments

theta	(proportion) the target toxicity probability.
xmin	(number) the minimum of the dose range.
xmax	(number) the maximum of the dose range.

Details

Let rho0 = p(xmin) be the probability of a DLT at the minimum dose xmin, and let gamma be the dose with target toxicity probability theta, i.e. $p(gamma) = theta$. Then it can easily be shown that the logistic regression model has intercept

$$[gamma * logit(rho0) - xmin * logit(theta)]/[gamma - xmin]$$

and slope

$$[logit(theta) - logit(rho0)]/[gamma - xmin].$$

The priors are

$$gamma\ Unif(xmin, xmax).$$

and

$$rho0\ Unif(0, theta).$$

Slots

theta (proportion)
: the target toxicity probability.

xmin (number)
: the minimum of the dose range.

xmax (number)
: the maximum of the dose range.

Note

The slots of this class, required for creating the model, are the target toxicity, as well as the minimum and maximum of the dose range. Note that these can be different from the minimum and maximum of the dose grid in the data later on.

Typically, end-users will not use the `.DefaultLogisticKadane()` function.

See Also

`ModelLogNormal`

Examples

```
my_model <- LogisticKadane(theta = 0.33, xmin = 1, xmax = 200)
```

LogisticKadaneBetaGamma-class
LogisticKadaneBetaGamma

Description

[Experimental]

`LogisticKadaneBetaGamma` is the class for the logistic model in the parametrization of Kadane et al. (1980), using a beta and a gamma distribution as the model priors.

Usage

```
LogisticKadaneBetaGamma(theta, xmin, xmax, alpha, beta, shape, rate)

.DefaultLogisticKadaneBetaGamma()
```

Arguments

`theta`	(`proportion`) the target toxicity probability.
`xmin`	(`number`) the minimum of the dose range.
`xmax`	(`number`) the maximum of the dose range.
`alpha`	(`number`) the first shape parameter of the Beta prior distribution `rho0` = `p(xmin)` the probability of a DLT at the minimum dose `xmin`.
`beta`	(`number`) the second shape parameter of the Beta prior distribution `rho0` = `p(xmin)` the probability of a DLT at the minimum dose `xmin`.

shape (number)
: the shape parameter of the Gamma prior distribution gamma the dose with target toxicity probability theta.

rate (number)
: the rate parameter of the Gamma prior distribution gamma the dose with target toxicity probability theta.

Details

Let rho0 = p(xmin) be the probability of a DLT at the minimum dose xmin, and let gamma be the dose with target toxicity probability theta, i.e. $p(gamma) = theta$. Then it can easily be shown that the logistic regression model has intercept

$$[gamma * logit(rho0) - xmin * logit(theta)]/[gamma - xmin]$$

and slope

$$[logit(theta) - logit(rho0)]/[gamma - xmin].$$

The prior for gamma, is

$$gamma\ Gamma(shape, rate).$$

. The prior for rho0 = p(xmin), is

$$rho0\ Beta(alpha, beta).$$

Slots

theta (proportion)
: the target toxicity probability.

xmin (number)
: the minimum of the dose range.

xmax (number)
: the maximum of the dose range.

alpha (number)
: the first shape parameter of the Beta prior distribution of rho0 = p(xmin) the probability of a DLT at the minimum dose xmin.

beta (number)
: the second shape parameter of the Beta prior distribution of rho0 = p(xmin) the probability of a DLT at the minimum dose xmin.

shape (number)
: the shape parameter of the Gamma prior distribution of gamma the dose with target toxicity probability theta.

rate (number)
: the rate parameter of the Gamma prior distribution of gamma the dose with target toxicity probability theta.

Note

The slots of this class, required for creating the model, are the same as in the `LogisticKadane` class. In addition, the shape parameters of the Beta prior distribution of `rho0` and the shape and rate parameters of the Gamma prior distribution of `gamma`, are required for creating the prior model.

Typically, end users will not use the `.Default()` function.

See Also

`ModelLogNormal`, `LogisticKadane`.

Examples

```
my_model <- LogisticKadaneBetaGamma(
  theta = 0.3,
  xmin = 0,
  xmax = 7,
  alpha = 1,
  beta = 19,
  shape = 0.5625,
  rate = 0.125
)
```

LogisticLogNormal-class
LogisticLogNormal

Description

[Stable]

`LogisticLogNormal` is the class for the usual logistic regression model with a bivariate normal prior on the intercept and log slope.

Usage

```
LogisticLogNormal(mean, cov, ref_dose = 1)

.DefaultLogisticLogNormal()
```

Arguments

`mean`	(`numeric`) the prior mean vector.
`cov`	(`matrix`) the prior covariance matrix. The precision matrix `prec` is internally calculated as an inverse of `cov`.
`ref_dose`	(`number`) the reference dose $x*$ (strictly positive number).

Details

The covariate is the natural logarithm of the dose x divided by the reference dose $x*$, i.e.:

$$logit[p(x)] = alpha0 + alpha1 * log(x/x*),$$

where $p(x)$ is the probability of observing a DLT for a given dose x. The prior

$$(alpha0, log(alpha1))\ Normal(mean, cov).$$

Note

Typically, end users will not use the `.DefaultLogisticLogNormal()` function.

See Also

`ModelLogNormal`, `LogisticNormal`, `LogisticLogNormalSub`, `ProbitLogNormal`, `ProbitLogNormalRel`, `LogisticLogNormalMixture`, `DALogisticLogNormal`.

Examples

```
my_model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 50
)
my_model
```

LogisticLogNormalGrouped-class
LogisticLogNormalGrouped

Description

[Experimental]

`LogisticLogNormalGrouped` is the class for a logistic regression model for both the mono and the combo arms of the simultaneous dose escalation design.

Usage

```
LogisticLogNormalGrouped(mean, cov, ref_dose = 1)

.DefaultLogisticLogNormalGrouped()
```

Arguments

`mean`	(`numeric`) the prior mean vector.
`cov`	(`matrix`) the prior covariance matrix. The precision matrix `prec` is internally calculated as an inverse of `cov`.
`ref_dose`	(`number`) the reference dose $x*$ (strictly positive number).

Details

The continuous covariate is the natural logarithm of the dose x divided by the reference dose $x*$ as in `LogisticLogNormal`. In addition, I_c is a binary indicator covariate which is 1 for the combo arm and 0 for the mono arm. The model is then defined as:

$$logit[p(x)] = (alpha0 + I_c * delta0) + (alpha1 + I_c * delta1) * log(x/x*),$$

where $p(x)$ is the probability of observing a DLT for a given dose x, and `delta0` and `delta1` are the differences in the combo arm compared to the mono intercept and slope parameters `alpha0` and `alpha1`. The prior is defined as

$$(alpha0, log(delta0), log(alpha1), log(delta1))\ Normal(mean, cov).$$

Note

Typically, end users will not use the `.DefaultLogisticLogNormalGrouped()` function.

See Also

`ModelLogNormal`, `LogisticLogNormal`.

Examples

```
my_model <- LogisticLogNormalGrouped(
  mean = c(-0.85, 0, 1, 0),
  cov = diag(1, 4),
  ref_dose = 50
)
my_model
```

LogisticLogNormalMixture-class
LogisticLogNormalMixture

Description

[Stable]

`LogisticLogNormalMixture` is the class for standard logistic model with online mixture of two bivariate log normal priors.

Usage

```
LogisticLogNormalMixture(mean, cov, ref_dose, share_weight)

.DefaultLogisticLogNormalMixture()
```

Arguments

mean	(numeric) the prior mean vector.
cov	(matrix) the prior covariance matrix. The precision matrix prec is internally calculated as an inverse of cov.
ref_dose	(number) the reference dose $x*$ (strictly positive number).
share_weight	(proportion) the prior weight for the share component.

Details

This model can be used when data is arising online from the informative component of the prior, at the same time with the data of the trial of main interest. Formally, this is achieved by assuming that the probability of a DLT at dose x is given by

$$p(x) = \pi * p1(x) + (1 - \pi) * p2(x)$$

where π is the probability for the model $p(x)$ being the same as the model $p1(x)$, which is the informative component of the prior. From this model data arises in parallel: at doses xshare, DLT information yshare is observed, in total nObsshare data points (see DataMixture). On the other hand, $1 - \pi$, is the probability of a separate model $p2(x)$. Both components have the same log normal prior distribution, which can be specified by the user, and which is inherited from the LogisticLogNormal class.

Slots

share_weight (proportion)
: the prior weight for the share component $p_1(x)$.

Note

Typically, end users will not use the .DefaultLogNormalMixture() function.

See Also

ModelLogNormal, LogisticNormalMixture, LogisticNormalFixedMixture.

Examples

```
# Decide on the dose grid and MCMC options.
dose_grid <- 1:80
my_options <- McmcOptions()

# Classic model.
my_model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 50
)

empty_data <- Data(doseGrid = dose_grid)
prior_samples <- mcmc(empty_data, my_model, my_options)
plot(prior_samples, my_model, empty_data)

# Set up the mixture model and data share object.
model_share <- LogisticLogNormalMixture(
  share_weight = 0.1,
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 50
)

empty_data_share <- DataMixture(
  doseGrid = dose_grid,
  xshare = rep(c(10, 20, 40), each = 4),
  yshare = rep(0L, 12),
)

# Compare with the resulting prior model.
prior_samples_share <- mcmc(empty_data_share, model_share, my_options)
plot(prior_samples_share, model_share, empty_data_share)
```

LogisticLogNormalOrdinal-class

LogisticLogNormalOrdinal

Description

[Experimental]

LogisticLogNormalOrdinal is the class for a logistic lognormal CRM model using an ordinal toxicity scale.

Usage

```
LogisticLogNormalOrdinal(mean, cov, ref_dose)

.DefaultLogisticLogNormalOrdinal()
```

Arguments

mean	(numeric) the prior mean vector.
cov	(matrix) the prior covariance matrix. The precision matrix prec is internally calculated as an inverse of cov.
ref_dose	(number) the reference dose $x*$ (strictly positive number).

Note

Typically, end users will not use the .DefaultLogisticLogNormalOrdinal() function.

Examples

```
LogisticLogNormalOrdinal(
  mean = c(3, 4, 0),
  cov = diag(c(4, 3, 1)),
  ref_dose = 1
)
```

LogisticLogNormalSub-class
LogisticLogNormalSub

Description

[Stable]

LogisticLogNormalSub is the class for a standard logistic model with bivariate (log) normal prior with subtractive dose standardization.

Usage

```
LogisticLogNormalSub(mean, cov, ref_dose = 0)

.DefaultLogisticLogNormalSub()
```

Arguments

mean	(numeric) the prior mean vector.
cov	(matrix) the prior covariance matrix. The precision matrix prec is internally calculated as an inverse of cov.
ref_dose	(number) the reference dose $x*$.

Details

The covariate is the dose x minus the reference dose $x*$, i.e.:

$$logit[p(x)] = alpha0 + alpha1 * (x - x*),$$

where $p(x)$ is the probability of observing a DLT for a given dose x. The prior

$$(alpha0, log(alpha1))\ Normal(mean, cov).$$

Slots

`params` (`ModelParamsNormal`)
: bivariate normal prior parameters.

`ref_dose` (`number`)
: the reference dose $x*$.

Note

Typically, end-users will not use the `.DefaultLogisticLogNormalSub()` function.

See Also

`LogisticNormal`, `LogisticLogNormal`, `ProbitLogNormal`, `ProbitLogNormalRel`.

Examples

```
my_model <- LogisticLogNormalSub(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 50
)
```

LogisticNormal-class LogisticNormal

Description

[Stable]

`LogisticNormal` is the class for the usual logistic regression model with a bivariate normal prior on the intercept and slope.

Usage

```
LogisticNormal(mean, cov, ref_dose = 1)

.DefaultLogisticNormal()
```

Arguments

mean	(numeric) the prior mean vector.
cov	(matrix) the prior covariance matrix. The precision matrix prec is internally calculated as an inverse of cov.
ref_dose	(number) the reference dose $x*$ (strictly positive number).

Details

The covariate is the natural logarithm of the dose x divided by the reference dose $x*$, i.e.:

$$logit[p(x)] = alpha0 + alpha1 * log(x/x*),$$

where $p(x)$ is the probability of observing a DLT for a given dose x. The prior

$$(alpha0, alpha1)\ Normal(mean, cov).$$

Note

Typically, end users will not use the `.DefaultLogisticNormal()` function.

See Also

ModelLogNormal, LogisticLogNormal, LogisticLogNormalSub, ProbitLogNormal, ProbitLogNormalRel, LogisticNormalMixture.

Examples

```
# Define the dose-grid.
empty_data <- Data(doseGrid = c(1, 3, 5, 10, 15, 20, 25, 40, 50, 80, 100))

my_model <- LogisticNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2)
)

my_options <- McmcOptions(burnin = 10, step = 2, samples = 100)

samples <- mcmc(empty_data, my_model, my_options)
samples
```

`LogisticNormalFixedMixture-class` `LogisticNormalFixedMixture`

Description

[Stable]

`LogisticNormalFixedMixture` is the class for standard logistic regression model with fixed mixture of multiple bivariate (log) normal priors on the intercept and slope parameters. The weights of the normal priors are fixed, hence no additional model parameters are introduced. This type of prior is often used to better approximate a given posterior distribution, or when the information is given in terms of a mixture.

Usage

```
LogisticNormalFixedMixture(components, weights, ref_dose, log_normal = FALSE)

.DefaultLogisticNormalFixedMixture()
```

Arguments

`components`	(`list`) the specifications of the mixture components, a list with `ModelParamsNormal` objects for each bivariate (log) normal prior.
`weights`	(`numeric`) the weights of the components; these must be positive and will be normalized to sum to 1.
`ref_dose`	(`number`) the reference dose $x*$ (strictly positive number).
`log_normal`	(`flag`) should a log normal prior be specified, such that the mean vectors and covariance matrices are valid for the intercept and log slope?

Details

The covariate is the natural logarithm of the dose x divided by the reference dose $x*$, i.e.:

$$logit[p(x)] = alpha0 + alpha1 * log(x/x*),$$

where $p(x)$ is the probability of observing a DLT for a given dose x. The prior

$$(alpha0, alpha1)\ w1 * Normal(mean1, cov1) + ... + wK * Normal(meanK, covK),$$

if a normal prior is used and

$$(alpha0, log(alpha1))\ w1 * Normal(mean1, cov1) + ... + wK * Normal(meanK, covK),$$

if a log normal prior is used. The weights $w1, ..., wK$ of the components are fixed and sum to 1.

The slots of this class comprise a list with components parameters. Every single component contains the mean vector and the covariance matrix of bivariate normal distributions. Remaining slots are the weights of the components as well as the reference dose. Moreover, a special indicator slot specifies whether a log normal prior is used.

Slots

`components` (`list`)
: the specifications of the mixture components, a list with `ModelParamsNormal` objects for each bivariate (log) normal prior.

`weights` (`numeric`)
: the weights of the components; these must be positive and must sum to 1.

`ref_dose` (`positive_number`)
: the reference dose.

`log_normal` (`flag`)
: should a log normal prior be used, such that the mean vectors and covariance matrices are valid for the intercept and log slope?

Note

Typically, end-users will not use the `.DefaultLogisticNormalFixedMixture()` function.

See Also

`ModelParamsNormal`, `ModelLogNormal`, `LogisticNormalMixture`, `LogisticLogNormalMixture`.

Examples

```
my_model <- LogisticNormalFixedMixture(
  components = list(
    comp1 = ModelParamsNormal(
      mean = c(-0.85, 1),
      cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2)
    ),
    comp2 = ModelParamsNormal(
      mean = c(1, 1.5),
      cov = matrix(c(1.2, -0.45, -0.45, 0.6), nrow = 2)
    )
  ),
  weights = c(0.3, 0.7),
  ref_dose = 50
)
```

`LogisticNormalMixture-class` `LogisticNormalMixture`

Description

[Stable]

`LogisticNormalMixture` is the class for standard logistic regression model with a mixture of two bivariate normal priors on the intercept and slope parameters.

Usage

```
LogisticNormalMixture(comp1, comp2, weightpar, ref_dose)

.DefaultLogisticNormalMixture()
```

Arguments

`comp1`	(`ModelParamsNormal`) bivariate normal prior specification of the first component. See `ModelParamsNormal` for more details.
`comp2`	(`ModelParamsNormal`) bivariate normal prior specification of the second component. See `ModelParamsNormal` for more details.
`weightpar`	(`numeric`) the beta parameters for the weight of the first component. It must a be a named vector of length 2 with names `a` and `b` and with strictly positive values.
`ref_dose`	(`number`) the reference dose $x*$ (strictly positive number).

Details

The covariate is the natural logarithm of the dose x divided by the reference dose $x*$, i.e.:

$$logit[p(x)] = alpha0 + alpha1 * log(x/x*),$$

where $p(x)$ is the probability of observing a DLT for a given dose x. The prior

$$(alpha0, alpha1)\ w * Normal(mean1, cov1) + (1 - w) * Normal(mean2, cov2).$$

The weight w for the first component is assigned a beta prior `B(a, b)`.

Slots

`comp1` (`ModelParamsNormal`)
: bivariate normal prior specification of the first component.

`comp2` (`ModelParamsNormal`)
: bivariate normal prior specification of the second component.

`weightpar` (`numeric`)
: the beta parameters for the weight of the first component. It must a be a named vector of length 2 with names `a` and `b` and with strictly positive values.

`ref_dose` (`positive_number`)
: the reference dose.

Note

The weight of the two normal priors is a model parameter, hence it is a flexible mixture. This type of prior is often used with a mixture of a minimal informative and an informative component, in order to make the CRM more robust to data deviations from the informative component.

Typically, end-users will not use the `.DefaultLogisticNormalMixture()` function.

See Also

`ModelParamsNormal`, `ModelLogNormal`, `LogisticNormalFixedMixture`, `LogisticLogNormalMixture`.

Examples

```
my_model <- LogisticNormalMixture(
  comp1 = ModelParamsNormal(
    mean = c(-0.85, 1),
    cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2)
  ),
  comp2 = ModelParamsNormal(
    mean = c(1, 1.5),
    cov = matrix(c(1.2, -0.45, -0.45, 0.6), nrow = 2)
  ),
  weightpar = c(a = 1, b = 1),
  ref_dose = 50
)
```

---

logit *Shorthand for logit function*

Description

Shorthand for logit function

Usage

```
logit(x)
```

Arguments

x	the function argument

Value

the logit(x)

---

match_within_tolerance *Helper function for value matching with tolerance*

Description

This is a modified version of match that supports tolerance.

Usage

```
match_within_tolerance(x, table)
```

Arguments

x	the values to be matched
table	the values to be matched against

Value

A vector of the same length as x or empty vector if table is empty.

---

maxDose *Determine the Maximum Possible Next Dose*

Description

[Stable]

This function determines the upper limit of the next dose based on the incrementsand the data.

Usage

```
maxDose(increments, data, ...)

## S4 method for signature 'IncrementsRelative,Data'
maxDose(increments, data, ...)

## S4 method for signature 'IncrementsRelativeDLT,Data'
maxDose(increments, data, ...)

## S4 method for signature 'IncrementsRelativeDLTCurrent,Data'
maxDose(increments, data, ...)

## S4 method for signature 'IncrementsRelativeParts,DataParts'
maxDose(increments, data, ...)

## S4 method for signature 'IncrementsDoseLevels,Data'
maxDose(increments, data, ...)

## S4 method for signature 'IncrementsHSRBeta,Data'
maxDose(increments, data, ...)

## S4 method for signature 'IncrementsMin,Data'
maxDose(increments, data, ...)

## S4 method for signature 'IncrementsMin,DataOrdinal'
maxDose(increments, data, ...)

## S4 method for signature 'IncrementsOrdinal,DataOrdinal'
maxDose(increments, data, ...)
```

Arguments

`increments`	(`Increments`) the rule for the next best dose.
`data`	(`Data`) input data.
`...`	additional arguments without method dispatch.

Value

A `number`, the maximum possible next dose.

Functions

- `maxDose(increments = IncrementsRelative, data = Data)`: determine the maximum possible next dose based on relative increments.
- `maxDose(increments = IncrementsRelativeDLT, data = Data)`: determine the maximum possible next dose based on relative increments determined by DLTs so far.

- `maxDose(increments = IncrementsRelativeDLTCurrent, data = Data)`: determine the maximum possible next dose based on relative increments determined by DLTs in the current cohort.
- `maxDose(increments = IncrementsRelativeParts, data = DataParts)`: determine the maximum possible next dose based on relative increments as well as part 1 and beginning of part 2.
- `maxDose(increments = IncrementsDoseLevels, data = Data)`: determine the maximum possible next dose based on the number of dose grid levels. That is, the max dose is determined as the one which level is equal to: base dose level + level increment. The base dose level is the level of the last dose in grid or the level of the maximum dose applied, which is defined in `increments` object. Find out more in `IncrementsDoseLevels`.
- `maxDose(increments = IncrementsHSRBeta, data = Data)`: determine the maximum possible next dose for escalation.
- `maxDose(increments = IncrementsMin, data = Data)`: determine the maximum possible next dose based on multiple increment rules, taking the minimum across individual increments.
- `maxDose(increments = IncrementsMin, data = DataOrdinal)`: determine the maximum possible next dose based on multiple increment rules, taking the minimum across individual increments.
- `maxDose(increments = IncrementsOrdinal, data = DataOrdinal)`: determine the maximum possible next dose in an ordinal CRM trial

Examples

```
# Example of usage for `IncrementsRelative` maxDose class.

# Create the data.
my_data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 8, 8, 8),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  ID = 1:8,
  cohort = c(0, 1, 2, 3, 4, 5, 5, 5),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, 8, 10:40)
)

# Define a rule for dose increments which allows for:
#  - doubling the dose if the last dose was below 20,
#  - increasing the dose by 33% of the last dose, only if the last dose was
#    above or equal to 20.
my_increments <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

# Based on the rule above, the maximum dose allowed is:
max_dose <- maxDose(my_increments, data = my_data)
# Example of usage for `IncrementsRelativeDLT` maxDose class.

# Create the data.
```

```
my_data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 8, 8, 8),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  ID = 1:8,
  cohort = c(0, 1, 2, 3, 4, 5, 5, 5),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, 8, seq(from = 10, to = 80, by = 2))
)

# Define a rule for dose increments which allows for:
#  - doubling the dose if no DLTs were yet observed,
#  - increasing the dose by 33% if 1 or 2 DLTs were already observed,
#  - increasing the dose by 20% if at least 3 DLTs were already observed.
my_increments <- IncrementsRelativeDLT(
  intervals = c(0, 1, 3),
  increments = c(1, 0.33, 0.2)
)

# Based on the rule above, the maximum dose allowed is:
max_dose <- maxDose(my_increments, data = my_data)
# Example of usage for `IncrementsRelativeDLTCurrent` maxDose class.

# Create the data.
my_data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  ID = 1:8,
  cohort = c(0, 1, 2, 3, 4, 5, 5, 5),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)

# Define a rule for dose increments which allows for:
#  - doubling the dose if no DLTs were observed in current (i.e. last) cohort,
#  - only increasing the dose by 33% if 1 or 2 DLTs were observed in current cohort,
#  - only increasing the dose by 20% if at least 3 DLTs were observed in current cohort.
my_increments <- IncrementsRelativeDLTCurrent(
  intervals = c(0, 1, 3),
  increments = c(1, 0.33, 0.2)
)

# Based on the rule above, the maximum dose allowed is:
max_dose <- maxDose(my_increments, data = my_data)
# Example of usage for `IncrementsRelativeParts` maxDose class.

# Create an object of class `DataParts`.
my_data <- DataParts(
  x = c(0.1, 0.5, 1.5),
  y = c(0, 0, 0),
  ID = 1:3,
  cohort = 1:3,
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, 10),
  part = c(1L, 1L, 1L),
  nextPart = 1L,
  part1Ladder = c(0.1, 0.5, 1.5, 3, 6, 10)
```

```
)

my_increments <- IncrementsRelativeParts(
  dlt_start = 0,
  clean_start = 1
)

max_dose <- maxDose(my_increments, data = my_data)
# Example of usage for `IncrementsDoseLevels` maxDose class.

# Create the data.
my_data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 8, 8, 12, 12, 12, 16, 16, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1),
  ID = 1:14,
  cohort = c(1, 2, 3, 4, 5, 6, 6, 7, 7, 7, 8, 8, 9, 9),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, 8, 10:30)
)

# In this first example we define a rule for dose increments which allows for
# maximum skip one dose level, that is 2 dose levels higher than the last dose
# given.
my_increments_1 <- IncrementsDoseLevels(levels = 2, basis_level = "last")

# Based on the rule above, the maximum dose allowed is:
max_dose_1 <- maxDose(my_increments_1, data = my_data)

# In this second example we define a rule for dose increments which allows for
# maximum skip one dose level, that is 2 dose levels higher than the max dose
# given.
my_increments_2 <- IncrementsDoseLevels(levels = 2, basis_level = "max")

# Based on the rule above, the maximum dose allowed is:
max_dose_2 <- maxDose(my_increments_2, data = my_data)

# Create the data.
my_data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 8, 8, 8, 6, 6, 6),
  y = c(0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1),
  cohort = c(0, 1, 2, 3, 4, 5, 5, 5, 6, 6, 6),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, 8,
      seq(from = 10, to = 80, by = 2)
    )
)

# In this example we define a rule for dose increments that limits the further
# dose escalation to doses below 6, because dose 6 is above the probability
# toxicity threshold.
my_increments <- IncrementsHSRBeta(target = 0.3, prob = 0.95)

# Based on the rule above, we then calculate the maximum dose allowed.
my_next_max_dose <- maxDose(my_increments, data = my_data)
# Example of usage for `IncrementsRelativeDLTCurrent` maxDose class.
```

```
# Create the data.
my_data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 8, 8, 8),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  ID = 1:8,
  cohort = c(0, 1, 2, 3, 4, 5, 5, 5),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, 8, 10:80)
)

# Here, we combine two different increment rules.

# The first rule allows for:
#  - doubling the dose if no DLTs were observed at the current dose,
#  - increasing the dose by 33% if 1 or 2 DLTs were observed at the current dose,
#  - increasing the dose by 22% if 3 or more DLTs were observed.
my_increments_1 <- IncrementsRelativeDLT(
  intervals = c(0, 1, 3),
  increments = c(1, 0.33, 0.2)
)

# The second rule allows for:
#  - doubling the dose if the current dose is <20,
#  - increasing the dose by 33% if the current dose is >=20.
my_increments_2 <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

# Finally, the maximum dose allowed is computed by taking the minimum dose from
# the maximum doses computed by the two rules.
my_increments <- IncrementsMin(
  increments_list = list(my_increments_1, my_increments_2)
)
max_dose <- maxDose(my_increments, my_data)
maxDose(
  increments = IncrementsOrdinal(2L, .DefaultIncrementsRelative()),
  data = .DefaultDataOrdinal()
)
```

`maxSize` *"MAX" combination of cohort size rules*

Description

This function combines cohort size rules by taking the maximum of all sizes.

Usage

```
maxSize(...)
```

```
## S4 method for signature 'CohortSize'
maxSize(...)
```

Arguments

...	Objects of class CohortSize

Value

the combination as an object of class CohortSizeMax

Functions

- maxSize(CohortSize): The method combining cohort size rules by taking maximum

See Also

minSize

Examples

```
# Here is the rule for:
#   having cohort of size 1 for doses <30
#   and having cohort of size 3 for doses >=30
mySize1 <- CohortSizeRange(intervals = c(0, 30), cohort_size = c(1, 3))

# Here is the rule for:
#   having cohort of size 1 until no DLT were observed
#   and having cohort of size 3 as soon as 1 DLT is observed
mySize2 <- CohortSizeDLT(intervals = c(0, 1), cohort_size = c(1, 3))

# This is combining the two rules above by taking the maximum of the sample sizes of
# the single rules
mySize <- maxSize(mySize1, mySize2)
```

---

mcmc *Obtaining Posterior Samples for all Model Parameters*

Description

[Stable]

This is the function that actually runs the JAGS MCMC machinery to produce posterior samples from all model parameters and required derived values. It is a generic function, so that customized versions may be conveniently defined for specific subclasses of GeneralData, GeneralModel, and McmcOptions input.

Usage

```
mcmc(data, model, options, ...)

## S4 method for signature 'GeneralData,GeneralModel,McmcOptions'
mcmc(data, model, options, from_prior = data@nObs == 0L, ...)

## S4 method for signature 'GeneralData,DualEndpointRW,McmcOptions'
mcmc(data, model, options, from_prior = data@nObs == 0L, ...)

## S4 method for signature 'GeneralData,DualEndpointBeta,McmcOptions'
mcmc(data, model, options, from_prior = data@nObs == 0L, ...)

## S4 method for signature 'GeneralData,DualEndpointEmax,McmcOptions'
mcmc(data, model, options, from_prior = data@nObs == 0L, ...)

## S4 method for signature 'GeneralData,OneParLogNormalPrior,McmcOptions'
mcmc(data, model, options, from_prior = data@nObs == 0L, ...)

## S4 method for signature 'GeneralData,OneParExpPrior,McmcOptions'
mcmc(data, model, options, from_prior = data@nObs == 0L, ...)

## S4 method for signature 'DataMixture,GeneralModel,McmcOptions'
mcmc(
  data,
  model,
  options,
  from_prior = data@nObs == 0L & data@nObsshare == 0L,
  ...
)

## S4 method for signature 'Data,LogisticIndepBeta,McmcOptions'
mcmc(data, model, options, ...)

## S4 method for signature 'DataDual,Effloglog,McmcOptions'
mcmc(data, model, options, ...)

## S4 method for signature 'DataDual,EffFlexi,McmcOptions'
mcmc(data, model, options, ...)

## S4 method for signature 'DataOrdinal,LogisticLogNormalOrdinal,McmcOptions'
mcmc(data, model, options, ...)
```

Arguments

data	(GeneralData) an input data.
model	(GeneralModel) an input model.

options	(McmcOptions) MCMC options.
...	not used.
from_prior	(flag) sample from the prior only? Default to TRUE when number of observations in data is 0. For some models it might be necessary to specify it manually here though.

Value

The posterior samples, an object of class Samples.

Functions

- mcmc(data = GeneralData, model = GeneralModel, options = McmcOptions): Standard method which uses JAGS.
- mcmc(data = GeneralData, model = DualEndpointRW, options = McmcOptions): Standard method which uses JAGS. For the DualEndpointRW model, it is required that there are at least two (in case of random walk prior of the first order on the biomarker level) or three doses in the grid.
- mcmc(data = GeneralData, model = DualEndpointBeta, options = McmcOptions): Standard method which uses JAGS. For the DualEndpointBeta model, it is required that the value of ref_dose_beta slot is greater than the maximum dose in a grid. This requirement comes from definition of the beta function that is used to model dose-biomarker relationship in DualEndpointBeta model. The other requirement is that there must be at least one dose in the grid.
- mcmc(data = GeneralData, model = DualEndpointEmax, options = McmcOptions): Standard method which uses JAGS. For the DualEndpointEmax model, it is required that there is at least one dose in the grid.
- mcmc(data = GeneralData, model = OneParLogNormalPrior, options = McmcOptions): Standard method which uses JAGS. For the OneParLogNormalPrior model, it is required that the length of skeleton prior probabilities vector should be equal to the length of the number of doses.
- mcmc(data = GeneralData, model = OneParExpPrior, options = McmcOptions): Standard method which uses JAGS. For the OneParExpPrior model, it is required that the length of skeleton prior probabilities vector should be equal to the length of the number of doses.
- mcmc(data = DataMixture, model = GeneralModel, options = McmcOptions): Method for DataMixture with different from_prior default
- mcmc(data = Data, model = LogisticIndepBeta, options = McmcOptions): Obtain posterior samples for the model parameters based on the pseudo 'LogisticsIndepBeta' DLE model. The joint prior and posterior probability density function of the intercept ϕ_1 (phi1) and the slope ϕ_2 (phi2) are given in Whitehead and Williamson (1998) and TsuTakawa (1975). However, since asymptotically, the joint posterior probability density will be bivariate normal and we will use the bivariate normal distribution to generate posterior samples of the intercept and the slope parameters. For the prior samples of of the intercept and the slope a bivariate normal distribution with mean and the covariance matrix given in Whitehead and Williamson (1998) is used.

- mcmc(data = DataDual, model = Effloglog, options = McmcOptions): Obtain the posterior samples for the model parameters in the Efficacy log log model. Given the value of ν, the precision of the efficacy responses, the joint prior or the posterior probability of the intercept θ_1 (theta1) and the slope θ_2 (theta2) is a bivariate normal distribution. The ν (nu), the precision of the efficacy responses is either a fixed value or has a gamma distribution. If a gamma distribution is used, the samples of nu will be first generated. Then the mean of the of the nu samples will be used the generate samples of the intercept and slope parameters of the model
- mcmc(data = DataDual, model = EffFlexi, options = McmcOptions): Obtain the posterior samples for the estimates in the Efficacy Flexible form. This is the mcmc procedure based on what is described in Lang and Brezger (2004) such that samples of the mean efficacy responses at all dose levels, samples of sigma2 $sigma^2$, the variance of the efficacy response and samples of sigma2betaW $sigma^2_{betaW}$, the variance of the random walk model will be generated. Please refer to Lang and Brezger (2004) for the procedures and the form of the joint prior and posterior probability density for the mean efficacy responses. In addition, both sigma2 and sigma2betaW can be fixed or having an inverse-gamma prior and posterior distribution. Therefore, if the inverse gamma distribution(s) are used, the parameters in the distribution will be first updated and then samples of sigma2 and sigma2betaW will be generated using the updated parameters.
- mcmc(data = DataOrdinal, model = LogisticLogNormalOrdinal, options = McmcOptions): Obtain the posterior samples for the model parameters in the LogisticLogNormalOrdinal. The generic mcmc method returns a Samples object with elements of the data slot named alpha[1], alpha[2], ..., alpha[k] and beta when passed a LogisticLogNormalOrdinal object. This makes the "alpha elements" awkward to access and is inconsistent with other Model objects. So rename the alpha elements to alpha1, alpha2, ..., alpha<k> for ease and consistency.

Note

The type of Random Number Generator (RNG) and its initial seed used by JAGS are taken from the options argument. If no initial values are supplied (i.e RNG kind or seed slot in options has NA), then they will be generated automatically by JAGS.

Examples

```
# Create some data from the class `Data`.
my_data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)

# Initialize the CRM model.
my_model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 56
)

# Sample from the posterior distribution.
my_options <- McmcOptions(
```

```
  burnin = 100,
  step = 2,
  samples = 1000
)

samples <- mcmc(data = my_data, model = my_model, options = my_options)
samples
# Create some data from the class `DataDual`.
plcb <- 0.001
my_data <- DataDual(
  w = c(13, 77, 86, 26, 27, 36, 37, 97, 21, 49, 87, 48),
  x = c(plcb, 25, 25, 25, plcb, 50, 50, 50, plcb, 100, 100, 100),
  y = c(0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L, 1L, 0L),
  doseGrid = c(plcb, seq(25, 300, 25)),
  placebo = TRUE,
  ID = 1:12,
  cohort = c(1L, 1L, 1L, 1L, 2L, 2L, 2L, 2L, 3L, 3L, 3L, 3L)
)

# Initialize the CRM model.
my_model <- DualEndpointRW(
  mean = c(0, 1),
  cov = matrix(c(1, 0, 0, 1), nrow = 2),
  sigma2W = c(a = 0.1, b = 0.1),
  rho = c(a = 1, b = 1),
  sigma2betaW = 0.01,
  rw1 = TRUE
)

# Sample from the posterior distribution.
my_options <- McmcOptions(
  burnin = 50,
  step = 2,
  samples = 4,
  rng_kind = "Mersenne-Twister",
  rng_seed = 1
)

samples <- mcmc(data = my_data, model = my_model, options = my_options)
samples
# Create some data from the class `DataDual`.
plcb <- 0.001
my_data <- DataDual(
  w = c(13, 77, 86, 26, 27, 36, 37, 97, 21, 49, 87, 48),
  x = c(plcb, 25, 25, 25, plcb, 50, 50, 50, plcb, 100, 100, 100),
  y = c(0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L, 1L, 0L),
  doseGrid = c(plcb, seq(25, 300, 25)),
  placebo = TRUE,
  ID = 1:12,
  cohort = c(1L, 1L, 1L, 1L, 2L, 2L, 2L, 2L, 3L, 3L, 3L, 3L)
)

# Initialize the CRM model.
```

```
my_model <- DualEndpointBeta(
  mean = c(0, 1),
  cov = diag(2),
  ref_dose = 2,
  use_log_dose = FALSE,
  sigma2W = c(a = 1, b = 2),
  rho = c(a = 1.5, b = 2.5),
  E0 = 2,
  Emax = 50,
  delta1 = 6,
  mode = 9,
  ref_dose_beta = my_data@doseGrid[my_data@nGrid] + 10
)

# Sample from the posterior distribution.
my_options <- McmcOptions(
  burnin = 50,
  step = 2,
  samples = 4,
  rng_kind = "Mersenne-Twister",
  rng_seed = 1
)

samples <- mcmc(data = my_data, model = my_model, options = my_options)
samples
##obtain mcmc DLE samples given the data, LogisticIndepBeta (DLE model) and mcmc simulations options
## data must be of 'Data' class
data<-Data(x=c(25,50,50,75,100,100,225,300),y=c(0,0,0,0,1,1,1,1),
           doseGrid=seq(25,300,25))
## model must be of 'LogisticIndepBeta' class
model<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=data)
## options must be ''McmcOptions' class
options<-McmcOptions(burnin=100,step=2,samples=200)
set.seed(94)
samples<-mcmc(data=data,model=model,options=options)
# nolint start
##obtain mcmc efficacy samples given the data, 'Effloglog' model (efficacy model) and
## mcmc simulations options data must be of 'DataDual' class
data<-DataDual(x=c(25,50,25,50,75,300,250,150),
               y=c(0,0,0,0,0,1,1,0),
               w=c(0.31,0.42,0.59,0.45,0.6,0.7,0.6,0.52),
               doseGrid=seq(25,300,25),placebo=FALSE)
## model must be of 'Effloglog' class
Effmodel<-Effloglog(eff=c(1.223,2.513),eff_dose=c(25,300),nu=c(a=1,b=0.025),data=data)

## options must be ''McmcOptions' class
options<-McmcOptions(burnin=100,step=2,samples=200)
set.seed(94)
samples<-mcmc(data=data,model=Effmodel,options=options)
# nolint end
## obtain mcmc efficacy samples given the data, 'EffFlexi' model (efficacy model) and
## mcmc simulations options
## data must be of 'DataDual' class
```

```
data <- DataDual(
  x = c(25, 50, 25, 50, 75, 300, 250, 150),
  y = c(0, 0, 0, 0, 0, 1, 1, 0),
  w = c(0.31, 0.42, 0.59, 0.45, 0.6, 0.7, 0.6, 0.52),
  doseGrid = seq(25, 300, 25)
)
## model must be of 'EffFlexi' class

effmodel <- EffFlexi(
  eff = c(1.223, 2.513), eff_dose = c(25, 300),
  sigma2W = c(a = 0.1, b = 0.1), sigma2betaW = c(a = 20, b = 50), rw1 = FALSE, data = data
)

## options must be ''McmcOptions' class
options <- McmcOptions(burnin = 100, step = 2, samples = 200)
set.seed(94)
samples <- mcmc(data = data, model = effmodel, options = options)
ordinal_data <- .DefaultDataOrdinal()
ordinal_model <- .DefaultLogisticLogNormalOrdinal()
mcmc_options <- .DefaultMcmcOptions()

samples <- mcmc(ordinal_data, ordinal_model, mcmc_options)
```

`McmcOptions-class` `McmcOptions`

Description

[Stable]

`McmcOptions` is a class for the three canonical MCMC options as well as Random Number Generator settings.

Usage

```
McmcOptions(
  burnin = 10000L,
  step = 2L,
  samples = 10000L,
  rng_kind = NA_character_,
  rng_seed = NA_integer_
)

.DefaultMcmcOptions()
```

Arguments

`burnin`	(count) number of burn-in iterations which are not saved.

step (count)
only every step-th iteration is saved after the burn-in.

samples (count)
number of resulting samples.

rng_kind (string)
the name of the RNG type. Possible types are: Wichmann-Hill, Marsaglia-Multicarry, Super-Duper, Mersenne-Twister. If it is NA (default), then the RNG kind will be chosen by [rjags].

rng_seed (number)
RNG seed corresponding to chosen rng_kind. It must be an integer value or NA (default), which means that the seed will be chosen by [rjags].

Slots

iterations (count)
number of MCMC iterations.

burnin (count)
number of burn-in iterations which are not saved.

step (count)
only every step-th iteration is saved after the burnin. In other words, a sample from iteration i = 1,...,iterations, is saved if and only if (i - burnin) mod step = 0.
For example, for iterations = 6, burnin = 0 and step = 2, only samples from iterations 2,4,6 will be saved.

rng_kind (string)
a Random Number Generator (RNG) type used by rjags. It must be one out of the following four values: base::Wichmann-Hill, base::Marsaglia-Multicarry, base::Super-Duper, base::Mersenne-Twister, or NA_character_. If it is NA_character_ (default), then the RNG kind will be chosen by rjags.

rng_seed (number)
a Random Number Generator (RNG) seed used by rjags for a chosen rng_kind. It must be an integer scalar or NA_integer_, which means that the seed will be chosen by rjags.

Note

Typically, end users will not use the .DefaultMcmcOptions() function.

Examples

```
# Set up MCMC option in order to have a burn-in of 10000 iterations and
# then take every other iteration up to a collection of 10000 samples.
McmcOptions(burnin = 10000, step = 2, samples = 10000)
```

`MinimalInformative` *Construct a minimally informative prior*

Description

This function constructs a minimally informative prior, which is captured in a `LogisticNormal` (or `LogisticLogNormal`) object.

Usage

```
MinimalInformative(
  dosegrid,
  refDose,
  threshmin = 0.2,
  threshmax = 0.3,
  probmin = 0.05,
  probmax = 0.05,
  ...
)
```

Arguments

`dosegrid`	the dose grid
`refDose`	the reference dose
`threshmin`	Any toxicity probability above this threshold would be very unlikely (see `probmin`) at the minimum dose (default: 0.2)
`threshmax`	Any toxicity probability below this threshold would be very unlikely (see `probmax`) at the maximum dose (default: 0.3)
`probmin`	the prior probability of exceeding `threshmin` at the minimum dose (default: 0.05)
`probmax`	the prior probability of being below `threshmax` at the maximum dose (default: 0.05)
`...`	additional arguments for computations, see `Quantiles2LogisticNormal`, e.g. `refDose` and `logNormal=TRUE` to obtain a minimal informative log normal prior.

Details

Based on the proposal by Neuenschwander et al (2008, Statistics in Medicine), a minimally informative prior distribution is constructed. The required key input is the minimum (d_1 in the notation of the Appendix A.1 of that paper) and the maximum value (d_J) of the dose grid supplied to this function. Then `threshmin` is the probability threshold q_1, such that any probability of DLT larger than q_1 has only 5% probability. Therefore q_1 is the 95% quantile of the beta distribution and hence $p_1 = 0.95$. Likewise, `threshmax` is the probability threshold q_J, such that any probability of DLT smaller than q_J has only 5% probability ($p_J = 0.05$). The probabilities $1 - p_1$ and p_J can be controlled with the arguments `probmin` and `probmax`, respectively. Subsequently, for all doses supplied

in the dosegrid argument, beta distributions are set up from the assumption that the prior medians are linear in log-dose on the logit scale, and Quantiles2LogisticNormal is used to transform the resulting quantiles into an approximating LogisticNormal (or LogisticLogNormal) model. Note that the reference dose is not required for these computations.

Value

see Quantiles2LogisticNormal

Examples

```
# Setting up a minimal informative prior
# max.time is quite small only for the purpose of showing the example. They
# should be increased for a real case.
set.seed(132)
coarseGrid <- c(0.1, 10, 30, 60, 100)
minInfModel <- MinimalInformative(dosegrid = coarseGrid,
                                  refDose=50,
                                  threshmin=0.2,
                                  threshmax=0.3,
                                  control=## for real case: leave out control
                                    list(max.time=0.1))

# Plotting the result
matplot(x=coarseGrid,
        y=minInfModel$required,
        type="b", pch=19, col="blue", lty=1,
        xlab="dose",
        ylab="prior probability of DLT")
matlines(x=coarseGrid,
         y=minInfModel$quantiles,
         type="b", pch=19, col="red", lty=1)
legend("right",
       legend=c("quantiles", "approximation"),
       col=c("blue", "red"),
       lty=1,
       bty="n")
```

---

minSize *"MIN" combination of cohort size rules*

Description

This function combines cohort size rules by taking the minimum of all sizes.

Usage

```
minSize(...)

## S4 method for signature 'CohortSize'
minSize(...)
```

Arguments

...	Objects of class CohortSize

Value

the combination as an object of class CohortSizeMin

Functions

- minSize(CohortSize): The method combining cohort size rules by taking minimum

See Also

maxSize

Examples

```
# Here is the rule for:
#   having cohort of size 1 for doses <30
#   and having cohort of size 3 for doses >=30
mySize1 <- CohortSizeRange(intervals = c(0, 30), cohort_size = c(1, 3))

# Here is the rule for:
#   having cohort of size 1 until no DLT were observed
#   and having cohort of size 3 as soon as 1 DLT is observed
mySize2 <- CohortSizeDLT(intervals = c(0, 1), cohort_size = c(1, 3))

# This is combining the two rules above by taking the minimum of the sample sizes of
# the single rules
mySize <- minSize(mySize1, mySize2)
```

ModelEff-class ModelEff

Description

[Stable]

ModelEff is the parent class for efficacy models using pseudo data prior. It is dedicated all efficacy models that have their prior specified in the form of pseudo data (as if there is some data before the trial starts).

The data must obey the convention of the DataDual class. This refers to any observed efficacy/biomarker responses (w in DataDual), the dose levels at which these responses are observed (x in DataDual), all dose levels considered in the study (doseGrid in DataDual), and finally other specifications in DataDual class that can be used to generate prior or posterior modal estimates or samples estimates for model parameter(s). If no responses are observed, at least doseGrid has to be specified in data for which prior modal estimates or samples can be obtained for model parameters based on the specified pseudo data.

Usage

```
.DefaultModelEff()
```

Slots

data (DataDual)
: observed data that is used to obtain model parameters estimates or samples (see details above).

Note

Typically, end users will not use the .DefaultModelEff() function.

See Also

ModelTox.

ModelLogNormal-class ModelLogNormal

Description

[Stable]

ModelLogNormal is the class for a model with a reference dose and bivariate normal prior on the model parameters alpha0 and natural logarithm of alpha1, i.e.:

$$(alpha0, log(alpha1))\ Normal(mean, cov),$$

. Transformations other than log, e.g. identity, can be specified too in priormodel slot. The parameter alpha1 has a log-normal distribution by default to ensure positivity of alpha1 which further guarantees exp(alpha1) > 1. The slots of this class contain the mean vector, the covariance and precision matrices of the bivariate normal distribution, as well as the reference dose. Note that the precision matrix is an inverse of the covariance matrix in the JAGS. All ("normal") model specific classes inherit from this class.

Usage

```
ModelLogNormal(mean, cov, ref_dose = 1)

.DefaultModelLogNormal()
```

Arguments

mean	(numeric) the prior mean vector.
cov	(matrix) the prior covariance matrix. The precision matrix prec is internally calculated as an inverse of cov.
ref_dose	(number) the reference dose $x*$ (strictly positive number).

Slots

params (ModelParamsNormal)
: bivariate normal prior parameters.

ref_dose (positive_number)
: the reference dose.

Note

Typically, end users will not use the .DefaultModelLogNormal() function.

See Also

ModelParamsNormal, LogisticNormal, LogisticLogNormal, LogisticLogNormalSub, ProbitLogNormal, ProbitLogNormalRel.

ModelParamsNormal-class
ModelParamsNormal

Description

[Experimental]

ModelParamsNormal is the class for a bivariate normal model parameters, i.e. the mean vector, covariance matrix and precision matrix. The precision matrix is an inverse of the covariance matrix in the JAGS and it is computed internally by the object constructor function.

Usage

```
ModelParamsNormal(mean, cov)

.DefaultModelParamsNormal()
```

Arguments

mean	(numeric) the prior mean vector.
cov	(matrix) the prior covariance matrix. The precision matrix prec is internally calculated as an inverse of cov.

Slots

mean (numeric)
: the mean vector.

cov (matrix)
: the covariance matrix.

prec (matrix)
: the precision matrix, which is an inverse matrix of the cov.

Note

Typically, end users will not use the .ModelPAramsNormal() function.

See Also

ModelLogNormal, LogisticNormalMixture.

Examples

```
ModelParamsNormal(mean = c(1, 6), cov = diag(2))
```

ModelPseudo-class ModelPseudo

Description

[Stable]

ModelPseudo is the parent class for models that express their prior in the form of pseudo data (as if there is some data before the trial starts).

Usage

```
.DefaultModelPseudo()
```

Note

Typically, end users will not use the .DefaultModelPseudo() function.

See Also

GeneralModel.

ModelTox-class ModelTox

Description

[Stable]

ModelTox is the parent class for DLE (dose-limiting events) models using pseudo data prior. It is dedicated for DLE models or toxicity models that have their prior specified in the form of pseudo data (as if there is some data before the trial starts).

The data must obey the convention of the Data class. This refers to any observed DLE responses (y in Data), the dose levels (x in Data) at which these responses are observed, all dose levels considered in the study (doseGrid in Data), and finally other specifications in Data class that can be used to generate prior or posterior modal estimates or samples estimates for model parameter(s). If no responses are observed, at least doseGrid has to be specified in data for which prior modal estimates or samples can be obtained for model parameters based on the specified pseudo data.

Usage

```
.DefaultModelTox()
```

Slots

data (Data)
: observed data that is used to obtain model parameters estimates or samples (see details above).

Note

Typically, end users will not use the .DefaultModelTox() function.

See Also

ModelEff.

names,Samples-method *The Names of the Sampled Parameters*

Description

[Stable]

A method that returns names of the parameters that are sampled.

Usage

```
## S4 method for signature 'Samples'
names(x)
```

Arguments

x	(Samples) object with samples.

Examples

```
my_samples <- Samples(
  data = list(alpha = 1:5, beta = 15:19),
  options = McmcOptions(burnin = 2, step = 2, samples = 5)
)

names(my_samples)
```

---

nextBest *Finding the Next Best Dose*

Description

[Stable]

A function that computes the recommended next best dose based on the corresponding rule `nextBest`, the posterior `samples` from the `model` and the underlying `data`.

Usage

```
nextBest(nextBest, doselimit, samples, model, data, ...)

## S4 method for signature 'NextBestMTD,numeric,Samples,GeneralModel,Data'
nextBest(nextBest, doselimit = Inf, samples, model, data, ...)

## S4 method for signature 'NextBestNCRM,numeric,Samples,GeneralModel,Data'
nextBest(nextBest, doselimit = Inf, samples, model, data, ...)

## S4 method for signature
## 'NextBestNCRM,numeric,Samples,GeneralModel,DataParts'
nextBest(nextBest, doselimit = Inf, samples, model, data, ...)

## S4 method for signature 'NextBestNCRMLoss,numeric,Samples,GeneralModel,Data'
nextBest(nextBest, doselimit = Inf, samples, model, data, ...)

## S4 method for signature
## 'NextBestThreePlusThree,missing,missing,missing,Data'
nextBest(nextBest, doselimit, samples, model, data, ...)

## S4 method for signature
## 'NextBestDualEndpoint,numeric,Samples,DualEndpoint,Data'
nextBest(nextBest, doselimit = Inf, samples, model, data, ...)
```

```
## S4 method for signature 'NextBestMinDist,numeric,Samples,GeneralModel,Data'
nextBest(nextBest, doselimit = Inf, samples, model, data, ...)

## S4 method for signature
## 'NextBestInfTheory,numeric,Samples,GeneralModel,Data'
nextBest(nextBest, doselimit = Inf, samples, model, data, ...)

## S4 method for signature 'NextBestTD,numeric,missing,LogisticIndepBeta,Data'
nextBest(nextBest, doselimit = Inf, model, data, in_sim = FALSE, ...)

## S4 method for signature
## 'NextBestTDsamples,numeric,Samples,LogisticIndepBeta,Data'
nextBest(nextBest, doselimit = Inf, samples, model, data, in_sim, ...)

## S4 method for signature 'NextBestMaxGain,numeric,missing,ModelTox,DataDual'
nextBest(
  nextBest,
  doselimit = Inf,
  model,
  data,
  model_eff,
  in_sim = FALSE,
  ...
)

## S4 method for signature
## 'NextBestMaxGainSamples,numeric,Samples,ModelTox,DataDual'
nextBest(
  nextBest,
  doselimit = Inf,
  samples,
  model,
  data,
  model_eff,
  samples_eff,
  in_sim = FALSE,
  ...
)

## S4 method for signature
## 'NextBestProbMTDLTE,numeric,Samples,GeneralModel,Data'
nextBest(nextBest, doselimit, samples, model, data, ...)

## S4 method for signature
## 'NextBestProbMTDMinDist,numeric,Samples,GeneralModel,Data'
nextBest(nextBest, doselimit, samples, model, data, ...)

## S4 method for signature 'NextBestOrdinal,numeric,Samples,GeneralModel,Data'
```

```
nextBest(nextBest, doselimit = Inf, samples, model, data, ...)

## S4 method for signature
## 'NextBestOrdinal,numeric,Samples,LogisticLogNormalOrdinal,DataOrdinal'
nextBest(nextBest, doselimit = Inf, samples, model, data, ...)
```

Arguments

nextBest	(NextBest) the rule for the next best dose.
doselimit	(number) the maximum allowed next dose. If it is an infinity (default), then essentially no dose limit will be applied in the course of dose recommendation calculation.
samples	(Samples) posterior samples from model parameters given data.
model	(ModelTox) the DLT model.
data	(Data) data that was used to generate the samples.
...	additional arguments without method dispatch.
in_sim	(flag) is this method used in simulations? Default as FALSE. If this flag is TRUE and target dose estimates (during trial and end-of-trial) are outside of the dose grid range, the information message is printed by this method.
model_eff	(Effloglog or EffFlexi) the efficacy model.
samples_eff	(Samples) posterior samples from model_eff parameters given data.

Value

A list with the next best dose recommendation (element named value) from the grid defined in data, and a plot depicting this recommendation (element named plot). In case of multiple plots also an element named singlePlots is included. The singlePlots is itself a list with single plots. An additional list with elements describing the outcome of the rule can be contained too.

Functions

- nextBest(nextBest = NextBestMTD, doselimit = numeric, samples = Samples, model = GeneralModel, data = Data): find the next best dose based on the MTD rule.
- nextBest(nextBest = NextBestNCRM, doselimit = numeric, samples = Samples, model = GeneralModel, data = Data): find the next best dose based on the NCRM method. The additional element probs in the output's list contains the target and overdosing probabilities (across all doses in the dose grid) used in the derivation of the next best dose.
- nextBest(nextBest = NextBestNCRM, doselimit = numeric, samples = Samples, model = GeneralModel, data = DataParts): find the next best dose based on the NCRM method when two parts trial is used.

- `nextBest( nextBest = NextBestNCRMLoss, doselimit = numeric, samples = Samples, model = GeneralModel, data = Data )`: find the next best dose based on the NCRM method and loss function.
- `nextBest( nextBest = NextBestThreePlusThree, doselimit = missing, samples = missing, model = missing, data = Data )`: find the next best dose based on the 3+3 method.
- `nextBest( nextBest = NextBestDualEndpoint, doselimit = numeric, samples = Samples, model = DualEndpoint, data = Data )`: find the next best dose based on the dual endpoint model. The additional list element `probs` contains the target and overdosing probabilities (across all doses in the dose grid) used in the derivation of the next best dose.
- `nextBest( nextBest = NextBestMinDist, doselimit = numeric, samples = Samples, model = GeneralModel, data = Data )`: gives the dose which is below the dose limit and has an estimated DLT probability which is closest to the target dose.
- `nextBest( nextBest = NextBestInfTheory, doselimit = numeric, samples = Samples, model = GeneralModel, data = Data )`: gives the appropriate dose within an information theoretic framework.
- `nextBest( nextBest = NextBestTD, doselimit = numeric, samples = missing, model = LogisticIndepBeta, data = Data )`: find the next best dose based only on the DLT responses and for `LogisticIndepBeta` model class object without DLT samples.
- `nextBest( nextBest = NextBestTDsamples, doselimit = numeric, samples = Samples, model = LogisticIndepBeta, data = Data )`: find the next best dose based only on the DLT responses and for `LogisticIndepBeta` model class object involving DLT samples.
- `nextBest( nextBest = NextBestMaxGain, doselimit = numeric, samples = missing, model = ModelTox, data = DataDual )`: find the next best dose based only on pseudo DLT model `ModelTox` and `Effloglog` efficacy model without samples.
- `nextBest( nextBest = NextBestMaxGainSamples, doselimit = numeric, samples = Samples, model = ModelTox, data = DataDual )`: find the next best dose based on DLT and efficacy responses with DLT and efficacy samples.
- `nextBest( nextBest = NextBestProbMTDLTE, doselimit = numeric, samples = Samples, model = GeneralModel, data = Data )`: find the next best dose based with the highest probability of having a toxicity rate less or equal to the target toxicity level.
- `nextBest( nextBest = NextBestProbMTDMinDist, doselimit = numeric, samples = Samples, model = GeneralModel, data = Data )`: find the next best dose based with the highest probability of having a toxicity rate with minimum distance to the target toxicity level.
- `nextBest( nextBest = NextBestOrdinal, doselimit = numeric, samples = Samples, model = GeneralModel, data = Data )`: find the next best dose for ordinal CRM models.
- `nextBest( nextBest = NextBestOrdinal, doselimit = numeric, samples = Samples, model = LogisticLogNormalOrdinal, data = DataOrdinal )`: find the next best dose for ordinal CRM models.

Examples

```
# Example of usage for `NextBestMTD` NextBest class.

# Create the data.
my_data <- Data(
```

```
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  ID = 1:8,
  cohort = c(0, 1, 2, 3, 4, 5, 5, 5),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)

# Initialize the CRM model used to model the data.
my_model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 56
)

# Set-up some MCMC parameters and generate samples from the posterior.
my_options <- McmcOptions(burnin = 100, step = 2, samples = 500)
my_samples <- mcmc(my_data, my_model, my_options)

# Define the rule for dose increments and calculate the maximum dose allowed.
my_increments <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)
next_max_dose <- maxDose(my_increments, data = my_data)

# Define the rule which will be used to select the next best dose
# based on the 'NextBestMTD' class.
mtd_next_best <- NextBestMTD(
  target = 0.33,
  derive = function(mtd_samples) {
    quantile(mtd_samples, probs = 0.25)
  }
)

# Calculate the next best dose.
dose_recommendation <- nextBest(
  nextBest = mtd_next_best,
  doselimit = next_max_dose,
  samples = my_samples,
  model = my_model,
  data = my_data
)

# Example of usage for `NextBestNCRM` NextBest class.

# Create the data.
my_data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  ID = 1:8,
  cohort = c(0, 1, 2, 3, 4, 5, 5, 5),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)
```

```
# Initialize the CRM model used to model the data.
my_model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 56
)

# Set-up some MCMC parameters and generate samples from the posterior.
my_options <- McmcOptions(burnin = 100, step = 2, samples = 500)
my_samples <- mcmc(my_data, my_model, my_options)

# Define the rule for dose increments and calculate the maximum dose allowed.
my_increments <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)
next_max_dose <- maxDose(my_increments, data = my_data)

# Define the rule which will be used to select the next best dose
# based on the 'NextBestNCRM' class.
nrcm_next_best <- NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Calculate the next best dose.
dose_recommendation <- nextBest(
  nextBest = nrcm_next_best,
  doselimit = next_max_dose,
  samples = my_samples,
  model = my_model,
  data = my_data
)

# See the probabilities.
dose_recommendation$probs

# Example of usage for `NextBestNCRM-DataParts` NextBest class.

# Create the data.
my_data <- DataParts(
  x = c(0.1, 0.5, 1.5),
  y = c(0, 0, 0),
  ID = 1:3,
  cohort = 1:3,
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2)),
  part = c(1L, 1L, 1L),
  nextPart = 1L,
  part1Ladder = c(0.1, 0.5, 1.5, 3, 6, 10)
)
```

```
# Initialize the CRM model used to model the data.
my_model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 56
)

# Set-up some MCMC parameters and generate samples from the posterior.
my_options <- McmcOptions(burnin = 100, step = 2, samples = 500)
my_samples <- mcmc(my_data, my_model, my_options)

# Define the rule for dose increments and calculate the maximum dose allowed.
my_increments <- IncrementsRelativeParts(
  dlt_start = 0,
  clean_start = 1
)
next_max_dose <- maxDose(my_increments, data = my_data)

# Define the rule which will be used to select the next best dose
# based on the 'NextBestNCRM' class.
nrcm_next_best <- NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Calculate the next best dose.
dose_recommendation <- nextBest(
  nextBest = nrcm_next_best,
  doselimit = next_max_dose,
  samples = my_samples,
  model = my_model,
  data = my_data
)

dose_recommendation

# Example of usage for `NextBestNCRMLoss` NextBest class.

# Create the data.
my_data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  ID = 1:8,
  cohort = c(0, 1, 2, 3, 4, 5, 5, 5),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)

# Initialize the CRM model used to model the data.
my_model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 56
```

```
)

# Set-up some MCMC parameters and generate samples from the posterior.
my_options <- McmcOptions(burnin = 100, step = 2, samples = 500)
my_samples <- mcmc(my_data, my_model, my_options)

# Define the rule for dose increments and calculate the maximum dose allowed.
my_increments <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)
next_max_dose <- maxDose(my_increments, data = my_data)

# Define the rule which will be used to select the next best dose
# based on the class `NextBestNCRMLoss`.
nrcm_loss_next_best <- NextBestNCRMLoss(
  target = c(0.2, 0.35),
  overdose = c(0.35, 0.6),
  unacceptable = c(0.6, 1),
  max_overdose_prob = 0.999,
  losses = c(1, 0, 1, 2)
)

# Calculate the next best dose.
dose_recommendation <- nextBest(
  nextBest = nrcm_loss_next_best,
  doselimit = next_max_dose,
  samples = my_samples,
  model = my_model,
  data = my_data
)

# Next best dose.
dose_recommendation$value

# Look at the probabilities.
dose_recommendation$probs

# Define another rule (loss function of 3 elements).
nrcm_loss_next_best_losses_3 <- NextBestNCRMLoss(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.30,
  losses = c(1, 0, 2)
)

# Calculate the next best dose.
dose_recommendation_losses_3 <- nextBest(
  nextBest = nrcm_loss_next_best_losses_3,
  doselimit = next_max_dose,
  samples = my_samples,
  model = my_model,
  data = my_data
```

```
)

# Next best dose.
dose_recommendation_losses_3$value

# Look at the probabilities.
dose_recommendation_losses_3$probs

# Example of usage for `NextBestThreePlusThree` NextBest class.

# Create the data.
my_data <- Data(
  x = c(5, 5, 5, 10, 10, 10),
  y = c(0, 0, 0, 0, 1, 0),
  ID = 1:6,
  cohort = c(0, 0, 0, 1, 1, 1),
  doseGrid = c(0.1, 0.5, 1.5, 3, 5, seq(from = 10, to = 80, by = 2))
)

# The rule to select the next best dose will be based on the 3+3 method.
my_next_best <- NextBestThreePlusThree()

# Calculate the next best dose.
dose_recommendation <- nextBest(my_next_best, data = my_data)
# Example of usage for `NextBestDualEndpoint` NextBest class.

# Create the data.
my_data <- DataDual(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10, 20, 20, 20, 40, 40, 40, 50, 50, 50),
  y = c(0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1),
  ID = 1:17,
  cohort = c(1L, 2L, 3L, 4L, 5L, 6L, 6L, 6L, 7L, 7L, 7L, 8L, 8L, 8L, 9L, 9L, 9L),
  w = c(
    0.31, 0.42, 0.59, 0.45, 0.6, 0.7, 0.55, 0.6, 0.52, 0.54,
    0.56, 0.43, 0.41, 0.39, 0.34, 0.38, 0.21
  ),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)

# Initialize the Dual-Endpoint model (in this case RW1).
my_model <- DualEndpointRW(
  mean = c(0, 1),
  cov = matrix(c(1, 0, 0, 1), nrow = 2),
  sigma2betaW = 0.01,
  sigma2W = c(a = 0.1, b = 0.1),
  rho = c(a = 1, b = 1),
  rw1 = TRUE
)

# Set-up some MCMC parameters and generate samples from the posterior.
my_options <- McmcOptions(burnin = 100, step = 2, samples = 500)
my_samples <- mcmc(my_data, my_model, my_options)
```

```
# Define the rule for dose increments and calculate the maximum dose allowed.
my_increments <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)
next_max_dose <- maxDose(my_increments, data = my_data)

# Define the rule which will be used to select the next best dose. In this case,
# target a dose achieving at least 0.9 of maximum biomarker level (efficacy)
# and with a probability below 0.25 that prob(DLT)>0.35 (safety).
de_next_best <- NextBestDualEndpoint(
  target = c(0.9, 1),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Calculate the next best dose.
dose_recommendation <- nextBest(
  nextBest = de_next_best,
  doselimit = next_max_dose,
  samples = my_samples,
  model = my_model,
  data = my_data
)

# See the probabilities.
dose_recommendation$probs

# Joint plot.
print(dose_recommendation$plot)

# Show customization of single plot.
variant1 <- dose_recommendation$singlePlots$plot1 + xlim(0, 20)
print(variant1)

# Example of usage for `NextBestTD` NextBest class.
my_data <- Data(
  x = c(25, 50, 50, 75, 150, 200, 225, 300),
  y = c(0, 0, 0, 0, 1, 1, 1, 1),
  ID = 1:8,
  cohort = c(1L, 2L, 2L, 3L, 4L, 5L, 6L, 7L),
  doseGrid = seq(from = 25, to = 300, by = 25)
)

my_model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = my_data
)

# Target probabilities of the occurrence of a DLT during trial and
# at the end of the trial are defined as 0.35 and 0.3, respectively.
```

```
td_next_best <- NextBestTD(prob_target_drt = 0.35, prob_target_eot = 0.3)

# doselimit is the maximum allowable dose level to be given to subjects.
dose_recommendation <- nextBest(
  nextBest = td_next_best,
  doselimit = max(my_data@doseGrid),
  model = my_model,
  data = my_data
)

dose_recommendation$next_dose_drt
dose_recommendation$plot
# Example of usage for `NextBestTDsamples` NextBest class.
my_data <- Data(
  x = c(25, 50, 50, 75, 150, 200, 225, 300),
  y = c(0, 0, 0, 0, 1, 1, 1, 1),
  ID = 1:8,
  cohort = c(1L, 2L, 2L, 3L, 4L, 5L, 6L, 7L),
  doseGrid = seq(from = 25, to = 300, by = 25)
)

my_model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = my_data
)

# Set-up some MCMC parameters and generate samples.
my_options <- McmcOptions(burnin = 100, step = 2, samples = 800)
my_samples <- mcmc(my_data, my_model, my_options)

# Target probabilities of the occurrence of a DLT during trial and
# at the end of the trial are defined as 0.35 and 0.3, respectively.
# 'derive' is specified such that the 30% posterior quantile of the TD35 and
# TD30 samples will be used as TD35 and TD30 estimates.
tds_next_best <- NextBestTDsamples(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3,
  derive = function(samples) {
    as.numeric(quantile(samples, probs = 0.3))
  }
)

# doselimit is the maximum allowable dose level to be given to subjects.
dose_recommendation <- nextBest(
  nextBest = tds_next_best,
  doselimit = max(my_data@doseGrid),
  samples = my_samples,
  model = my_model,
  data = my_data
)
```

```
dose_recommendation$next_dose_drt
dose_recommendation$plot

# Example of usage for `NextBestMaxGain` NextBest class.

# Create the data.
my_data <- DataDual(
  x = c(25, 50, 25, 50, 75, 300, 250, 150),
  y = c(0, 0, 0, 0, 0, 1, 1, 0),
  ID = 1:8,
  cohort = 1:8,
  w = c(0.31, 0.42, 0.59, 0.45, 0.6, 0.7, 0.6, 0.52),
  doseGrid = seq(25, 300, 25),
  placebo = FALSE
)

# 'ModelTox' DLT model, e.g 'LogisticIndepBeta'.
my_model_dlt <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = my_data
)

# 'ModelEff' efficacy model, e.g. 'Effloglog'.
my_model_eff <- Effloglog(
  eff = c(1.223, 2.513),
  eff_dose = c(25, 300),
  nu = c(a = 1, b = 0.025),
  data = my_data
)

# Target probabilities of the occurrence of a DLT during trial and at the
# end of trial are defined as 0.35 and 0.3, respectively.
mg_next_best <- NextBestMaxGain(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3
)

# doselimit is the maximum allowable dose level to be given to subjects.
dose_recommendation <- nextBest(
  nextBest = mg_next_best,
  doselimit = 300,
  model = my_model_dlt,
  model_eff = my_model_eff,
  data = my_data
)

dose_recommendation$next_dose
dose_recommendation$plot

# Example of usage for `NextBestMaxGainSamples` NextBest class.
```

```
# Create the data.
my_data <- DataDual(
  x = c(25, 50, 25, 50, 75, 300, 250, 150),
  y = c(0, 0, 0, 0, 0, 1, 1, 0),
  w = c(0.31, 0.42, 0.59, 0.45, 0.6, 0.7, 0.6, 0.52),
  ID = 1:8,
  cohort = 1:8,
  doseGrid = seq(25, 300, 25),
  placebo = FALSE
)

# 'ModelTox' DLT model, e.g 'LogisticIndepBeta'.
my_model_dlt <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = my_data
)

# 'ModelEff' efficacy model, e.g 'Effloglog'.
my_model_effll <- Effloglog(
  eff = c(1.223, 2.513),
  eff_dose = c(25, 300),
  nu = c(a = 1, b = 0.025),
  data = my_data
)

# Set-up some MCMC parameters and generate samples from the posterior.
my_options <- McmcOptions(burnin = 100, step = 2, samples = 500)
my_samples_dlt <- mcmc(my_data, my_model_dlt, my_options)
my_samples_effll <- mcmc(my_data, my_model_effll, my_options)

# Target probabilities of the occurrence of a DLT during trial and at the end of
# trial are defined as 0.35 and 0.3, respectively.
# Use 30% posterior quantile of the TD35 and TD30 samples as estimates of TD35
# and TD30.
# Use 50% posterior quantile of the Gstar (the dose which gives the maxim gain value)
# samples as Gstar estimate.
mgs_next_best <- NextBestMaxGainSamples(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3,
  derive = function(samples) {
    as.numeric(quantile(samples, prob = 0.3))
  },
  mg_derive = function(mg_samples) {
    as.numeric(quantile(mg_samples, prob = 0.5))
  }
)

dose_recommendation <- nextBest(
  nextBest = mgs_next_best,
  doselimit = max(my_data@doseGrid),
  samples = my_samples_dlt,
```

```
  model = my_model_dlt,
  data = my_data,
  model_eff = my_model_effll,
  samples_eff = my_samples_effll
)

dose_recommendation$next_dose
dose_recommendation$plot

# Now using the 'EffFlexi' class efficacy model:

my_model_effflexi <- EffFlexi(
  eff = c(1.223, 2.513),
  eff_dose = c(25, 300),
  sigma2W = c(a = 0.1, b = 0.1),
  sigma2betaW = c(a = 20, b = 50),
  rw1 = FALSE,
  data = my_data
)

my_samples_effflexi <- mcmc(my_data, my_model_effflexi, my_options)

dose_recommendation <- nextBest(
  nextBest = mgs_next_best,
  doselimit = max(my_data@doseGrid),
  samples = my_samples_dlt,
  model = my_model_dlt,
  data = my_data,
  model_eff = my_model_effflexi,
  samples_eff = my_samples_effflexi
)

dose_recommendation$next_dose
dose_recommendation$plot
# Example of usage for `NextBestProbMTDLTE` NextBest class.

# Create the data.
my_data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  ID = 1:8,
  cohort = c(0, 1, 2, 3, 4, 5, 5, 5),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)

# Initialize the CRM model used to model the data.
my_model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 56
)

# Set-up some MCMC parameters and generate samples from the posterior.
```

```
my_options <- McmcOptions(burnin = 100, step = 2, samples = 500)
my_samples <- mcmc(my_data, my_model, my_options)

# Define the rule for dose increments and calculate the maximum dose allowed.
my_increments <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)
next_max_dose <- maxDose(my_increments, data = my_data)

# Define the rule which will be used to select the next best dose
# based on the 'NextBestProbMTDLTE' class.
nb_mtd_lte <- NextBestProbMTDLTE(target = 0.33)

# Calculate the next best dose.
dose_recommendation <- nextBest(
  nextBest = nb_mtd_lte,
  doselimit = next_max_dose,
  samples = my_samples,
  model = my_model,
  data = my_data
)
# Example of usage for `NextBestProbMTDMinDist` NextBest class.

# Create the data.
my_data <- Data(
  x = c(1.5, 1.5, 1.5, 2.5, 2.5, 2.5, 3.5, 3.5, 3.5),
  y = c(0, 0, 0, 0, 0, 0, 1, 1, 0),
  ID = 1:9,
  cohort = c(1, 1, 1, 2, 2, 2, 3, 3, 3),
  doseGrid = c(1.5, 2.5, 3.5, 4.5, 6, 7)
)

# Initialize the CRM model used to model the data.
my_model <- my_model <- LogisticKadaneBetaGamma(
  theta = 0.3,
  xmin = 1.5,
  xmax = 7,
  alpha = 1,
  beta = 19,
  shape = 0.5625,
  rate = 0.125
)

# Set-up some MCMC parameters and generate samples from the posterior.
my_options <- McmcOptions(burnin = 100, step = 2, samples = 500)
my_samples <- mcmc(my_data, my_model, my_options)

# Define the rule for dose increments and calculate the maximum dose allowed.
my_increments <- IncrementsDoseLevels(levels = 1)

next_max_dose <- maxDose(my_increments, data = my_data)
```

```
# Define the rule which will be used to select the next best dose
# based on the 'NextBestProbMTDMinDist' class.
nb_mtd_min_dist <- NextBestProbMTDMinDist(target = 0.3)

# Calculate the next best dose.
dose_recommendation <- nextBest(
  nextBest = nb_mtd_min_dist,
  doselimit = next_max_dose,
  samples = my_samples,
  model = my_model,
  data = my_data
)
ordinal_data <- .DefaultDataOrdinal()
ordinal_model <- .DefaultLogisticLogNormalOrdinal()
options <- .DefaultMcmcOptions()
ordinal_samples <- mcmc(ordinal_data, ordinal_model, options)

nextBest(
  nextBest = NextBestOrdinal(2L, .DefaultNextBestNCRM()),
  samples = ordinal_samples,
  doselimit = Inf,
  model = ordinal_model,
  data = ordinal_data
)
ordinal_data <- .DefaultDataOrdinal()
ordinal_model <- .DefaultLogisticLogNormalOrdinal()
options <- .DefaultMcmcOptions()
ordinal_samples <- mcmc(ordinal_data, ordinal_model, options)

nextBest(
  nextBest = NextBestOrdinal(2L, .DefaultNextBestNCRM()),
  samples = ordinal_samples,
  doselimit = Inf,
  model = ordinal_model,
  data = ordinal_data
)
```

`NextBest-class` `NextBest`

Description

[Stable]

`NextBest` is a virtual class for finding next best dose, from which all other specific next best dose classes inherit.

Usage

```
.DefaultNextBest()
```

Note

Typically, end users will not use the `DefaultNextBest()` function.

See Also

`NextBestMTD`, `NextBestNCRM`, `NextBestDualEndpoint`, `NextBestThreePlusThree`, `NextBestDualEndpoint`, `NextBestMinDist`, `NextBestInfTheory`, `NextBestTD`, `NextBestTDsamples`, `NextBestMaxGain`, `NextBestMaxGainSamples`.

NextBestDualEndpoint-class
NextBestDualEndpoint

Description

[Experimental]

`NextBestDualEndpoint` is the class for next best dose that is based on the dual endpoint model.

Usage

```
NextBestDualEndpoint(
  target,
  overdose,
  max_overdose_prob,
  target_relative = TRUE,
  target_thresh = 0.01
)

.DefaultNextBestDualEndpoint()
```

Arguments

`target`	(`numeric`) see slot definition.
`overdose`	(`numeric`) see slot definition.
`max_overdose_prob`	(`proportion`) see slot definition.
`target_relative`	(`flag`) see slot definition.
`target_thresh`	(`proportion`) see slot definition.

Details

Under this rule, at first admissible doses are found, which are those with toxicity probability to fall in `overdose` category and being below `max_overdose_prob`. Next, it picks (from the remaining admissible doses) the one that maximizes the probability to be in the `target` biomarker range. By default (`target_relative = TRUE`) the target is specified as relative to the maximum biomarker level across the dose grid or relative to the `Emax` parameter in case a parametric model was selected (i.e. `DualEndpointBeta`, `DualEndpointEmax`). However, if `target_relative = FALSE`, then the absolute biomarker range can be used as a target.

Slots

`target (numeric)`
: the biomarker target range that needs to be reached. For example, the target range $(0.8, 1.0)$ and `target_relative = TRUE` means that we target a dose with at least 80% of maximum biomarker level. As an other example, $(0.5, 0.8)$ would mean that we target a dose between 50% and 80% of the maximum biomarker level.

`overdose (numeric)`
: the overdose toxicity interval (lower limit excluded, upper limit included).

`max_overdose_prob (proportion)`
: maximum overdose probability that is allowed.

`target_relative (flag)`
: is `target` specified as relative? If `TRUE`, then the `target` is interpreted relative to the maximum, so it must be a probability range. Otherwise, the `target` is interpreted as absolute biomarker range.

`target_thresh (proportion)`
: a target probability threshold that needs to be fulfilled before the target probability will be used for deriving the next best dose (default to 0.01).

Note

Typically, end users will not use the `.DefaultNextBestDualEndpoint()` function.

Examples

```
# Target a dose achieving at least 0.9 of maximum biomarker level (efficacy)
# and with a probability below 0.25 that prob(DLT) > 0.35 (safety).
my_next_best <- NextBestDualEndpoint(
  target = c(0.9, 1),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Now, using absolute target on the natural biomarker scale.
my_next_best_absolute <- NextBestDualEndpoint(
  target = c(200, 300),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25,
  target_relative = FALSE
)
```

NextBestInfTheory-class NextBestInfTheory

Description

[Stable]

`NextBestInfTheory` is the class for next best dose that is based on information theory as proposed in https://doi.org/10.1002/sim.8450.

Usage

```
NextBestInfTheory(target, asymmetry)

.DefaultNextBestInfTheory()
```

Arguments

`target`	(`proportion`) see slot definition.
`asymmetry`	(`number`) see slot definition.

Slots

`target` (`proportion`)
: target toxicity probability, except 0 or 1.

`asymmetry` (`number`)
: value of the asymmetry exponent in the divergence function that describes the rate of penalization for overly toxic does. It must be a value from $(0, 2)$ interval.

Note

Typically, end users will not use the `.DefaultNextBestInfTheory()` function.

NextBestMaxGain-class NextBestMaxGain

Description

[Stable]

NextBestMaxGain is the class to find a next best dose with maximum gain value based on a pseudo DLT and efficacy models without samples. It is based solely on the probabilities of the occurrence of a DLT and the values of the mean efficacy responses obtained by using the modal estimates of the DLT and efficacy model parameters. There are two target probabilities of the occurrence of a DLT that must be specified: target probability to be used during the trial and target probability to be used at the end of the trial. It is suitable to use it only with the ModelTox model and ModelEff classes (except EffFlexi).

Usage

```
NextBestMaxGain(prob_target_drt, prob_target_eot)

.DefaultNextBestMaxGain()
```

Arguments

prob_target_drt	(proportion) see slot definition.
prob_target_eot	(proportion) see slot definition.

Slots

`prob_target_drt` (proportion)
: the target probability of the occurrence of a DLT to be used during the trial.

`prob_target_eot` (proportion)
: the target probability of the occurrence of a DLT to be used at the end of the trial.

Note

Typically, end users will not use the `.DefaultNextBestMaxGain()` function.

Examples

```
my_next_best <- NextBestMaxGain(0.35, 0.3)
```

NextBestMaxGainSamples-class
NextBestMaxGainSamples

Description

[Stable]

NextBestMaxGainSamples is the class to find a next best dose with maximum gain value based on a pseudo DLT and efficacy models and DLT and efficacy samples. There are two target probabilities of the occurrence of a DLT that must be specified: target probability to be used during the trial and target probability to be used at the end of the trial. It is suitable to use it only with the ModelTox model and ModelEff classes.

Usage

```
NextBestMaxGainSamples(prob_target_drt, prob_target_eot, derive, mg_derive)

.DefaultNextBestMaxGainSamples()
```

Arguments

prob_target_drt	(proportion) see slot definition in NextBestMaxGain.
prob_target_eot	(proportion) see slot definition in NextBestMaxGain.
derive	(function) see slot definition.
mg_derive	(function) see slot definition.

Slots

`derive` (function)
: derives, based on a vector of posterior dose samples, the target dose that has the probability of the occurrence of DLT equals to either the `prob_target_drt` or `prob_target_eot`. It must therefore accept one and only one argument, which is a numeric vector, and return a number.

`mg_derive` (function)
: derives, based on a vector of posterior dose samples that give the maximum gain value, the final next best estimate of the dose that gives the maximum gain value. It must therefore accept one and only one argument, which is a numeric vector, and return a number.

Note

Typically, end users will not use the `.DefaultNextBestMaxGainSamples()` function.

Examples

```
# Target probability of the occurrence of a DLT during the trial is set to 0.35.
# Target probability of the occurrence of a DLT at the end of the trial is set to 0.3.
# We want the use the 30% posterior quantile (the 30th percentile) of the TD35
# (the dose level with probability of the DLT equals 0.35) and TD30 samples.
# For `mg_derive` function (which takes the sample of doses which give the maximum
# gain), we will use the 50% posterior quantile (the median or th 50th percentile)
# of the sample.
my_next_best <- NextBestMaxGainSamples(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3,
  derive = function(samples) {
    as.numeric(quantile(samples, prob = 0.3))
  },
  mg_derive = function(mg_samples) {
    as.numeric(quantile(mg_samples, prob = 0.5))
  }
)
```

NextBestMinDist-class NextBestMinDist

Description

[Stable]

NextBestMinDist is the class for next best dose that is based on minimum distance to target probability.

Usage

```
NextBestMinDist(target)

.DefaultNextBestMinDist()
```

Arguments

target	(proportion) see slot definition.

Slots

target (proportion)
: single target toxicity probability, except 0 or 1.

Note

Typically, end users will not use the .DefaultNextBestMinDist() function.

Examples

```
# In the example below, the MTD is defined as the dose with the toxicity rate
# with minimal distance to the target of 30%.
next_best_min_dist <- NextBestMinDist(target = 0.3)
```

NextBestMTD-class NextBestMTD

Description

[Stable]

NextBestMTD is the class for next best dose based on MTD estimate.

Usage

```
NextBestMTD(target, derive)

.DefaultNextBestMTD()
```

Arguments

target	(proportion) see slot definition.
derive	(function) see slot definition.

Slots

target (proportion)
: target toxicity probability, except 0 or 1.

derive (function)
: a function which derives the final next best MTD estimate, based on vector of posterior MTD samples. It must therefore accept one and only one argument, which is a numeric vector, and return a number.

Note

Typically, end users will not use the .DefaultNextBestMTD() function.

Examples

```
# In the example below, the MTD is defined as the dose for which prob(DLE) = 0.33
# and we will use the 25th quantile of the posterior of MTD as our next best dose.
next_best_mtd <- NextBestMTD(
  target = 0.33,
  derive = function(mtd_samples) {
    quantile(mtd_samples, probs = 0.25)
  }
)
```

NextBestNCRM-class NextBestNCRM

Description

[Stable]

NextBestNCRM is the class for next best dose that finds the next dose with high posterior probability to be in the target toxicity interval.

Usage

```
NextBestNCRM(target, overdose, max_overdose_prob)

.DefaultNextBestNCRM()
```

Arguments

target	(numeric) see slot definition.
overdose	(numeric) see slot definition.
max_overdose_prob	(proportion) see slot definition.

Details

To avoid numerical problems, the dose selection algorithm has been implemented as follows: First admissible doses are found, which are those with probability to fall in overdose category being below max_overdose_prob. Next, within the admissible doses, the maximum probability to fall in the target category is calculated. If that is above 5% (i.e. it is not just numerical error), then the corresponding dose is the next recommended dose. Otherwise, the highest admissible dose is the next recommended dose.

Slots

target (numeric)
: the target toxicity interval (limits included).

overdose (numeric)
: the overdose toxicity interval (lower limit excluded, upper limit included). It is used to filter probability samples.

max_overdose_prob (proportion)
: maximum overdose posterior probability that is allowed, except 0 or 1.

Note

Typically, end users will not use the .DefaultNextBestNCRM() function.

Examples

```
# In the example below, the target toxicity interval [0.2, 0.35] while the
# overdose interval is (0.35,1]. Finally we would like to constrain the
# probability of overdosing below 25%.
my_next_best <- NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)
```

NextBestNCRMLoss-class

NextBestNCRMLoss

Description

[Stable]

NextBestNCRMLoss is the class based on NCRM rule and loss function. This class is similar to NextBestNCRM class, but differences are the addition of loss function and re-defined toxicity intervals, see each toxicity interval documentation and the note for details. As in NCRM rule, first admissible doses are found, which are those with probability to fall in overdose category being below `max_overdose_prob`. Next, within the admissible doses, the loss function is calculated, i.e. `losses %*% target`. Finally, the corresponding dose with lowest loss function (Bayes risk) is recommended for the next dose.

Usage

```
NextBestNCRMLoss(
  target,
  overdose,
  unacceptable = c(1, 1),
  max_overdose_prob,
  losses
)

.DefaultNextBestNCRMLoss()
```

Arguments

`target`	(`numeric`) see slot definition.
`overdose`	(`numeric`) see slot definition.
`unacceptable`	(`numeric`) see slot definition.

`max_overdose_prob`
: (`proportion`)
see slot definition in `NextBestNCRM`.

`losses`
: (`numeric`)
see slot definition.

Slots

`target` (`numeric`)
: the target toxicity interval (limits included). It has to be a probability range excluding 0 and 1.

`overdose` (`numeric`)
: the overdose toxicity interval (lower limit excluded, upper limit included) or the excessive toxicity interval (lower limit excluded, upper limit included) if unacceptable is not provided. It has to be a probability range. It is used to filter probability samples.

`unacceptable` (`numeric`)
: an unacceptable toxicity interval (lower limit excluded, upper limit included). This must be specified if the `overdose` does not include 1. Otherwise, it is `c(1, 1)` (default), which is essentially a scalar equals 1. It has to be a probability range.

`losses` (`numeric`)
: a vector specifying the loss function. If the `unacceptable` is provided, the vector length must be 4, otherwise 3.

Note

The loss function should be a vector of either 3 or 4 values. This is because the loss function values must be specified for each interval, that is under-dosing, target toxicity, and overdosing toxicity or under-dosing, target toxicity, overdosing (excessive) toxicity, and unacceptable toxicity intervals.

Typically, end users will not use the `.DefaultNextBestnCRMLoss()` function.

Examples

```
# In the example below, the target toxicity interval [0.2, 0.35] while the
# overdose interval is (0.35, 1]. We would like to constrain the probability
# of overdosing below 25%. The loss function is c(1, 0, 1, 2).
my_next_best <- NextBestNCRMLoss(
  target = c(0.2, 0.35),
  overdose = c(0.35, 0.6),
  unacceptable = c(0.6, 1),
  max_overdose_prob = 0.25,
  losses = c(1, 0, 1, 2)
)
```

NextBestOrdinal-class NextBestOrdinal

Description

[Experimental]

NextBestOrdinal is the class for applying a standard NextBest rule to the results of an ordinal CRM trial.

Usage

```
NextBestOrdinal(grade, rule)

.DefaultNextBestOrdinal()
```

Arguments

grade	(numeric) see slot definition.
rule	(NextBest) see slot definition.

Slots

grade (integer)
: the toxicity grade to which the rule should be applied.

rule (NextBest)
: the standard NextBest rule to be applied

Note

Typically, end users will not use the .DefaultNextBestOrdinal() function.

Examples

```
NextBestOrdinal(
  grade = 1L,
  rule = NextBestMTD(
    0.25,
    function(mtd_samples) {
      quantile(mtd_samples, probs = 0.25)
    }
  )
)
```

NextBestProbMTDLTE-class
NextBestProbMTDLTE

Description

[Experimental]

NextBestProbMTDLTE is the class of finding a next best dose that selects the dose with the highest probability of having a toxicity rate less or equal to the toxicity target. The dose is determined by calculating the posterior toxicity probability for each dose per iteration and select the maximum dose that has a toxicity probability below or equal to the target. The dose with the highest frequency of being selected as MTD across iterations is the next best dose. Placebo is not considered in the calculation and removed from the dose grid for any calculations.

Usage

```
NextBestProbMTDLTE(target)

.DefaultNextBestProbMTDLTE()
```

Arguments

target	(numeric) see slot definition.

Slots

target (numeric)
: the target toxicity probability.

Note

Typically, end users will not use the .DefaultNextBestProbMTDLTE() function.

Examples

```
# In the example below, the MTD is defined as the dose with the highest
# probability of having a toxicity rate below 30%.
next_best_prob_mtd_lte <- NextBestProbMTDLTE(target = 0.3)
```

NextBestProbMTDMinDist-class
NextBestProbMTDMinDist

Description

[Experimental]

NextBestProbMTDMinDist is the class of finding a next best dose that selects the dose with the highest probability of having a toxicity rate with the smallest distance to the toxicity target. The dose is determined by calculating the posterior toxicity probability for each dose per iteration and select the dose that has the smallest toxicity probability distance to the target. The dose with the highest frequency of being selected as MTD across iterations is the next best dose. Placebo is not considered as the next dose and for that reason not used in calculations. I.e. for placebo the toxicity probability distance to target is not calculated and taken into account for determination of the next dose.

Usage

```
NextBestProbMTDMinDist(target)

.DefaultNextBestProbMTDMinDist()
```

Arguments

target	(numeric) see slot definition.

Slots

target (numeric)
: the target toxicity probability.

Note

Typically, end users will not use the .DefaultNextBestProbMTDMinDist() function.

Examples

```
# In the example below, the MTD is defined as the dose with the highest
# probability of having a toxicity rate with minimal distance
# to the target of 30%.
next_best_prob_mtd_min_dist <- NextBestProbMTDMinDist(target = 0.3)
```

NextBestTD-class NextBestTD

Description

[Stable]

NextBestTD is the class to find a next best dose based on pseudo DLT model without samples. Namely, it is to find two next best doses, one for allocation during the trial and the second for final recommendation at the end of a trial without involving any samples, i.e. only DLT responses will be incorporated for the dose-allocation. This is based solely on the probabilities of the occurrence of a DLT obtained by using the modal estimates of the model parameters. There are two target probabilities of the occurrence of a DLT that must be specified: target probability to be used during the trial and target probability to be used at the end of the trial. It is suitable to use it only with the ModelTox model class.

Usage

```
.DefaultNextBestTD()

NextBestTD(prob_target_drt, prob_target_eot)
```

Arguments

prob_target_drt
: (proportion)
see slot definition.

prob_target_eot
: (proportion)
see slot definition.

Slots

prob_target_drt (proportion)
: the target probability (except 0 or 1) of the occurrence of a DLT to be used during the trial.

prob_target_eot (proportion)
: the target probability (except 0 or 1) of the occurrence of a DLT to be used at the end of the trial.

Note

Typically, end users will not use the .DefaultNextBestTD() function.

Examples

```
my_next_best <- NextBestTD(0.35, 0.3)
```

NextBestTDsamples-class
NextBestTDsamples

Description

[Stable]

NextBestTDsamples is the class to find a next best dose based on Pseudo DLT model with samples. Namely, it is to find two next best doses, one for allocation during the trial and the second for final recommendation at the end of a trial. Hence, there are two target probabilities of the occurrence of a DLT that must be specified: target probability to be used during the trial and target probability to be used at the end of the trial.

Usage

```
NextBestTDsamples(prob_target_drt, prob_target_eot, derive)

.DefaultNextBestTDsamples()
```

Arguments

prob_target_drt	(proportion) see slot definition in NextBestTD.
prob_target_eot	(proportion) see slot definition in NextBestTD.
derive	(function) see slot definition.

Slots

`derive` (function)
: derives, based on a vector of posterior dose samples, the target dose that has the probability of the occurrence of DLT equals to either the `prob_target_drt` or `prob_target_eot`. It must therefore accept one and only one argument, which is a numeric vector, and return a number.

Note

Typically, end users will not use the `.DefaultNextBestTDsamples()` function.

Examples

```
# Target probability of the occurrence of a DLT during the trial is set to 0.35.
# Target probability of the occurrence of a DLT at the end of the trial is set to 0.3.
# We want the use the 30% posterior quantile (the 30th percentile) of the TD35
# (the dose level with probability of the DLT equals 0.35) and TD30 samples.
```

```
my_next_best <- NextBestTDsamples(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3,
  derive = function(samples) {
    as.numeric(quantile(samples, probs = 0.3))
  }
)
```

`NextBestThreePlusThree-class` `NextBestThreePlusThree`

Description

[Stable]

`NextBestThreePlusThree` is the class for next best dose that implements the classical 3+3 dose recommendation. No input is required, hence this class has no slots.

Usage

```
NextBestThreePlusThree()

.DefaultNextBestThreePlusThree()
```

Note

Typically, end users will not use the `.DefaultNextBestThreePlusThree()` function.

Examples

```
# Next best dose class object using the classical 3+3 design.
my_next_best <- NextBestThreePlusThree()
```

`ngrid` *Number of Doses in Grid*

Description

[Stable]

A function that gets the number of doses in grid. User can choose whether the placebo dose (if any) should be counted or not.

Usage

```
ngrid(object, ignore_placebo = TRUE, ...)

## S4 method for signature 'Data'
ngrid(object, ignore_placebo = TRUE, ...)
```

Arguments

object	(Data) object with dose grid.
ignore_placebo	(flag) should placebo dose (if any) not be counted?
...	further arguments passed to class-specific methods.

Value

integer the number of doses in grid.

Examples

```
my_data <- Data(
  x = c(10, 50, 90, 100, 0.001, 20, 30, 30),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  ID = 1:8,
  cohort = c(1L, 2L, 3L, 4L, 5L, 5L, 6L, 6L),
  doseGrid = c(0.001, seq(from = 10, to = 100, by = 10)),
  placebo = TRUE
)
ngrid(my_data)
ngrid(my_data, ignore_placebo = FALSE)
```

OneParExpPrior-class OneParExpPrior

Description

[Experimental]

OneParExpPrior is the class for a standard CRM with an exponential prior on the power parameter for the skeleton prior probabilities. It is an implementation of a version of the one-parameter CRM (O'Quigley et al. 1990).

Usage

```
OneParExpPrior(skel_probs, dose_grid, lambda)

.DefaultOneParExpPrior()
```

Arguments

skel_probs	see slot definition.
dose_grid	(numeric) dose grid. It must be must be a sorted vector of the same length as skel_probs.
lambda	see slot definition.

Slots

`skel_fun` (`function`)
: function to calculate the prior DLT probabilities.

`skel_fun_inv` (`function`)
: inverse function of `skel_fun`.

`skel_probs` (`numeric`)
: skeleton prior probabilities. This is a vector of unique and sorted probability values between 0 and 1.

`lambda` (`number`)
: rate parameter of prior exponential distribution for theta.

Note

Typically, end users will not use the `.DefaultOneparExpPrior()` function.

Typically, end users will not use the `.DefaultOneParLogNormalPrior()` function.

Examples

```
my_model <- OneParExpPrior(
  skel_probs = c(0.1, 0.3, 0.5, 0.7, 0.9),
  dose_grid = 1:5,
  lambda = 2
)
```

OneParLogNormalPrior-class
OneParLogNormalPrior

Description

[Stable]

`OneParLogNormalPrior` is the class for a standard CRM with a normal prior on the log power parameter for the skeleton prior probabilities.

Usage

```
OneParLogNormalPrior(skel_probs, dose_grid, sigma2)

.DefaultOneParLogNormalPrior()
```

Arguments

skel_probs	(numeric) skeleton prior probabilities. This is a vector of unique and sorted probability values between 0 and 1.
dose_grid	(numeric) dose grid. It must be must be a sorted vector of the same length as skel_probs.
sigma2	(number) prior variance of log power parameter alpha.

Value

an instance of the OneParLogNormalPrior class

Slots

skel_fun (function)
: function to calculate the prior DLT probabilities.

skel_fun_inv (function)
: inverse function of skel_fun.

skel_probs (numeric)
: skeleton prior probabilities. This is a vector of unique and sorted probability values between 0 and 1.

sigma2 (number)
: prior variance of log power parameter alpha.

See Also

ModelLogNormal.

Examples

```
my_model <- OneParLogNormalPrior(
  skel_probs = seq(from = 0.1, to = 0.9, length = 5),
  dose_grid = 1:5,
  sigma2 = 2
)
```

---

or-Stopping-Stopping *The method combining two atomic stopping rules*

Description

The method combining two atomic stopping rules

Usage

```
## S4 method for signature 'Stopping,Stopping'
e1 | e2
```

Arguments

e1	First Stopping object
e2	Second Stopping object

Value

The StoppingAny object

Examples

```
## Example of combining two atomic stopping rules with an OR ('|') operator

myStopping1 <- StoppingMinCohorts(nCohorts=3)
myStopping2 <- StoppingTargetProb(target=c(0.2, 0.35),
                                  prob=0.5)

myStopping <- myStopping1 | myStopping2
```

---

or-Stopping-StoppingAny

The method combining a stopping list and an atomic

Description

The method combining a stopping list and an atomic

Usage

```
## S4 method for signature 'StoppingAny,Stopping'
e1 | e2
```

Arguments

e1	StoppingAny object
e2	Stopping object

Value

The modified StoppingAny object

Examples

```
## Example of combining a list of stopping rules with an atomic stopping rule
## with an OR ('|') operator

myStopping1 <- StoppingMinCohorts(nCohorts=3)
myStopping2 <- StoppingTargetProb(target=c(0.2, 0.35),
                                  prob=0.5)

myStopping3 <- StoppingMinPatients(nPatients=20)

myStopping <- (myStopping1 & myStopping2 ) | myStopping3
```

---

or-StoppingAny-Stopping

The method combining an atomic and a stopping list

Description

The method combining an atomic and a stopping list

Usage

```
## S4 method for signature 'Stopping,StoppingAny'
e1 | e2
```

Arguments

e1	Stopping object
e2	StoppingAny object

Value

The modified StoppingAny object

Examples

```
## Example of combining an atomic stopping rule with a list of stopping rules
## with an OR ('|') operator

myStopping1 <- StoppingMinCohorts(nCohorts=3)
myStopping2 <- StoppingTargetProb(target=c(0.2, 0.35),
                                  prob=0.5)

myStopping3 <- StoppingMinPatients(nPatients=20)
```

```
myStopping <-  myStopping3 | (myStopping1 & myStopping2 )
```

`plot,Data,ModelTox-method`

Plot of the fitted dose-tox based with a given pseudo DLE model and data without samples

---

Description

Plot of the fitted dose-tox based with a given pseudo DLE model and data without samples

Usage

```
## S4 method for signature 'Data,ModelTox'
plot(
  x,
  y,
  xlab = "Dose level",
  ylab = "Probability of DLE",
  showLegend = TRUE,
  ...
)
```

Arguments

`x`	the data of `Data` class object
`y`	the model of the `ModelTox` class object
`xlab`	the x axis label
`ylab`	the y axis label
`showLegend`	should the legend be shown? (default)
`...`	not used

Value

This returns the `ggplot` object for the dose-DLE model plot

Examples

```
## plot the dose-DLE curve given a pseudo DLE model using data without samples
## data must be of 'Data' class
## define the data
data <- Data(
  x = c(25, 50, 50, 75, 100, 100, 225, 300),
  y = c(0, 0, 0, 0, 1, 1, 1, 1),
  ID = 1L:8L,
  cohort = as.integer(c(1, 2, 2, 3, 4, 4, 5, 6)),
  doseGrid = seq(25, 300, 25)
)
## model must be from 'ModelTox' class e.g 'LogisticIndepBeta' class model
## define the model (see LogisticIndepBeta example)
model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = data
)
## plot the dose-DLE curve
## 'x' is the data and 'y' is the model in plot
plot(x = data, y = model)
```

---

plot,DataDA,missing-method

Plot Method for the DataDA Class

Description

[Stable]

A method that creates a plot for DataDA object.

Usage

```
## S4 method for signature 'DataDA,missing'
plot(x, y, blind = FALSE, ...)
```

Arguments

x	(DataDA) object we want to plot.
y	(missing) missing object, for compatibility with the generic function.
blind	(flag) indicates whether to blind the data. If TRUE, then placebo subjects are reported at the same level as the active dose level in the corresponding cohort, and DLTs are always assigned to the first subjects in a cohort.
...	passed to the first inherited method plot after this current method.

Value

The ggplot2 object.

Examples

```
# Create some data of class 'DataDA'.
my_data <- DataDA(
  u = c(42, 30, 15, 5, 20, 25, 30, 60),
  t0 = c(0, 15, 30, 40, 55, 70, 75, 85),
  Tmax = 60,
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 1, 1, 0, 0, 1, 0),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)

# Plot the data.
plot(my_data)
```

---

plot,DataDual,missing-method

Plot Method for the DataDual Class

Description

[Stable]

A method that creates a plot for DataDual object.

Usage

```
## S4 method for signature 'DataDual,missing'
plot(x, y, blind = FALSE, ...)
```

Arguments

x	(DataDual) object we want to plot.
y	(missing) missing object, for compatibility with the generic function.
blind	(flag) indicates whether to blind the data. If TRUE, then placebo subjects are reported at the same level as the active dose level in the corresponding cohort, and DLTs are always assigned to the first subjects in a cohort.
...	passed to the first inherited method plot after this current method.

Value

The ggplot2 object.

Examples

```
# Create some data of class 'DataDual'.
my_data <- DataDual(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  w = rnorm(8),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)

# Plot the data.
plot(my_data)
```

---

plot,DataDual,ModelEff-method

Plot of the fitted dose-efficacy based with a given pseudo efficacy model and data without samples

---

Description

Plot of the fitted dose-efficacy based with a given pseudo efficacy model and data without samples

Usage

```
## S4 method for signature 'DataDual,ModelEff'
plot(
  x,
  y,
  ...,
  xlab = "Dose level",
  ylab = "Expected Efficacy",
  showLegend = TRUE
)
```

Arguments

x	the data of DataDual class object
y	the model of the ModelEff class object
...	not used
xlab	the x axis label
ylab	the y axis label
showLegend	should the legend be shown? (default)

Value

This returns the ggplot object for the dose-efficacy model plot

Examples

```
# nolint start

##plot the dose-efficacy curve given a pseudo efficacy model using data without samples
##data must be of 'DataDual' class
##define the data
data<-DataDual(x=c(25,50,50,75,100,100,225,300),y=c(0,0,0,0,1,1,1,1),
               w=c(0.31,0.42,0.59,0.45,0.6,0.7,0.6,0.52),
               doseGrid=seq(25,300,25),placebo=FALSE)
##model must be from 'ModelEff' class e.g 'Effloglog' class model
##define the model (see Effloglog example)
Effmodel<-Effloglog(eff=c(1.223,2.513),eff_dose=c(25,300),nu=c(a=1,b=0.025),data=data)
## plot the dose-efficacy curve
## 'x' is the data and 'y' is the model in plot
plot(x=data,y=Effmodel)

# nolint end
```

`plot,DualSimulations,missing-method`

Plot dual-endpoint simulations

Description

This plot method can be applied to `DualSimulations` objects in order to summarize them graphically. In addition to the standard plot types, there is

sigma2W Plot a boxplot of the final biomarker variance estimates in the simulated trials

rho Plot a boxplot of the final correlation estimates in the simulated trials

Usage

```
## S4 method for signature 'DualSimulations,missing'
plot(x, y, type = c("trajectory", "dosesTried", "sigma2W", "rho"), ...)
```

Arguments

`x`	the `DualSimulations` object we want to plot from
`y`	missing
`type`	the type of plots you want to obtain.
`...`	not used

Value

A single `ggplot` object if a single plot is asked for, otherwise a `gridExtra{gTree}` object.

Examples

```
# Define the dose-grid.
emptydata <- DataDual(doseGrid = c(1, 3, 5, 10, 15, 20, 25, 40, 50, 80, 100))

# Create some data.
my_data <- DataDual(
  x = c(
    0.1, 0.5, 1.5, 3, 6, 10, 10, 10,
    20, 20, 20, 40, 40, 40, 50, 50, 50
  ),
  y = c(
    0, 0, 0, 0, 0, 0, 1, 0,
    0, 1, 1, 0, 0, 1, 0, 1, 1
  ),
  ID = 1:17,
  cohort = c(1L, 2L, 3L, 4L, 5L, 6L, 6L, 6L, 7L, 7L, 7L, 8L, 8L, 8L, 9L, 9L, 9L),
  w = c(
    0.31, 0.42, 0.59, 0.45, 0.6, 0.7, 0.55, 0.6,
    0.52, 0.54, 0.56, 0.43, 0.41, 0.39, 0.34, 0.38, 0.21
  ),
  doseGrid = c(
    0.1, 0.5, 1.5, 3, 6,
    seq(from = 10, to = 80, by = 2)
  )
)

# Initialize the CRM model.
my_model <- DualEndpointRW(
  mean = c(0, 1),
  cov = matrix(c(1, 0, 0, 1), nrow = 2),
  sigma2betaW = 0.01,
  sigma2W = c(a = 0.1, b = 0.1),
  rho = c(a = 1, b = 1),
  rw1 = TRUE
)

# Choose the rule for selecting the next dose.
my_next_best <- NextBestDualEndpoint(
  target = c(0.9, 1),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Choose the rule for the cohort-size
mySize1 <- CohortSizeRange(
  intervals = c(0, 30),
  cohort_size = c(1, 3)
)
mySize2 <- CohortSizeDLT(
  intervals = c(0, 1),
  cohort_size = c(1, 3)
)
```

```
mySize <- maxSize(mySize1, mySize2)

# Choose the rule for stopping
myStopping4 <- StoppingTargetBiomarker(
  target = c(0.9, 1),
  prob = 0.5
)
myStopping <- myStopping4 | StoppingMinPatients(40)

my_size1 <- CohortSizeRange(
  intervals = c(0, 30),
  cohort_size = c(1, 3)
)
my_size2 <- CohortSizeDLT(
  intervals = c(0, 1),
  cohort_size = c(1, 3)
)
my_size <- maxSize(my_size1, my_size2)

# Choose the rule for stopping
my_stopping4 <- StoppingTargetBiomarker(
  target = c(0.9, 1),
  prob = 0.5
)
my_stopping <- my_stopping4 | StoppingMinPatients(40) | StoppingMissingDose()

# Choose the rule for dose increments
my_increments <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

# Initialize the design

my_design <- DualDesign(
  model = my_model,
  data = emptydata,
  nextBest = my_next_best,
  stopping = my_stopping,
  increments = my_increments,
  cohort_size = CohortSizeConst(3),
  startingDose = 3
)

# Define scenarios for the TRUE toxicity and efficacy profiles.
beta_mod <- function(dose, e0, eMax, delta1, delta2, scal) {
  maxDens <- (delta1^delta1) * (delta2^delta2) / ((delta1 + delta2)^(delta1 + delta2))
  dose <- dose / scal
  e0 + eMax / maxDens * (dose^delta1) * (1 - dose)^delta2
}

true_biomarker <- function(dose) {
  beta_mod(dose, e0 = 0.2, eMax = 0.6, delta1 = 5, delta2 = 5 * 0.5 / 0.5, scal = 100)
```

```
}

true_tox <- function(dose) {
  pnorm((dose - 60) / 10)
}

# Draw the TRUE profiles
par(mfrow = c(1, 2))
curve(true_tox(x), from = 0, to = 80)
curve(true_biomarker(x), from = 0, to = 80)

# Run the simulation on the desired design.
# We only generate 1 trial outcome here for illustration, for the actual study.
# Also for illustration purpose, we will use 5 burn-ins to generate 20 samples,
# this should be increased of course.
my_sims <- simulate(
  object = my_design,
  trueTox = true_tox,
  trueBiomarker = true_biomarker,
  sigma2W = 0.01,
  rho = 0,
  nsim = 1,
  parallel = FALSE,
  seed = 9,
  startingDose = 6,
  mcmcOptions = McmcOptions(
    burnin = 1,
    step = 1,
    samples = 2
  )
)

# Plot the results of the simulation.
print(plot(my_sims))
```

---

plot,DualSimulationsSummary,missing-method

Plot summaries of the dual-endpoint design simulations

Description

This plot method can be applied to DualSimulationsSummary objects in order to summarize them graphically. Possible type of plots at the moment are those listed in plot,SimulationsSummary,missing-method plus:

meanBiomarkerFit Plot showing the average fitted dose-biomarker curve across the trials, together with 95% credible intervals, and comparison with the assumed truth (as specified by the trueBiomarker argument to summary,DualSimulations-method)

You can specify any subset of these in the type argument.

Usage

```
## S4 method for signature 'DualSimulationsSummary,missing'
plot(
  x,
  y,
  type = c("nObs", "doseSelected", "propDLTs", "nAboveTarget", "meanFit",
    "meanBiomarkerFit"),
  ...
)
```

Arguments

x	the DualSimulationsSummary object we want to plot from
y	missing
type	the types of plots you want to obtain.
...	not used

Value

A single ggplot object if a single plot is asked for, otherwise a gridExtra{gTree} object.

Examples

```
# Define the dose-grid.
emptydata <- DataDual(doseGrid = c(1, 3, 5, 10, 15, 20, 25, 40, 50, 80, 100))


# Initialize the CRM model.
my_model <- DualEndpointRW(
  mean = c(0, 1),
  cov = matrix(c(1, 0, 0, 1), nrow = 2),
  sigma2betaW = 0.01,
  sigma2W = c(a = 0.1, b = 0.1),
  rho = c(a = 1, b = 1),
  rw1 = TRUE
)


# Choose the rule for selecting the next dose.
my_next_best <- NextBestDualEndpoint(
  target = c(0.9, 1),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Choose the rule for the cohort-size.
my_size1 <- CohortSizeRange(
  intervals = c(0, 30),
  cohort_size = c(1, 3)
)
```

```
my_size2 <- CohortSizeDLT(
  intervals = c(0, 1),
  cohort_size = c(1, 3)
)
my_size <- maxSize(my_size1, my_size2)

# Choose the rule for stopping.
my_stopping4 <- StoppingTargetBiomarker(
  target = c(0.9, 1),
  prob = 0.5
)
# Only 10 patients here for illustration!
my_stopping <- my_stopping4 | StoppingMinPatients(10) | StoppingMissingDose()

# Choose the rule for dose increments.
my_increments <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

# Initialize the design.
my_design <- DualDesign(
  model = my_model,
  data = emptydata,
  nextBest = my_next_best,
  stopping = my_stopping,
  increments = my_increments,
  cohort_size = CohortSizeConst(3),
  startingDose = 3
)

# Define scenarios for the TRUE toxicity and efficacy profiles.
beta_mod <- function(dose, e0, eMax, delta1, delta2, scal) {
  maxDens <- (delta1^delta1) * (delta2^delta2) / ((delta1 + delta2)^(delta1 + delta2))
  dose <- dose / scal
  e0 + eMax / maxDens * (dose^delta1) * (1 - dose)^delta2
}



true_biomarker <- function(dose) {
  beta_mod(dose, e0 = 0.2, eMax = 0.6, delta1 = 5, delta2 = 5 * 0.5 / 0.5, scal = 100)
}

true_tox <- function(dose) {
  pnorm((dose - 60) / 10)
}

# Draw the TRUE profiles.
par(mfrow = c(1, 2))
curve(true_tox(x), from = 0, to = 80)
curve(true_biomarker(x), from = 0, to = 80)
```

```
# Run the simulation on the desired design.
# We only generate 1 trial outcome here for illustration, for the actual study.
# For illustration purpose we will use 5 burn-ins to generate 20 samples,
# this should be increased of course.
my_sims <- simulate(
  object = my_design,
  trueTox = true_tox,
  trueBiomarker = true_biomarker,
  sigma2W = 0.01,
  rho = 0,
  nsim = 1,
  parallel = FALSE,
  seed = 3,
  startingDose = 6,
  mcmcOptions = McmcOptions(
    burnin = 5,
    step = 1,
    samples = 20
  )
)

# Plot the summary of the Simulations.
plot(summary(my_sims,
  trueTox = true_tox,
  trueBiomarker = true_biomarker
))
```

---

plot,GeneralSimulations,missing-method

Plot simulations

Description

Summarize the simulations with plots

Usage

```
## S4 method for signature 'GeneralSimulations,missing'
plot(x, y, type = c("trajectory", "dosesTried"), ...)
```

Arguments

x	the GeneralSimulations object we want to plot from
y	missing
type	the type of plots you want to obtain.
...	not used

Details

This plot method can be applied to GeneralSimulations objects in order to summarize them graphically. Possible types of plots at the moment are:

trajectory Summary of the trajectory of the simulated trials

dosesTried Average proportions of the doses tested in patients

You can specify one or both of these in the type argument.

Value

A single ggplot object if a single plot is asked for, otherwise a gridExtra{gTree} object.

Examples

```
# nolint start

## obtain the plot for the simulation results
## If only DLE responses are considered in the simulations

## Specified your simulations when no DLE samples are used
## Define your data set first using an empty data set
## with dose levels from 25 to 300 with increments 25
data <- Data(doseGrid = seq(25, 300, 25))

## Specified the model of 'ModelTox' class eg 'LogisticIndepBeta' class model
model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = data
)
## Then the escalation rule
tdNextBest <- NextBestTD(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3
)

## The cohort size, size of 3 subjects
mySize <- CohortSizeConst(size = 3)
## Deifne the increments for the dose-escalation process
## The maximum increase of 200% for doses up to the maximum of the dose specified in the doseGrid
## The maximum increase of 200% for dose above the maximum of the dose specified in the doseGrid
## This is to specified a maximum of 3-fold restriction in dose-esclation
myIncrements <- IncrementsRelative(
  intervals = c(min(data@doseGrid), max(data@doseGrid)),
  increments = c(2, 2)
)
## Specified the stopping rule e.g stop when the maximum sample size of 12 patients has been reached
myStopping <- StoppingMinPatients(nPatients = 12)
## Now specified the design with all the above information and starting with a dose of 25
design <- TDDesign(
```

```
  model = model,
  nextBest = tdNextBest,
  stopping = myStopping,
  increments = myIncrements,
  cohort_size = mySize,
  data = data, startingDose = 25
)

## Specify the truth of the DLE responses
myTruth <- probFunction(model, phi1 = -53.66584, phi2 = 10.50499)

## Then specified the simulations and generate the trial
## For illustration purpose only 1 simulation is produced (nsim=1).
## The simulations
mySim <- simulate(design,
  args = NULL,
  truth = myTruth,
  nsim = 1,
  seed = 819,
  parallel = FALSE
)


## plot the simulations
print(plot(mySim))



## If DLE samples are involved
## The escalation rule
tdNextBest <- NextBestTDsamples(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3,
  derive = function(samples) {
    as.numeric(quantile(samples, probs = 0.3))
  }
)
## specify the design
design <- TDsamplesDesign(
  model = model,
  nextBest = tdNextBest,
  stopping = myStopping,
  increments = myIncrements,
  cohort_size = mySize,
  data = data, startingDose = 25
)
## options for MCMC
## The simulations
## For illustration purpose only 1 simulation is produced (nsim=1).
# mySim <- simulate(design,
#                   args=NULL,
#                   truth=myTruth,
#                   nsim=1,
```

```
#                   seed=819,
#                   mcmcOptions=options,
#                   parallel=FALSE)
#
# ##plot the simulations
# print(plot(mySim))
#

# nolint end
```

`plot,GeneralSimulationsSummary,missing-method`
Graphical display of the general simulation summary

Description

This plot method can be applied to `GeneralSimulationsSummary` objects in order to summarize them graphically. Possible types of plots at the moment are:

Usage

```
## S4 method for signature 'GeneralSimulationsSummary,missing'
plot(x, y, type = c("nObs", "doseSelected", "propDLTs", "nAboveTarget"), ...)
```

Arguments

`x`	the `GeneralSimulationsSummary` object we want to plot from
`y`	missing
`type`	the types of plots you want to obtain.
`...`	not used

Details

nObs Distribution of the number of patients in the simulated trials

doseSelected Distribution of the final selected doses in the trials. Note that this can include zero entries, meaning that the trial was stopped because all doses in the dose grid appeared too toxic.

propDLTs Distribution of the proportion of patients with DLTs in the trials

nAboveTarget Distribution of the number of patients treated at doses which are above the target toxicity interval (as specified by the `truth` and `target` arguments to `summary,GeneralSimulations-method`)

You can specify any subset of these in the `type` argument.

Value

A single `ggplot` object if a single plot is asked for, otherwise a `gridExtra{gTree}` object.

`plot,PseudoDualFlexiSimulations,missing-method`

This plot method can be applied to `PseudoDualFlexiSimulations` *objects in order to summarize them graphically. Possible* `type`*s of plots at the moment are:*

trajectory *Summary of the trajectory of the simulated trials*

dosesTried *Average proportions of the doses tested in patients*

sigma2 *The variance of the efficacy responses*

sigma2betaW *The variance of the random walk model*

You can specify one or both of these in the `type` *argument.*

Description

This plot method can be applied to `PseudoDualFlexiSimulations` objects in order to summarize them graphically. Possible `type`s of plots at the moment are:

trajectory Summary of the trajectory of the simulated trials

dosesTried Average proportions of the doses tested in patients

sigma2 The variance of the efficacy responses

sigma2betaW The variance of the random walk model

You can specify one or both of these in the `type` argument.

Usage

```
## S4 method for signature 'PseudoDualFlexiSimulations,missing'
plot(x, y, type = c("trajectory", "dosesTried", "sigma2", "sigma2betaW"), ...)
```

Arguments

`x`	the `PseudoDualFlexiSimulations` object we want to plot from
`y`	missing
`type`	the type of plots you want to obtain.
`...`	not used

Value

A single `ggplot` object if a single plot is asked for, otherwise a `gridExtra{gTree}` object.

Examples

```
# Obtain the plot for the simulation results if DLE and efficacy responses
# are considered in the simulations.
emptydata <- DataDual(doseGrid = seq(25, 300, 25))

# The DLE model must be of 'ModelTox' (e.g 'LogisticIndepBeta') class.
dle_model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = emptydata
)

# The efficacy model must be of 'EffFlexi' class.
eff_model <- EffFlexi(
  eff = c(1.223, 2.513),
  eff_dose = c(25, 300),
  sigma2W = c(a = 0.1, b = 0.1),
  sigma2betaW = c(a = 20, b = 50),
  rw1 = FALSE,
  data = emptydata
)

# The escalation rule using the 'NextBestMaxGainSamples' class.
my_next_best <- NextBestMaxGainSamples(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3,
  derive = function(samples) {
    as.numeric(quantile(samples, prob = 0.3))
  },
  mg_derive = function(mg_samples) {
    as.numeric(quantile(mg_samples, prob = 0.5))
  }
)

# The cohort size, size of 3 subjects.
my_size <- CohortSizeConst(size = 3)

# Allow increase of 200%.
my_increments <- IncrementsRelative(intervals = 0, increments = 2)

# Define the stopping rule. Stop when the maximum sample size of 36 patients has
# been reached or when the next dose is NA.
my_stopping <- StoppingMinPatients(nPatients = 36) | StoppingMissingDose()

# Specify the design.
design <- DualResponsesSamplesDesign(
  nextBest = my_next_best,
  cohort_size = my_size,
  startingDose = 25,
  model = dle_model,
  eff_model = eff_model,
```

```
  data = emptydata,
  stopping = my_stopping,
  increments = my_increments
)
# Specify the true DLE curve and the true expected efficacy values
# at all dose levels.
my_truth_dle <- probFunction(dle_model, phi1 = -53.66584, phi2 = 10.50499)

my_truth_eff <- c(
  -0.5478867, 0.1645417, 0.5248031, 0.7604467,
  0.9333009, 1.0687031, 1.1793942, 1.2726408,
  1.3529598, 1.4233411, 1.4858613, 1.5420182
)

# The true gain curve.
my_truth_gain <- function(dose) {
  return((myTruthEff(dose)) / (1 + (myTruthDLE(dose) / (1 - myTruthDLE(dose)))))
}

# MCMC options.
my_options <- McmcOptions(burnin = 10, step = 1, samples = 20)

# For illustration purpose only 1 simulation is produced.
my_sim <- simulate(
  object = design,
  args = NULL,
  trueDLE = my_truth_dle,
  trueEff = my_truth_eff,
  trueSigma2 = 0.025,
  trueSigma2betaW = 1,
  mcmcOptions = my_options,
  nsim = 1,
  seed = 819,
  parallel = FALSE
)

# Plot the simulated results.
print(plot(my_sim))
```

`plot,PseudoDualSimulations,missing-method`
Plot simulations

Description

Summarize the simulations with plots

Usage

```
## S4 method for signature 'PseudoDualSimulations,missing'
plot(x, y, type = c("trajectory", "dosesTried", "sigma2"), ...)
```

Arguments

x	the PseudoDualSimulations object we want to plot from
y	missing
type	the type of plots you want to obtain.
...	not used

Details

This plot method can be applied to PseudoDualSimulations objects in order to summarize them graphically. Possible types of plots at the moment are:

trajectory Summary of the trajectory of the simulated trials

dosesTried Average proportions of the doses tested in patients

sigma2 The variance of the efficacy responses

You can specify one or both of these in the type argument.

Value

A single ggplot object if a single plot is asked for, otherwise a gridExtra{gTree} object.

Examples

```
# Obtain the plot for the simulation results if DLE and efficacy responses
# are considered in the simulations.

# Example to run simulations when no samples are used. The data object
# must be defined with doses >= 1:
emptydata <- DataDual(doseGrid = seq(25, 300, 25), placebo = FALSE)

# The DLE model must be of 'ModelTox' (e.g 'LogisticIndepBeta') class.
dle_model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = emptydata
)

# The efficacy model must be of 'ModelEff' (e.g 'Effloglog') class.
eff_model <- Effloglog(
  eff = c(1.223, 2.513), eff_dose = c(25, 300),
  nu = c(a = 1, b = 0.025),
  data = emptydata
)

# The escalation rule using the 'NextBestMaxGain' class.
my_next_best <- NextBestMaxGain(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3
)
```

```
# Allow increase of 200%.
my_increments <- IncrementsRelative(intervals = 0, increments = 2)

# Cohort size of 3.
my_size <- CohortSizeConst(size = 3)

# Stop only when 36 subjects are treated or next dose is NA.
my_stopping <- StoppingMinPatients(nPatients = 36) | StoppingMissingDose()

# Now specify the design with all the above information and starting with a
# dose of 25 (for details please refer to the 'DualResponsesDesign' example).
my_design <- DualResponsesDesign(
  nextBest = my_next_best,
  model = dle_model,
  eff_model = eff_model,
  stopping = my_stopping,
  increments = my_increments,
  cohort_size = my_size,
  data = emptydata,
  startingDose = 25
)

# Specify the true DLE and efficacy curves.
my_truth_dle <- probFunction(dle_model, phi1 = -53.66584, phi2 = 10.50499)
my_truth_eff <- efficacyFunction(eff_model, theta1 = -4.818429, theta2 = 3.653058)

# Run simulations (for illustration purpose only 1 simulation is produced).
my_sim <- simulate(
  object = my_design,
  args = NULL,
  trueDLE = my_truth_dle,
  trueEff = my_truth_eff,
  trueNu = 1 / 0.025,
  nsim = 1,
  seed = 819,
  parallel = FALSE
)

# Plot the simulation results.
print(plot(my_sim))

# Example if DLE and efficacy samples are involved.
# The escalation rule using the 'NextBestMaxGainSamples' class.
my_next_best <- NextBestMaxGainSamples(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3,
  derive = function(samples) {
    as.numeric(quantile(samples, prob = 0.3))
  },
  mg_derive = function(mg_samples) {
    as.numeric(quantile(mg_samples, prob = 0.5))
  }
```

```
)

# The design of 'DualResponsesSamplesDesign' class.
my_design <- DualResponsesSamplesDesign(
  nextBest = my_next_best,
  cohort_size = my_size,
  startingDose = 25,
  model = dle_model,
  eff_model = eff_model,
  data = emptydata,
  stopping = my_stopping,
  increments = my_increments
)

# Options for MCMC.
my_options <- McmcOptions(burnin = 10, step = 1, samples = 20)

# For illustration purpose only 1 simulation is produced (nsim = 1).
my_sim <- simulate(
  object = my_design,
  args = NULL,
  trueDLE = my_truth_dle,
  trueEff = my_truth_eff,
  trueNu = 1 / 0.025,
  nsim = 1,
  mcmcOptions = my_options,
  seed = 819,
  parallel = FALSE
)

# Plot the simulation results.
print(plot(my_sim))
```

---

plot,PseudoDualSimulationsSummary,missing-method

Plot the summary of Pseudo Dual Simulations summary

Description

This plot method can be applied to PseudoDualSimulationsSummary objects in order to summarize them graphically. Possible type of plots at the moment are those listed in plot,PseudoSimulationsSummary,missing- plus:

meanEffFit Plot showing the fitted dose-efficacy curve. If no samples are involved, only the average fitted dose-efficacy curve across the trials will be plotted. If samples (DLE and efficacy) are involved, the average fitted dose-efficacy curve across the trials, together with the 95% credibility interval; and comparison with the assumed truth (as specified by the trueEff argument to summary,PseudoDualSimulations-method)

You can specify any subset of these in the type argument.

Usage

```
## S4 method for signature 'PseudoDualSimulationsSummary,missing'
plot(
  x,
  y,
  type = c("nObs", "doseSelected", "propDLE", "nAboveTargetEndOfTrial", "meanFit",
    "meanEffFit"),
  ...
)
```

Arguments

x	the PseudoDualSimulationsSummary object we want to plot from
y	missing
type	the types of plots you want to obtain.
...	not used

Value

A single ggplot object if a single plot is asked for, otherwise a gridExtra{gTree} object.

Examples

```
# Obtain the summary plot for the simulation results if DLE and efficacy
# responses are considered in the simulations.

# In the example when no samples are used a data object with doses >= 1
# needs to be defined.
emptydata <- DataDual(doseGrid = seq(25, 300, 25), placebo = FALSE)

# The DLE model must be of 'ModelTox' (e.g 'LogisticIndepBeta') class.
dle_model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = emptydata
)

# The efficacy model of 'ModelEff' (e.g 'Effloglog') class.
eff_model <- Effloglog(
  eff = c(1.223, 2.513),
  eff_dose = c(25, 300),
  nu = c(a = 1, b = 0.025),
  data = emptydata
)

# The escalation rule using the 'NextBestMaxGain' class.
my_next_best <- NextBestMaxGain(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3
```

```
)

# Allow increase of 200%.
my_increments <- IncrementsRelative(intervals = 0, increments = 2)

# Cohort size of 3.
my_size <- CohortSizeConst(size = 3)

# Stop when 10 subjects are treated (for illustration only).
my_stopping <- StoppingMinPatients(nPatients = 10)

## Now specified the design with all the above information and starting with a dose of 25

# Specify the design. (For details please refer to the 'DualResponsesDesign' example.)
my_design <- DualResponsesDesign(
  nextBest = my_next_best,
  model = dle_model,
  eff_model = eff_model,
  stopping = my_stopping,
  increments = my_increments,
  cohort_size = my_size,
  data = emptydata,
  startingDose = 25
)

# Specify the true DLE and efficacy curves.
my_truth_dle <- probFunction(dle_model, phi1 = -53.66584, phi2 = 10.50499)
my_truth_eff <- efficacyFunction(eff_model, theta1 = -4.818429, theta2 = 3.653058)

# For illustration purpose only 1 simulation is produced.
my_sim <- simulate(
  object = my_design,
  args = NULL,
  trueDLE = my_truth_dle,
  trueEff = my_truth_eff,
  trueNu = 1 / 0.025,
  nsim = 1,
  mcmcOptions = McmcOptions(burnin = 10, step = 1, samples = 50),
  seed = 819,
  parallel = FALSE
)

# Summary of the simulations.
my_sum <- summary(
  my_sim,
  trueDLE = my_truth_dle,
  trueEff = my_truth_eff
)

# Plot the summary of the simulations.
print(plot(my_sum))

# Example where DLE and efficacy samples are involved.
```

```
# Please refer to design-method 'simulate DualResponsesSamplesDesign' examples
# for details.
# Specify the next best method.
my_next_best <- NextBestMaxGainSamples(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3,
  derive = function(samples) {
    as.numeric(quantile(samples, prob = 0.3))
  },
  mg_derive = function(mg_samples) {
    as.numeric(quantile(mg_samples, prob = 0.5))
  }
)

# Specify the design.
my_design <- DualResponsesSamplesDesign(
  nextBest = my_next_best,
  cohort_size = my_size,
  startingDose = 25,
  model = dle_model,
  eff_model = eff_model,
  data = emptydata,
  stopping = my_stopping,
  increments = my_increments
)

# MCMC options.
my_options <- McmcOptions(burnin = 10, step = 2, samples = 50)

# For illustration purpose only 1 simulation is produced.
my_sim <- simulate(
  object = my_design,
  args = NULL,
  trueDLE = my_truth_dle,
  trueEff = my_truth_eff,
  trueNu = 1 / 0.025,
  nsim = 1,
  mcmcOptions = my_options,
  seed = 819,
  parallel = FALSE
)


# Generate a summary of the simulations.
my_sum <- summary(
  my_sim,
  trueDLE = my_truth_dle,
  trueEff = my_truth_eff
)

# Plot the summary of the simulations.
print(plot(my_sum))
```

```
# Example where the 'EffFlexi' class is used for the efficacy model.
eff_model <- EffFlexi(
  eff = c(1.223, 2.513),
  eff_dose = c(25, 300),
  sigma2W = c(a = 0.1, b = 0.1),
  sigma2betaW = c(a = 20, b = 50),
  rw1 = FALSE,
  data = emptydata
)

# Specify the design.
my_design <- DualResponsesSamplesDesign(
  nextBest = my_next_best,
  cohort_size = my_size,
  startingDose = 25,
  model = dle_model,
  eff_model = eff_model,
  data = emptydata,
  stopping = my_stopping,
  increments = my_increments
)

# Specify the true DLE curve and the true expected efficacy values at all dose levels.
my_truth_dle <- probFunction(dle_model, phi1 = -53.66584, phi2 = 10.50499)

my_truth_eff <- c(
  -0.5478867, 0.1645417, 0.5248031, 0.7604467,
  0.9333009, 1.0687031, 1.1793942, 1.2726408,
  1.3529598, 1.4233411, 1.4858613, 1.5420182
)

# Define the true gain curve.
my_truth_gain <- function(dose) {
  return((my_truth_eff(dose)) / (1 + (my_truth_dle(dose) / (1 - my_truth_dle(dose)))))
}

## The simulations
## For illustration purpose only 1 simulation is produced (nsim=1).
mySim <- simulate(
  object = my_design,
  args = NULL,
  trueDLE = my_truth_dle,
  trueEff = my_truth_eff,
  trueSigma2 = 0.025,
  trueSigma2betaW = 1,
  nsim = 1,
  mcmcOptions = my_options,
  seed = 819,
  parallel = FALSE
)

# Produce a summary of the simulations.
my_sum <- summary(
```

```
  my_sim,
  trueDLE = my_truth_dle,
  trueEff = my_truth_eff
)

# Plot the summary of the simulations.
print(plot(my_sim))
```

`plot,PseudoSimulationsSummary,missing-method`

Plot summaries of the pseudo simulations

Description

Graphical display of the simulation summary

Usage

```
## S4 method for signature 'PseudoSimulationsSummary,missing'
plot(
  x,
  y,
 type = c("nObs", "doseSelected", "propDLE", "nAboveTargetEndOfTrial", "meanFit"),
  ...
)
```

Arguments

x	the PseudoSimulationsSummary object we want to plot from
y	missing
type	the types of plots you want to obtain.
...	not used

Details

This plot method can be applied to PseudoSimulationsSummary objects in order to summarize them graphically. This can be used when only DLE responses are involved in the simulations. This also applied to results with or without samples generated during the simulations

Value

A single ggplot object if a single plot is asked for, otherwise a gridExtra{gTree} object.

Examples

```
# nolint start

## obtain the plot for the simulation results
## If only DLE responses are considered in the simulations
## Specified your simulations when no DLE samples are used
## Define your data set first using an empty data set
## with dose levels from 25 to 300 with increments 25
data <- Data(doseGrid = seq(25, 300, 25))

## Specified the model of 'ModelTox' class eg 'LogisticIndepBeta' class model
model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = data
)
## Then the escalation rule
tdNextBest <- NextBestTD(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3
)

## The cohort size, size of 3 subjects
mySize <- CohortSizeConst(size = 3)
## Deifne the increments for the dose-escalation process
## The maximum increase of 200% for doses up to the maximum of the dose specified in the doseGrid
## The maximum increase of 200% for dose above the maximum of the dose specified in the doseGrid
## This is to specified a maximum of 3-fold restriction in dose-esclation
myIncrements <- IncrementsRelative(
  intervals = c(min(data@doseGrid), max(data@doseGrid)),
  increments = c(2, 2)
)
## Specified the stopping rule e.g stop when the maximum sample size of 12 patients has been reached
myStopping <- StoppingMinPatients(nPatients = 12)
## Now specified the design with all the above information and starting with a dose of 25
design <- TDDesign(
  model = model,
  nextBest = tdNextBest,
  stopping = myStopping,
  increments = myIncrements,
  cohort_size = mySize,
  data = data, startingDose = 25
)

## Specify the truth of the DLE responses
myTruth <- probFunction(model, phi1 = -53.66584, phi2 = 10.50499)

## Then specified the simulations and generate the trial
## For illustration purpose only 1 simulation is produced (nsim=1).
## The simulations
mySim <- simulate(design,
```

```
  args = NULL,
  truth = myTruth,
  nsim = 1,
  seed = 819,
  parallel = FALSE
)

## Then produce a summary of your simulations
MYSUM <- summary(mySim,
  truth = myTruth
)
## plot the summary of the simulations
print(plot(MYSUM))

## If DLE samples are involved
## The escalation rule
tdNextBest <- NextBestTDsamples(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3,
  derive = function(samples) {
    as.numeric(quantile(samples, probs = 0.3))
  }
)
## specify the design
design <- TDsamplesDesign(
  model = model,
  nextBest = tdNextBest,
  stopping = myStopping,
  increments = myIncrements,
  cohort_size = mySize,
  data = data, startingDose = 25
)
## options for MCMC
options <- McmcOptions(burnin = 100, step = 2, samples = 200)
## The simulations
## For illustration purpose only 1 simulation is produced (nsim=1).
# mySim <- simulate(design,
#                   args=NULL,
#                   truth=myTruth,
#                   nsim=1,
#                   seed=819,
#                   mcmcOptions=options,
#                   parallel=FALSE)
# ##Then produce a summary of your simulations
# MYSUM <- summary(mySim,
#                  truth=myTruth)
# ##plot the summary of the simulations
# print(plot(MYSUM))

# nolint end
```

plot,Samples,DALogisticLogNormal-method

Plotting dose-toxicity model fits

Description

Plotting dose-toxicity model fits

Usage

```
## S4 method for signature 'Samples,DALogisticLogNormal'
plot(x, y, data, hazard = FALSE, ..., showLegend = TRUE)
```

Arguments

x	the Samples object
y	the DALogisticLogNormal object
data	the DataDA object
hazard	see fitPEM for the explanation
...	not used
showLegend	should the legend be shown? (default)

Value

This returns the ggplot object for the dose-toxicity model fit

plot,Samples,DualEndpoint-method

Plotting dose-toxicity and dose-biomarker model fits

Description

When we have the dual endpoint model, also the dose-biomarker fit is shown in the plot

Usage

```
## S4 method for signature 'Samples,DualEndpoint'
plot(x, y, data, extrapolate = TRUE, showLegend = FALSE, ...)
```

Arguments

x	the Samples object
y	the DualEndpoint object
data	the DataDual object
extrapolate	should the biomarker fit be extrapolated to the whole dose grid? (default)
showLegend	should the legend be shown? (not default)
...	additional arguments for the parent method plot,Samples,GeneralModel-method

Value

This returns the ggplot object with the dose-toxicity and dose-biomarker model fits

Examples

```
# nolint start

# Create some data
data <- DataDual(
  x=c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10,
      20, 20, 20, 40, 40, 40, 50, 50, 50),
  y=c(0, 0, 0, 0, 0, 0, 1, 0,
      0, 1, 1, 0, 0, 1, 0, 1, 1),
  w=c(0.31, 0.42, 0.59, 0.45, 0.6, 0.7, 0.55, 0.6,
      0.52, 0.54, 0.56, 0.43, 0.41, 0.39, 0.34, 0.38, 0.21),
  doseGrid=c(0.1, 0.5, 1.5, 3, 6,
             seq(from=10, to=80, by=2)))

# Initialize the Dual-Endpoint model (in this case RW1)
model <- DualEndpointRW(mean = c(0, 1),
                        cov = matrix(c(1, 0, 0, 1), nrow=2),
                        sigma2betaW = 0.01,
                        sigma2W = c(a=0.1, b=0.1),
                        rho = c(a=1, b=1),
                        rw1 = TRUE)

# Set-up some MCMC parameters and generate samples from the posterior
options <- McmcOptions(burnin=100,
                       step=2,
                       samples=500)
set.seed(94)
samples <- mcmc(data, model, options)

# Plot the posterior mean  (and empirical 2.5 and 97.5 percentile)
# for the prob(DLT) by doses and the Biomarker by doses
#grid.arrange(plot(x = samples, y = model, data = data))

plot(x = samples, y = model, data = data)

# nolint end
```

`plot,Samples,GeneralModel-method`
Plotting dose-toxicity model fits

Description

Plotting dose-toxicity model fits

Usage

```
## S4 method for signature 'Samples,GeneralModel'
plot(
  x,
  y,
  data,
  ...,
  xlab = "Dose level",
  ylab = "Probability of DLT [%]",
  showLegend = TRUE
)
```

Arguments

x	the Samples object
y	the GeneralModel object
data	the Data object
...	not used
xlab	the x axis label
ylab	the y axis label
showLegend	should the legend be shown? (default)

Value

This returns the ggplot object for the dose-toxicity model fit

Examples

```
# nolint start

# Create some data
data <- Data(x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
             y = c(0, 0, 0, 0, 0, 0, 1, 0),
             cohort = c(0, 1, 2, 3, 4, 5, 5, 5),
             doseGrid = c(0.1, 0.5, 1.5, 3, 6,
                          seq(from = 10, to = 80, by=2)))
```

```
# Initialize a model
model <- LogisticLogNormal(mean = c(-0.85, 1),
                           cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
                           ref_dose = 56)

# Get posterior for all model parameters
options <- McmcOptions(burnin = 100,
                       step = 2,
                       samples = 2000)
set.seed(94)
samples <- mcmc(data, model, options)

# Plot the posterior mean  (and empirical 2.5 and 97.5 percentile)
# for the prob(DLT) by doses
plot(x = samples, y = model, data = data)

# nolint end
```

`plot,Samples,ModelEff-method`

Plot the fitted dose-efficacy curve using a model from `ModelEff` *class with samples*

Description

Plot the fitted dose-efficacy curve using a model from `ModelEff` class with samples

Usage

```
## S4 method for signature 'Samples,ModelEff'
plot(
  x,
  y,
  data,
  ...,
  xlab = "Dose level",
  ylab = "Expected Efficacy",
  showLegend = TRUE
)
```

Arguments

`x`	the `Samples` object
`y`	the `ModelEff` model class object
`data`	the `Data` object
`...`	not used
`xlab`	the x axis label
`ylab`	the y axis label
`showLegend`	should the legend be shown? (default)

Value

This returns the ggplot object for the dose-efficacy model fit

Examples

```
# nolint start

## we need a data object with doses >= 1:
data <-DataDual(x=c(25,50,25,50,75,300,250,150),
                y=c(0,0,0,0,0,1,1,0),
                w=c(0.31,0.42,0.59,0.45,0.6,0.7,0.6,0.52),
                doseGrid=seq(25,300,25),
                placebo=FALSE)
##plot the dose-efficacy curve with samples using the model from 'ModelEff'
##class e.g. 'Effloglog' class model
##define the model (see Effloglog example)
Effmodel<-Effloglog(eff=c(1.223,2.513),eff_dose=c(25,300),nu=c(a=1,b=0.025),data=data)
## define the samples obtained using the 'Effloglog' model (see details in 'Samples' example)
##options for MCMC
options<-McmcOptions(burnin=100,step=2,samples=200)
## samples must be of 'Samples' class
samples <- mcmc(data=data,model=Effmodel,options=options)
## plot the fitted dose-efficacy curve including the 95% credibility interval of the samples
## 'x' should be of 'Samples' class and 'y' of 'ModelEff' class
plot(x=samples,y=Effmodel,data=data)
# nolint end
```

---

plot,Samples,ModelTox-method

Plot the fitted dose-DLE curve using a ModelTox class model with samples

---

Description

Plot the fitted dose-DLE curve using a ModelTox class model with samples

Usage

```
## S4 method for signature 'Samples,ModelTox'
plot(
  x,
  y,
  data,
  ...,
  xlab = "Dose level",
  ylab = "Probability of DLT [%]",
  showLegend = TRUE
)
```

Arguments

x	the Samples object
y	the ModelTox model class object
data	the Data object
...	not used
xlab	the x axis label
ylab	the y axis label
showLegend	should the legend be shown? (default)

Value

This returns the ggplot object for the dose-DLE model fit

Examples

```
## we need a data object with doses >= 1:
data <- Data(
  x = c(25, 50, 50, 75, 150, 200, 225, 300),
  y = c(0, 0, 0, 0, 1, 1, 1, 1),
  doseGrid = seq(from = 25, to = 300, by = 25)
)
## plot the dose-DLE curve with samples using the model from 'ModelTox'
## class e.g. 'LogisticIndepBeta' class model
## define the model (see LogisticIndepBeta example)
model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = data
)
## define the samples obtained using the 'LogisticIndepGBeta' model

## Define options for MCMC
options <- McmcOptions(burnin = 100, step = 2, samples = 200)
## (see details in 'Samples' example) samples must be of 'Samples' class
samples <- mcmc(data = data, model = model, options = options)
## plot the fitted dose-DLE curve including the 95% credibility interval of the samples
## 'x' should be of 'Samples' class and 'y' of 'ModelTox' class
plot(x = samples, y = model, data = data)
```

---

plot,SimulationsSummary,missing-method

Plot summaries of the model-based design simulations

Description

Graphical display of the simulation summary

Usage

```
## S4 method for signature 'SimulationsSummary,missing'
plot(
  x,
  y,
  type = c("nObs", "doseSelected", "propDLTs", "nAboveTarget", "meanFit"),
  ...
)
```

Arguments

x	the SimulationsSummary object we want to plot from
y	missing
type	the types of plots you want to obtain.
...	not used

Details

This plot method can be applied to SimulationsSummary objects in order to summarize them graphically. Possible type of plots at the moment are those listed in plot,GeneralSimulationsSummary,missing-method plus:

meanFit Plot showing the average fitted dose-toxicity curve across the trials, together with 95% credible intervals, and comparison with the assumed truth (as specified by the truth argument to summary,Simulations-method)

You can specify any subset of these in the type argument.

Value

A single ggplot object if a single plot is asked for, otherwise a gridExtra{gTree} object.

Examples

```
# nolint start

# Define the dose-grid
emptydata <- Data(doseGrid = c(1, 3, 5, 10, 15, 20, 25, 40, 50, 80, 100))

# Initialize the CRM model
model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov =
    matrix(c(1, -0.5, -0.5, 1),
      nrow = 2
    ),
  ref_dose = 56
)

# Choose the rule for selecting the next dose
```

```
myNextBest <- NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Choose the rule for the cohort-size
mySize1 <- CohortSizeRange(
  intervals = c(0, 30),
  cohort_size = c(1, 3)
)
mySize2 <- CohortSizeDLT(
  intervals = c(0, 1),
  cohort_size = c(1, 3)
)
mySize <- maxSize(mySize1, mySize2)

# Choose the rule for stopping
myStopping1 <- StoppingMinCohorts(nCohorts = 3)
myStopping2 <- StoppingTargetProb(
  target = c(0.2, 0.35),
  prob = 0.5
)
myStopping3 <- StoppingMinPatients(nPatients = 20)
myStopping <- (myStopping1 & myStopping2) | myStopping3

# Choose the rule for dose increments
myIncrements <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

# Initialize the design
design <- Design(
  model = model,
  nextBest = myNextBest,
  stopping = myStopping,
  increments = myIncrements,
  cohort_size = mySize,
  data = emptydata,
  startingDose = 3
)

## define the true function
myTruth <- probFunction(model, alpha0 = 7, alpha1 = 8)

# Run the simulation on the desired design
# We only generate 1 trial outcomes here for illustration, for the actual study
# this should be increased of course
options <- McmcOptions(
  burnin = 10,
  step = 1,
  samples = 100
```

```
)
time <- system.time(mySims <- simulate(design,
  args = NULL,
  truth = myTruth,
  nsim = 1,
  seed = 819,
  mcmcOptions = options,
  parallel = FALSE
))[3]

# Plot the Summary of the Simulations
plot(summary(mySims, truth = myTruth))

# nolint end
```

`plot.gtable` *Plot* `gtable` *Objects*

Description

This is needed because `crmPack` uses `gridExtra::arrangeGrob()` to combine `ggplot2` plots, and the resulting `gtable` object is not plotted otherwise when implicitly printing it in the console, e.g.

Usage

```
## S3 method for class 'gtable'
plot(x, ...)

## S3 method for class 'gtable'
print(x, ...)
```

Arguments

`x`	(`gtable`) object to plot.
`...`	additional parameters for `grid::grid.draw()`.

`plotDualResponses` *Plot of the DLE and efficacy curve side by side given a DLE pseudo model, a DLE sample, an efficacy pseudo model and a given efficacy sample*

Description

Plot of the DLE and efficacy curve side by side given a DLE pseudo model, a DLE sample, an efficacy pseudo model and a given efficacy sample

Plot of the dose-DLE and dose-efficacy curve side by side given a DLE pseudo model and a given pseudo efficacy model without DLE and efficacy samples

Usage

```
plotDualResponses(DLEmodel, DLEsamples, Effmodel, Effsamples, data, ...)

## S4 method for signature 'ModelTox,Samples,ModelEff,Samples'
plotDualResponses(
  DLEmodel,
  DLEsamples,
  Effmodel,
  Effsamples,
  data,
  extrapolate = TRUE,
  showLegend = FALSE,
  ...
)

## S4 method for signature 'ModelTox,missing,ModelEff,missing'
plotDualResponses(DLEmodel, DLEsamples, Effmodel, Effsamples, data, ...)
```

Arguments

DLEmodel	the pseudo DLE model of ModelTox class object
DLEsamples	the DLE samples of Samples class object
Effmodel	the pseudo efficacy model of ModelEff class object
Effsamples	the Efficacy samples of Samples class object
data	the data input of DataDual class object
...	additional arguments for the parent method plot,Samples,GeneralModel-method
extrapolate	should the biomarker fit be extrapolated to the whole dose grid? (default)
showLegend	should the legend be shown? (not default)

Value

This returns the ggplot object with the dose-toxicity and dose-efficacy model fits

Functions

- plotDualResponses(DLEmodel = ModelTox, DLEsamples = Samples, Effmodel = ModelEff, Effsamples = Samples): function still to be documented
- plotDualResponses(DLEmodel = ModelTox, DLEsamples = missing, Effmodel = ModelEff, Effsamples = missing): Plot the DLE and efficacy curve side by side given a DLE model and an efficacy model without any samples

Examples

```
# nolint start

## we need a data object with doses >= 1:
data <-DataDual(x=c(25,50,25,50,75,300,250,150),
```

```
                y=c(0,0,0,0,0,1,1,0),
                w=c(0.31,0.42,0.59,0.45,0.6,0.7,0.6,0.52),
                doseGrid=seq(25,300,25),
                placebo=FALSE)
##plot the dose-DLE and dose-efficacy curves in two plots with DLE and efficacy samples
##define the DLE model which must be of 'ModelTox' class
##(e.g 'LogisticIndepBeta' class model)
DLEmodel<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=data)
## define the efficacy model which must be of 'ModelEff' class
## (e.g 'Effloglog' class)
Effmodel<-Effloglog(eff=c(1.223,2.513),eff_dose=c(25,300),nu=c(a=1,b=0.025),data=data)
##define the DLE sample of 'Samples' class
##set up the same data set in class 'Data' for MCMC sampling for DLE
data1 <- Data(x=data@x,y=data@y,doseGrid=data@doseGrid)
##Specify the options for MCMC
options <- McmcOptions(burnin=100,step=2,samples=1000)


DLEsamples <- mcmc(data=data1,model=DLEmodel,options=options)
##define the efficacy sample of 'Samples' class
Effsamples <- mcmc(data=data,model=Effmodel,options=options)
##plot the dose-DLE and dose-efficacy curves with two plot side by side.
##For each curve the 95% credibility interval of the two samples are alos given
plotDualResponses(DLEmodel=DLEmodel,DLEsamples=DLEsamples,
         Effmodel=Effmodel,Effsamples=Effsamples,
         data=data)

# nolint end
# nolint start

## we need a data object with doses >= 1:
data <-DataDual(x=c(25,50,25,50,75,300,250,150),
                y=c(0,0,0,0,0,1,1,0),
                w=c(0.31,0.42,0.59,0.45,0.6,0.7,0.6,0.52),
                doseGrid=seq(25,300,25),
                placebo=FALSE)
##plot the dose-DLE and dose-efficacy curves in two plots without DLE and efficacy samples
##define the DLE model which must be of 'ModelTox' class
##(e.g 'LogisticIndepBeta' class model)
DLEmodel<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=data)
## define the efficacy model which must be of 'ModelEff' class
## (e.g 'Effloglog' class)
Effmodel<-Effloglog(eff=c(1.223,2.513),eff_dose=c(25,300),nu=c(a=1,b=0.025),data=data)
##plot the dose-DLE and dose-efficacy curves with two plot side by side.
plotDualResponses(DLEmodel=DLEmodel,
                  Effmodel=Effmodel,
                  data=data)

# nolint end
```

`plotGain` *Plot the gain curve in addition with the dose-DLE and dose-efficacy curve using a given DLE pseudo model, a DLE sample, a given efficacy pseudo model and an efficacy sample*

Description

Plot the gain curve in addition with the dose-DLE and dose-efficacy curve using a given DLE pseudo model, a DLE sample, a given efficacy pseudo model and an efficacy sample

Plot the gain curve in addition with the dose-DLE and dose-efficacy curve using a given DLE pseudo model, and a given efficacy pseudo model

Usage

```
plotGain(DLEmodel, DLEsamples, Effmodel, Effsamples, data, ...)

## S4 method for signature 'ModelTox,Samples,ModelEff,Samples'
plotGain(DLEmodel, DLEsamples, Effmodel, Effsamples, data, ...)

## S4 method for signature 'ModelTox,missing,ModelEff,missing'
plotGain(DLEmodel, Effmodel, data, size = c(8L, 8L), shape = c(16L, 17L), ...)
```

Arguments

`DLEmodel`	the dose-DLE model of `ModelTox` class object
`DLEsamples`	the DLE sample of `Samples` class object
`Effmodel`	the dose-efficacy model of `ModelEff` class object
`Effsamples`	the efficacy sample of of `Samples` class object
`data`	the data input of `DataDual` class object
`...`	not used
`size`	(`integer`) a vector of length two defining the sizes of the shapes used to identify the doses with, respectively, p(DLE = 0.3) and the maximum gain
`shape`	(`integer`) a vector of length two defining the shapes used to identify the doses with, respectively, p(DLE = 0.3) and the maximum gain

Value

This returns the `ggplot` object for the plot

Functions

- `plotGain( DLEmodel = ModelTox, DLEsamples = Samples, Effmodel = ModelEff, Effsamples = Samples )`: Standard method
- `plotGain( DLEmodel = ModelTox, DLEsamples = missing, Effmodel = ModelEff, Effsamples = missing )`: Standard method

Examples

```
# nolint start

## we need a data object with doses >= 1:
data <-DataDual(x=c(25,50,25,50,75,300,250,150),
                y=c(0,0,0,0,0,1,1,0),
                w=c(0.31,0.42,0.59,0.45,0.6,0.7,0.6,0.52),
                doseGrid=seq(25,300,25),
                placebo=FALSE)
##plot the dose-DLE , dose-efficacy and gain curve in the same plot with DLE and efficacy samples
##define the DLE model which must be of 'ModelTox' class
##(e.g 'LogisticIndepBeta' class model)
DLEmodel<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=data)
## define the efficacy model which must be of 'ModelEff' class
## (e.g 'Effloglog' class)
Effmodel<-Effloglog(eff=c(1.223,2.513),eff_dose=c(25,300),nu=c(a=1,b=0.025),data=data,const=0)
##define the DLE sample of 'Samples' class
##set up the same data set in class 'Data' for MCMC sampling for DLE
data1 <- Data(x=data@x,y=data@y,doseGrid=data@doseGrid)

##Define the options for MCMC
options <- McmcOptions(burnin=100,step=2,samples=1000)


DLEsamples <- mcmc(data=data1,model=DLEmodel,options=options)
##define the efficacy sample of 'Samples' class
Effsamples <- mcmc(data=data,model=Effmodel,options=options)
##plot the three curves of mean values of the DLEsamples, Effsamples and
##gain value samples (obtained within this plotGain function) at all dose levels
plotGain(DLEmodel=DLEmodel,DLEsamples=DLEsamples,
         Effmodel=Effmodel,Effsamples=Effsamples,
         data=data)
# nolint end
# nolint start
## we need a data object with doses >= 1:
data <-DataDual(x=c(25,50,25,50,75,300,250,150),
                y=c(0,0,0,0,0,1,1,0),
                w=c(0.31,0.42,0.59,0.45,0.6,0.7,0.6,0.52),
                doseGrid=seq(25,300,25),
                placebo=FALSE)
##plot the dose-DLE , dose-efficacy and gain curve in the same plot with DLE and efficacy samples
##define the DLE model which must be of 'ModelTox' class
##(e.g 'LogisticIndepBeta' class model)
DLEmodel<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=data)
## define the efficacy model which must be of 'ModelEff' class
## (e.g 'Effloglog' class)
Effmodel<-Effloglog(eff=c(1.223,2.513),eff_dose=c(25,300),nu=c(a=1,b=0.025),data=data)
##plot the three curves of using modal estimates of model parameters at all dose levels
plotGain(DLEmodel=DLEmodel,
         Effmodel=Effmodel,
         data=data)
# nolint end
```

positive_number positive_number

Description

[Experimental]

The positive_number class is a class to store not NULL, non NA, finite and strictly positive numerical value. It is mainly used to store reference dose value in model classes.

prob *Computing Toxicity Probabilities for a Given Dose, Model and Samples*

Description

[Stable]

A function that computes the probability of the occurrence of a DLE at a specified dose level, based on the model parameters (samples).

Usage

```
prob(dose, model, samples, ...)

## S4 method for signature 'numeric,LogisticNormal,Samples'
prob(dose, model, samples, ...)

## S4 method for signature 'numeric,LogisticLogNormal,Samples'
prob(dose, model, samples, ...)

## S4 method for signature 'numeric,LogisticLogNormalSub,Samples'
prob(dose, model, samples, ...)

## S4 method for signature 'numeric,ProbitLogNormal,Samples'
prob(dose, model, samples, ...)

## S4 method for signature 'numeric,ProbitLogNormalRel,Samples'
prob(dose, model, samples, ...)

## S4 method for signature 'numeric,LogisticLogNormalGrouped,Samples'
prob(dose, model, samples, group, ...)

## S4 method for signature 'numeric,LogisticKadane,Samples'
prob(dose, model, samples, ...)
```

```
## S4 method for signature 'numeric,LogisticKadaneBetaGamma,Samples'
prob(dose, model, samples, ...)

## S4 method for signature 'numeric,LogisticNormalMixture,Samples'
prob(dose, model, samples, ...)

## S4 method for signature 'numeric,LogisticNormalFixedMixture,Samples'
prob(dose, model, samples, ...)

## S4 method for signature 'numeric,LogisticLogNormalMixture,Samples'
prob(dose, model, samples, ...)

## S4 method for signature 'numeric,DualEndpoint,Samples'
prob(dose, model, samples, ...)

## S4 method for signature 'numeric,LogisticIndepBeta,Samples'
prob(dose, model, samples, ...)

## S4 method for signature 'numeric,LogisticIndepBeta,missing'
prob(dose, model, samples, ...)

## S4 method for signature 'numeric,OneParLogNormalPrior,Samples'
prob(dose, model, samples, ...)

## S4 method for signature 'numeric,OneParExpPrior,Samples'
prob(dose, model, samples, ...)

## S4 method for signature 'numeric,LogisticLogNormalOrdinal,Samples'
prob(dose, model, samples, grade, cumulative = TRUE, ...)
```

Arguments

dose	(number or numeric) the dose which is targeted. The following recycling rule applies when samples is not missing: vectors of size 1 will be recycled to the size of the sample (i.e. size(samples)). Otherwise, dose must have the same size as the sample.
model	(GeneralModel or ModelTox) the model for single agent dose escalation or pseudo DLE (dose-limiting events)/toxicity model.
samples	(Samples) the samples of model's parameters that will be used to compute toxicity probabilities. Can also be missing for some models.
...	model specific parameters when samples are not used.
group	(character or factor) for LogisticLogNormalGrouped, indicating whether to calculate the probability for the mono or for the combo arm.
grade	(integer or integer_vector) The toxicity grade for which probabilities are required

`cumulative` (flag)
Should the returned probability be cumulative (the default) or grade-specific?

Details

The `prob()` function computes the probability of toxicity for given doses, using samples of the model parameter(s). If you work with multivariate model parameters, then assume that your model specific `prob()` method receives a samples matrix where the rows correspond to the sampling index, i.e. the layout is then `nSamples x dimParameter`.

Value

A `proportion` or `numeric` vector with the toxicity probabilities. If non-scalar `samples` were used, then every element in the returned vector corresponds to one element of a sample. Hence, in this case, the output vector is of the same length as the sample vector. If scalar `samples` were used or no `samples` were used, e.g. for pseudo DLE/toxicity `model`, then the output is of the same length as the length of the `dose`. In the case of `LogisticLogNormalOrdinal`, the probabilities relate to toxicities of grade given by `grade`.

Functions

- `prob(dose = numeric, model = LogisticNormal, samples = Samples)`:
- `prob(dose = numeric, model = LogisticLogNormal, samples = Samples)`:
- `prob(dose = numeric, model = LogisticLogNormalSub, samples = Samples)`:
- `prob(dose = numeric, model = ProbitLogNormal, samples = Samples)`:
- `prob(dose = numeric, model = ProbitLogNormalRel, samples = Samples)`:
- `prob(dose = numeric, model = LogisticLogNormalGrouped, samples = Samples)`: method for `LogisticLogNormalGrouped` which needs `group` argument in addition.
- `prob(dose = numeric, model = LogisticKadane, samples = Samples)`:
- `prob(dose = numeric, model = LogisticKadaneBetaGamma, samples = Samples)`:
- `prob(dose = numeric, model = LogisticNormalMixture, samples = Samples)`:
- `prob(dose = numeric, model = LogisticNormalFixedMixture, samples = Samples)`:
- `prob(dose = numeric, model = LogisticLogNormalMixture, samples = Samples)`:
- `prob(dose = numeric, model = DualEndpoint, samples = Samples)`:
- `prob(dose = numeric, model = LogisticIndepBeta, samples = Samples)`: compute toxicity probabilities of the occurrence of a DLE at a specified dose level, based on the samples of `LogisticIndepBeta` model parameters.
- `prob(dose = numeric, model = LogisticIndepBeta, samples = missing)`: compute toxicity probabilities of the occurrence of a DLE at a specified dose level, based on the `LogisticIndepBeta` model parameters. All model parameters (except `dose`) should be present in the `model` object.
- `prob(dose = numeric, model = OneParLogNormalPrior, samples = Samples)`:
- `prob(dose = numeric, model = OneParExpPrior, samples = Samples)`:
- `prob(dose = numeric, model = LogisticLogNormalOrdinal, samples = Samples)`:

Note

The prob() and dose() functions are the inverse of each other, for all dose() methods for which its first argument, i.e. a given independent variable that dose depends on, represents toxicity probability.

See Also

probFunction(), dose(), efficacy().

Examples

```
# Create some data.
my_data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  cohort = c(0, 1, 2, 3, 4, 5, 5, 5),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)

# Initialize a model, e.g. 'LogisticLogNormal'.
my_model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 56
)

# Get samples from posterior.
my_options <- McmcOptions(burnin = 100, step = 2, samples = 20)
my_samples <- mcmc(data = my_data, model = my_model, options = my_options)

# Posterior for Prob(DLT | dose = 50).
prob(dose = 50, model = my_model, samples = my_samples)

# Create data from the 'DataDual' class.
data_dual <- DataDual(
  x = c(25, 50, 25, 50, 75, 300, 250, 150),
  y = c(0, 0, 0, 0, 0, 1, 1, 0),
  w = c(0.31, 0.42, 0.59, 0.45, 0.6, 0.7, 0.6, 0.52),
  doseGrid = seq(from = 25, to = 300, by = 25)
)

# Initialize a toxicity model using 'LogisticIndepBeta' model.
dlt_model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = data_dual
)

# Get samples from posterior.
dlt_sample <- mcmc(data = data_dual, model = dlt_model, options = my_options)
```

```
# Posterior for Prob(DLT | dose = 100).
prob(dose = 100, model = dlt_model, samples = dlt_sample)
prob(dose = c(50, 150), model = dlt_model)
```

---

probFunction *Getting the Prob Function for a Given Model Type*

Description

[Experimental]

A function that returns a prob() function that computes the toxicity probabilities for a given dose level, based on the model specific parameters.

Usage

```
probFunction(model, ...)

## S4 method for signature 'GeneralModel'
probFunction(model, ...)

## S4 method for signature 'ModelTox'
probFunction(model, ...)

## S4 method for signature 'LogisticLogNormalOrdinal'
probFunction(model, grade, ...)
```

Arguments

model	(GeneralModel or ModelTox) the model.
...	model specific parameters.
grade	(integer) the toxicity grade for which the dose function is required

Value

A prob() function that computes toxicity probabilities.

Functions

- probFunction(GeneralModel):
- probFunction(ModelTox):
- probFunction(LogisticLogNormalOrdinal):

See Also

prob(), doseFunction().

Examples

```
my_model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 50
)

prob_fun <- probFunction(my_model, alpha0 = 2, alpha1 = 3)
prob_fun(30)
ordinal_data <- .DefaultDataOrdinal()
ordinal_model <- .DefaultLogisticLogNormalOrdinal()
opts <- .DefaultMcmcOptions()
samples <- mcmc(ordinal_data, ordinal_model, opts)

probFunction(
  ordinal_model,
  grade = 2L,
  alpha2 = samples@data$alpha1,
  beta = samples@data$beta
)(50)
```

---

probit *Shorthand for probit function*

Description

Shorthand for probit function

Usage

```
probit(x)
```

Arguments

x	the function argument

Value

the probit(x)

ProbitLogNormal-class ProbitLogNormal

Description

[Stable]

ProbitLogNormal is the class for probit regression model with a bivariate normal prior on the intercept and log slope.

Usage

```
ProbitLogNormal(mean, cov, ref_dose = 1)

.DefaultProbitLogNormal()
```

Arguments

mean	(numeric) the prior mean vector.
cov	(matrix) the prior covariance matrix. The precision matrix prec is internally calculated as an inverse of cov.
ref_dose	(number) the reference dose $x*$ (strictly positive number).

Details

The covariate is the natural logarithm of dose x divided by a reference dose $x*$, i.e.:

$$probit[p(x)] = alpha0 + alpha1 * log(x/x*),$$

where $p(x)$ is the probability of observing a DLT for a given dose x. The prior

$$(alpha0, log(alpha1))\ Normal(mean, cov).$$

Note

This model is also used in the DualEndpoint classes, so this class can be used to check the prior assumptions on the dose-toxicity model, even when sampling from the prior distribution of the dual endpoint model is not possible.

Typically, end users will not use the .DefaultProbitLogNormal() function.

See Also

ModelLogNormal, LogisticNormal, LogisticLogNormal, LogisticLogNormalSub, ProbitLogNormalRel.

Examples

```
my_model <- ProbitLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 7.2
)
```

`ProbitLogNormalRel-class` `ProbitLogNormalRel`

Description

[Stable]

`ProbitLogNormalRel` is the class for probit regression model with a bivariate normal prior on the intercept and log slope.

Usage

```
ProbitLogNormalRel(mean, cov, ref_dose = 1)

.DefaultProbitLogNormalRel()
```

Arguments

`mean`	(`numeric`) the prior mean vector.
`cov`	(`matrix`) the prior covariance matrix. The precision matrix `prec` is internally calculated as an inverse of `cov`.
`ref_dose`	(`number`) the reference dose $x*$ (strictly positive number).

Details

The covariate is the dose x divided by a reference dose $x*$, i.e.:

$$probit[p(x)] = alpha0 + alpha1 * x/x*,$$

where $p(x)$ is the probability of observing a DLT for a given dose x. The prior

$$(alpha0, log(alpha1))\ Normal(mean, cov).$$

Note

This model is also used in the `DualEndpoint` classes, so this class can be used to check the prior assumptions on the dose-toxicity model, even when sampling from the prior distribution of the dual endpoint model is not possible.

Typically, end users will not use the `.DefaultProbitLogNormalRel()` function.

See Also

ModelLogNormal, LogisticNormal, LogisticLogNormal, LogisticLogNormalSub, ProbitLogNormal.

Examples

```
my_model <- ProbitLogNormalRel(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2)
)
```

`PseudoDualFlexiSimulations` *Initialization function for 'PseudoDualFlexiSimulations' class*

Description

Initialization function for 'PseudoDualFlexiSimulations' class

Usage

```
PseudoDualFlexiSimulations(sigma2betaWest, ...)
```

Arguments

`sigma2betaWest`	please refer to `PseudoDualFlexiSimulations` class object
`...`	additional parameters from `PseudoDualSimulations`

Value

the `PseudoDualFlexiSimulations` object

`PseudoDualFlexiSimulations-class` *This is a class which captures the trial simulations design using both the DLE and efficacy responses. The design of model from `ModelTox` class and the efficacy model from `EffFlexi` class It contains all slots from `GeneralSimulations`, `PseudoSimulations` and `PseudoDualSimulations` object. In comparison to the parent class `PseudoDualSimulations`, it contains additional slots to capture the sigma2betaW estimates.*

Description

This is a class which captures the trial simulations design using both the DLE and efficacy responses. The design of model from `ModelTox` class and the efficacy model from `EffFlexi` class It contains all slots from `GeneralSimulations`, `PseudoSimulations` and `PseudoDualSimulations` object. In comparison to the parent class `PseudoDualSimulations`, it contains additional slots to capture the sigma2betaW estimates.

Usage

```
.DefaultPseudoDualFlexiSimulations()
```

Slots

`sigma2betaWest` the vector of the final posterior mean sigma2betaW estimates

Note

Typically, end users will not use the `.DefaultPseudoFlexiSimulations()` function.

PseudoDualSimulations-class
PseudoDualSimulations

Description

[Stable] This class conducts trial simulations for designs using both the DLE and efficacy responses. It defines final values for efficacy fit and DLE, estimates of Gstar, optimal dose and sigma2.

Usage

```
PseudoDualSimulations(
  fit_eff,
  final_gstar_estimates,
  final_gstar_at_dose_grid,
  final_gstar_cis,
  final_gstar_ratios,
  final_optimal_dose,
  final_optimal_dose_at_dose_grid,
  sigma2_est,
  ...
)

.DefaultPseudoDualSimulations()
```

Arguments

`fit_eff` (`list`)
see slot definition.

`final_gstar_estimates` (`numeric`)
see slot definition.

`final_gstar_at_dose_grid` (`numeric`)
see slot definition.

`final_gstar_cis` (`list`)
see slot definition.

`final_gstar_ratios` (`numeric`)
see slot definition.

`final_optimal_dose` (`numeric`)
see slot definition.

`final_optimal_dose_at_dose_grid` (`numeric`)
see slot definition.

`sigma2_est` (`numeric`)
see slot definition.

`...` additional parameters from `PseudoSimulations`

Slots

`fit_eff` (`list`)
final values of efficacy fit.

`final_gstar_estimates` (`numeric`)
final Gstar estimates.

`final_gstar_at_dose_grid` (`numeric`)
final Gstar estimates at dose grid.

`final_gstar_cis` (`list`)
list of 95% confidence interval for Gstar estimates.

`final_gstar_ratios` (`numeric`)
ratios of confidence intervals for Gstar estimates.

`final_optimal_dose` (`numeric`)
final optimal dose.

`final_optimal_dose_at_dose_grid` (`numeric`)
final optimal dose at dose grid.

`sigma2_est` (`numeric`)
final sigma2 estimates.

Note

Do not use the `.DefaultPseudoDualSimulations()` function.

PseudoDualSimulationsSummary-class
Class for the summary of the dual responses simulations using pseudo models

Description

It contains all slots from PseudoSimulationsSummary object. In addition to the slots in the parent class PseudoSimulationsSummary, it contains four more slots for the efficacy model fit information.

Usage

```
.DefaultPseudoDualSimulationsSummary()
```

Details

Note that objects should not be created by users, therefore no initialization function is provided for this class.

Slots

`targetGstar` the target dose level such that its gain value is at maximum

`targetGstarAtDoseGrid` the dose level at dose Grid closest and below Gstar

`GstarSummary` the six-number table summary (lowest, 25th, 50th (median), 75th percentile, mean and highest value) of the final Gstar values obtained across all simulations

`ratioGstarSummary` the six-number summary table of the ratios of the upper to the lower 95% credibility intervals of the final Gstar across all simulations

`EffFitAtDoseMostSelected` fitted expected mean efficacy value at dose most often selected

`meanEffFit` list with mean, lower (2.5%) and upper (97.5%) quantiles of the fitted expected efficacy value at each dose level.

Note

Typically, end users will not use the `.DefaultPseudoDualSimulationsSummary()` function.

PseudoSimulations-class PseudoSimulations

Description

[Stable] This class captures trial simulations from designs using pseudo model. It has additional slots `fit` and `stop_reasons` compared to the general class `GeneralSimulations`.

Usage

```
PseudoSimulations(
  fit,
  final_td_target_during_trial_estimates,
  final_td_target_end_of_trial_estimates,
  final_td_target_during_trial_at_dose_grid,
  final_td_target_end_of_trial_at_dose_grid,
  final_tdeot_cis,
  final_tdeot_ratios,
  final_cis,
  final_ratios,
  stop_report,
  stop_reasons,
  ...
)

.DefaultPseudoSimulations()
```

Arguments

`fit` (`list`)
see slot definition.

`final_td_target_during_trial_estimates` (`numeric`)
see slot definition.

`final_td_target_end_of_trial_estimates` (`numeric`)
see slot definition.

`final_td_target_during_trial_at_dose_grid` (`numeric`)
see slot definition.

`final_td_target_end_of_trial_at_dose_grid` (`numeric`)
see slot definition.

`final_tdeot_cis` (`list`)
see slot definition.

`final_tdeot_ratios`
: (`numeric`)
see slot definition.

`final_cis`
: (`list`)
see slot definition.

`final_ratios`
: (`numeric`)
see slot definition.

`stop_report`
: see `PseudoSimulations`

`stop_reasons`
: (`list`)
see slot definition.

`...`
: additional parameters from `GeneralSimulations`

Slots

`fit` (`list`)
: final fit values.

`final_td_target_during_trial_estimates` (`numeric`)
: final estimates of the `td_target_during_trial`.

`final_td_target_end_of_trial_estimates` (`numeric`)
: final estimates of the `td_target_end_of_trial`.

`final_td_target_during_trial_at_dose_grid` (`numeric`)
: dose levels at dose grid closest below the final `td_target_during_trial` estimates.

`final_td_target_end_of_trial_at_dose_grid` (`numeric`)
: dose levels at dose grid closest below the final `td_target_end_of_trial` estimates.

`final_tdeot_cis` (`list`)
: 95% credibility intervals of the final estimates for `td_target_end_of_trial`.

`final_tdeot_ratios` (`numeric`)
: ratio of the upper to the lower 95% credibility intervals for `td_target_end_of_trial`.

`final_cis` (`list`)
: final 95% credibility intervals for `td_target_end_of_trial` estimates.

`final_ratios` (`numeric`)
: final ratios of the upper to the lower 95% credibility interval for `td_target_end_of_trial`.

`stop_report` (`matrix`)
: outcomes of stopping rules.

`stop_reasons` (`list`)
: reasons for stopping each simulation run.

Note

Typically, end users will not use the `.DefaultPseudoSimulations()` function.

PseudoSimulationsSummary-class
Class for the summary of pseudo-models simulations output

Description

Note that objects should not be created by users, therefore no initialization function is provided for this class.

Slots

`targetEndOfTrial` the target probability of DLE wanted at the end of a trial

`targetDoseEndOfTrial` the dose level corresponds to the target probability of DLE wanted at the end of a trial, TDEOT

`targetDoseEndOfTrialAtDoseGrid` the dose level at dose grid corresponds to the target probability of DLE wanted at the end of a trial

`targetDuringTrial` the target probability of DLE wanted during a trial

`targetDoseDuringTrial` the dose level corresponds to the target probability of DLE wanted during the trial. TDDT

`targetDoseDuringTrialAtDoseGrid` the dose level at dose grid corresponds to the target probability of DLE wanted during a trial

`TDEOTSummary` the six-number table summary, include the lowest, the 25th precentile (lower quartile), the 50th percentile (median), the mean, the 27th percentile and the highest values of the final dose levels obtained corresponds to the target probability of DLE want at the end of a trial across all simulations

`TDDTSummary` the six-number table summary, include the lowest, the 25th precentile (lower quartile), the 50th percentile (median), the mean, the 27th percentile and the highest values of the final dose levels obtained corresponds to the target probability of DLE want during a trial across all simulations

`FinalDoseRecSummary` the six-number table summary, include the lowest, the 25th precentile (lower quartile), the 50th percentile (median), the mean, the 27th percentile and the highest values of the final optimal doses, which is either the TDEOT when only DLE response are incorporated into the escalation procedure or the minimum of the TDEOT and Gstar when DLE and efficacy responses are incorporated, across all simulations

`ratioTDEOTSummary` the six-number summary table of the final ratios of the upper to the lower 95% credibility intervals of the final TDEOTs across all simulations

`FinalRatioSummary` the six-number summary table of the final ratios of the upper to the lower 95% credibility intervals of the final optimal doses across all simulations #@slot doseRec the dose level that will be recommend for subsequent study

`nsim` number of simulations

`propDLE` proportions of DLE in the trials

`meanToxRisk` mean toxicity risks for the patients

`doseSelected` doses selected as MTD (targetDoseEndOfTrial)

`toxAtDosesSelected` true toxicity at doses selected

`propAtTargetEndOfTrial` Proportion of trials selecting at the doseGrid closest below the MTD, the targetDoseEndOfTrial

`propAtTargetDuringTrial` Proportion of trials selecting at the doseGrid closest below the target-DoseDuringTrial

`doseMostSelected` dose most often selected as MTD

`obsToxRateAtDoseMostSelected` observed toxicity rate at dose most often selected

`nObs` number of patients overall

`nAboveTargetEndOfTrial` number of patients treated above targetDoseEndOfTrial

`nAboveTargetDuringTrial` number of patients treated above targetDoseDuringTrial

`doseGrid` the dose grid that has been used

`fitAtDoseMostSelected` fitted toxicity rate at dose most often selected

`meanFit` list with the average, lower (2.5%) and upper (97.5%) quantiles of the mean fitted toxicity at each dose level

`stop_report` matrix of stopping rule outcomes

`Quantiles2LogisticNormal`

Convert prior quantiles (lower, median, upper) to logistic (log) normal model

---

Description

This function uses generalized simulated annealing to optimize a `LogisticNormal` model to be as close as possible to the given prior quantiles.

Usage

```
Quantiles2LogisticNormal(
  dosegrid,
  refDose,
  lower,
  median,
  upper,
  level = 0.95,
  logNormal = FALSE,
  parstart = NULL,
  parlower = c(-10, -10, 0, 0, -0.95),
  parupper = c(10, 10, 10, 10, 0.95),
  seed = 12345,
  verbose = TRUE,
 control = list(threshold.stop = 0.01, maxit = 50000, temperature = 50000, max.time =
    600)
)
```

Arguments

dosegrid	the dose grid
refDose	the reference dose
lower	the lower quantiles
median	the medians
upper	the upper quantiles
level	the credible level of the (lower, upper) intervals (default: 0.95)
logNormal	use the log-normal prior? (not default) otherwise, the normal prior for the logistic regression coefficients is used
parstart	starting values for the parameters. By default, these are determined from the medians supplied.
parlower	lower bounds on the parameters (intercept alpha and the slope beta, the corresponding standard deviations and the correlation.)
parupper	upper bounds on the parameters
seed	seed for random number generation
verbose	be verbose? (default)
control	additional options for the optimisation routine, see GenSA for more details

Value

a list with the best approximating model (LogisticNormal or LogisticLogNormal), the resulting quantiles, the required quantiles and the distance to the required quantiles, as well as the final parameters (which could be used for running the algorithm a second time)

Report *A Reference Class to represent sequentially updated reporting objects.*

Description

A Reference Class to represent sequentially updated reporting objects.

Fields

object The object from which to report

df the data frame to which columns are sequentially added

dfNames the names to which strings are sequentially added

RuleDesign-class RuleDesign

Description

[Stable]

RuleDesign is the class for rule-based designs. The difference between this class and the Design class is that RuleDesign does not contain model, stopping and increments slots.

Usage

```
RuleDesign(nextBest, cohort_size, data, startingDose)

.DefaultRuleDesign()

ThreePlusThreeDesign(doseGrid)
```

Arguments

nextBest	(NextBest) see slot definition.
cohort_size	(CohortSize) see slot definition.
data	(Data) see slot definition.
startingDose	(number) see slot definition.
doseGrid	(numeric) the dose grid to be used (sorted).

Functions

- ThreePlusThreeDesign(): creates a new 3+3 design object from a dose grid.

Slots

nextBest (NextBest)
: how to find the next best dose.

cohort_size (CohortSize)
: rules for the cohort sizes.

data (Data)
: specifies dose grid, any previous data, etc.

startingDose (number)
: the starting dose, it must lie on the dose grid in data.

Note

Typically, end users will not use the `.DefaultRuleDesign()` function.

Examples

```
# Specify the design to run simulations. The design comprises a model,
# the escalation rule, starting data, a cohort size and a starting dose.

# Initialing a 3+3 design with constant cohort size of 3 and starting dose equal 5.
my_design <- RuleDesign(
  nextBest = NextBestThreePlusThree(),
  cohort_size = CohortSizeConst(size = 3L),
  data = Data(doseGrid = c(5, 10, 15, 25, 35, 50, 80)),
  startingDose = 5
)
# Initialing a 3+3 design with constant cohort size of 3 and starting dose equal 8.
my_design <- ThreePlusThreeDesign(doseGrid = c(8, 10, 15, 25, 35, 50, 80))
```

RuleDesignOrdinal-class

`RuleDesignOrdinal`

Description

[Experimental]

`RuleDesignOrdinal` is the class for rule-based designs. The difference between this class and the `DesignOrdinal` class is that `RuleDesignOrdinal` does not contain `model`, `stopping` and `increments` slots.

Usage

```
RuleDesignOrdinal(next_best, cohort_size, data, starting_dose)

.DefaultRuleDesignOrdinal()
```

Arguments

`next_best`	(`NextBestOrdinal`) see slot definition.
`cohort_size`	(`CohortSizeOrdinal`) see slot definition.
`data`	(`DataOrdinal`) see slot definition.
`starting_dose`	(`number`) see slot definition.

Slots

`next_best` (`NextBestOrdinal`)
: how to find the next best dose.

`cohort_size` (`CohortSizeOrdinal`)
: rules for the cohort sizes.

`data` (`DataOrdinal`)
: specifies dose grid, any previous data, etc.

`starting_dose` (`number`)
: the starting dose, it must lie on the dose grid in `data`.

Note

Typically, end users will not use the `.DefaultRuleDesignOrdinal()` function.

Examples

```
RuleDesignOrdinal(
  next_best = NextBestOrdinal(
    1L,
    NextBestMTD(
      target = 0.25,
      derive = function(x) median(x, na.rm = TRUE)
    )
  ),
  cohort_size = CohortSizeOrdinal(1L, CohortSizeConst(size = 3L)),
  data = DataOrdinal(doseGrid = c(5, 10, 15, 25, 35, 50, 80)),
  starting_dose = 5
)
```

SafetyWindow-class SafetyWindow

Description

[Stable]

`SafetyWindow` is a class for safety window.

Usage

```
.DefaultSafetyWindow()
```

Note

Typically, end users will not use the `.DefaultSafetyWindow()` function.

See Also

`SafetyWindowSize`, `SafetyWindowConst`.

SafetyWindowConst-class SafetyWindowConst

Description

[Stable]

SafetyWindowConst is the class for safety window length and it is used when the gap should be kept constant across cohorts (though it may vary within a cohort).

Usage

```
SafetyWindowConst(gap, follow, follow_min)

.DefaultSafetyWindowConst()
```

Arguments

gap	see slot definition.
follow	see slot definition.
follow_min	see slot definition.

Slots

gap (integer)
: a vector, the constant gap between patients.

follow (count)
: how long to follow each patient. The period of time that each patient in the cohort needs to be followed before the next cohort opens.

follow_min (count)
: minimum follow up. At least one patient in the cohort needs to be followed at the minimal follow up time.

Note

Typically, end users will not use the .DefaultSafetyWindowConst() function.

Examples

```
# This is to have along the study constant parameters settings of safety window
# length, regardless of the cohort size.
my_win_len <- SafetyWindowConst(
  gap = c(7, 5, 3),
  follow = 7,
  follow_min = 14
)
```

SafetyWindowSize-class SafetyWindowSize

Description

[Stable]

SafetyWindowSize is the class for safety window length based on cohort size. This class is used to decide the rolling rule from the clinical perspective.

Usage

```
SafetyWindowSize(gap, size, follow, follow_min)

.DefaultSafetyWindowSize()
```

Arguments

gap	see slot definition.
size	see slot definition.
follow	see slot definition.
follow_min	see slot definition.

Slots

gap (list)
: observed period of the previous patient before the next patient can be dosed. This is used as follows. If for instance, the cohort size is 4 and we want to specify three time intervals between these four consecutive patients, i.e. 7 units of time between the 1st and the 2nd patient, 5 units between the 2nd and the 3rd one, and finally 3 units between the 3rd and the 4th one, then, gap = list(c(7L, 5L, 3L)). Sometimes, we want that the interval only between the 1st and 2nd patient should be increased for the safety consideration and the rest time intervals should remain constant, regardless of what the cohort size is. Then, gap = list(c(7L, 3L)) and the the package will automatically repeat the last element of the vector for the remaining time intervals.

size (integer)
: a vector with the left bounds of the relevant cohort size intervals. This is used as follows. For instance, when we want to change the gap based on the cohort size, i.e. the time interval between the 1st and 2nd patient = 9 units of time and the rest time intervals are of 5 units of time when the cohort size is equal to or larger than 4. And the time interval between the 1st and 2nd patient = 7 units of time and the rest time intervals are 3 units of time when the cohort size is smaller than 4, then we specify both gap = list(c(7, 3), c(9, 5)) and size = c(0L, 4L). This means, the right bounds of the intervals are excluded from the interval, and the last interval goes from the last value to infinity.

follow (count)
: the period of time that each patient in the cohort needs to be followed before the next cohort opens.

follow_min (count)
: at least one patient in the cohort needs to be followed at the minimal follow up time.

Note

Typically, end users will not use the `.DefaultSafetyWindowSize()` function.

Examples

```
# Rule for having patient gap (7,3,3,3,...) for cohort size < 4, and
# patient gap (9,5,5,5...) for cohort size >= 4.
my_window_len <- SafetyWindowSize(
  gap = list(c(7, 3), c(9, 5)),
  size = c(1, 4),
  follow = 7,
  follow_min = 14
)
```

Samples-class Samples

Description

[Stable]

Samples is the class to store the MCMC samples.

Usage

```
Samples(data, options)

.DefaultSamples()
```

Arguments

data	see slot definition.
options	see slot definition.

Slots

data (list)
: MCMC samples of the parameter. Each entry in this list must be a vector (in case of a scalar parameter) or matrix (in case of a vector-valued parameter) with samples. In case of matrix, every row is a separate sample, while columns correspond to the dimension of the parameter.

options (McmcOptions)
: MCMC options that were used to generate the samples.

Note

Typically, end users will not use the `.DefaultSamples()` function.

Examples

```
# The MCMC options that were used to generate the samples.
my_options <- McmcOptions(
  burnin = 1000,
  step = 2,
  samples = 1000
)

# Create an object of class 'Samples'
# Here the parameters 'alpha' and 'beta' are randomly generated. Of course, in
# a real example these would be a samples coming from MCMC procedures.
my_samples <- Samples(
  data = list(alpha = rnorm(1000), beta = rnorm(1000)),
  options = my_options
)
```

---

saveSample *Determining if this Sample Should be Saved*

Description

[Stable]

A method that determines if a sample from a given `iteration` should be saved. The sample should be saved if and only if: it is not in burn-in period and it matches the `step`.

Usage

```
saveSample(object, iteration, ...)

## S4 method for signature 'McmcOptions'
saveSample(object, iteration, ...)
```

Arguments

`object`	(`McmcOptions`) object based on which the answer is determined.
`iteration`	(`count`) the current iteration index.
`...`	not used.

Value

`TRUE` if this sample should be saved.

Functions

- `saveSample(McmcOptions)`: determine if a sample should be saved.

Examples

```
# Set up the MCMC option in order to have a burn-in of 10000 iterations and
# then take every other iteration up to a collection of 10000 samples.
my_options <- McmcOptions(burnin = 10000, step = 2, samples = 10000)

size(my_options)
saveSample(my_options, iteration = 5)
```

---

set_seed *Helper Function to Set and Save the RNG Seed*

Description

[Stable]

This code is basically copied from `stats:::simulate.lm`.

Usage

```
set_seed(seed = NULL)
```

Arguments

`seed`	an object specifying if and how the random number generator should be initialized ("seeded"). Either `NULL` (default) or an integer that will be used in a call to `set.seed()` before simulating the response vectors. If set, the value is saved as the `seed` slot of the returned object. The default, `NULL` will not change the random generator state.

Value

The integer vector containing the random number generate state will be returned, in order to call this function with this input to reproduce the obtained simulation results.

show,DualSimulationsSummary-method
Show the summary of the dual-endpoint simulations

Description

Show the summary of the dual-endpoint simulations

Usage

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

Arguments

object	the DualSimulationsSummary object we want to print

Value

invisibly returns a data frame of the results with one row and appropriate column names

Examples

```
# Define the dose-grid.
emptydata <- DataDual(doseGrid = c(1, 3, 5, 10, 15, 20, 25, 30))

# Initialize the CRM model.
my_model <- DualEndpointRW(
  mean = c(0, 1),
  cov = matrix(c(1, 0, 0, 1), nrow = 2),
  sigma2betaW = 0.01,
  sigma2W = c(a = 0.1, b = 0.1),
  rho = c(a = 1, b = 1),
  rw1 = TRUE
)

# Choose the rule for selecting the next dose.
my_next_best <- NextBestDualEndpoint(
  target = c(0.9, 1),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Choose the rule for the cohort-size.
my_size1 <- CohortSizeRange(
  intervals = c(0, 30),
  cohort_size = c(1, 3)
)
my_size2 <- CohortSizeDLT(
  intervals = c(0, 1),
```

```
  cohort_size = c(1, 3)
)
my_size <- maxSize(my_size1, my_size2)

# Choose the rule for stopping.
my_stopping1 <- StoppingTargetBiomarker(
  target = c(0.9, 1),
  prob = 0.5
)

# Stop with a small number of patients for illustration.
my_stopping <- my_stopping1 | StoppingMinPatients(10) | StoppingMissingDose()

# Choose the rule for dose increments.
my_increments <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

# Initialize the design.
my_design <- DualDesign(
  model = my_model,
  data = emptydata,
  nextBest = my_next_best,
  stopping = my_stopping,
  increments = my_increments,
  cohort_size = CohortSizeConst(3),
  startingDose = 3
)

# Define scenarios for the TRUE toxicity and efficacy profiles.
beta_mod <- function(dose, e0, eMax, delta1, delta2, scal) {
  maxDens <- (delta1^delta1) * (delta2^delta2) / ((delta1 + delta2)^(delta1 + delta2))
  dose <- dose / scal
  e0 + eMax / maxDens * (dose^delta1) * (1 - dose)^delta2
}

true_biomarker <- function(dose) {
  beta_mod(dose, e0 = 0.2, eMax = 0.6, delta1 = 5, delta2 = 5 * 0.5 / 0.5, scal = 100)
}

true_tox <- function(dose) {
  pnorm((dose - 60) / 10)
}

# Draw the TRUE profiles.
par(mfrow = c(1, 2))
curve(true_tox(x), from = 0, to = 80)
curve(true_biomarker(x), from = 0, to = 80)

# Run the simulation on the desired design.
# For illustration purposes only 1 trial outcome is generated and 5 burn-ins
# to generate 20 samples are used here.
```

```
my_sims <- simulate(
  object = my_design,
  trueTox = true_tox,
  trueBiomarker = true_biomarker,
  sigma2W = 0.01,
  rho = 0,
  nsim = 1,
  parallel = FALSE,
  seed = 3,
  startingDose = 6,
  mcmcOptions = McmcOptions(
    burnin = 5,
    step = 1,
    samples = 20
  )
)

# Show the summary of the simulations.
show(summary(
  my_sims,
  trueTox = true_tox,
  trueBiomarker = true_biomarker
))
```

`show,GeneralSimulationsSummary-method`
Show the summary of the simulations

Description

Show the summary of the simulations

Usage

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

Arguments

`object`	the `GeneralSimulationsSummary` object we want to print

Value

invisibly returns a data frame of the results with one row and appropriate column names

show,PseudoDualSimulationsSummary-method
Show the summary of Pseudo Dual simulations summary

Description

Show the summary of Pseudo Dual simulations summary

Usage

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

Arguments

object	the PseudoDualSimulationsSummary object we want to print

Value

invisibly returns a data frame of the results with one row and appropriate column names

Examples

```
# Example where DLE and efficacy responses are considered in the simulations.
# In simulations where no samples are used a data object with doses >= 1 needs
# to be generated.
emptydata <- DataDual(doseGrid = seq(25, 300, 25), placebo = FALSE)

# The DLE model must be of 'ModelTox' (e.g 'LogisticIndepBeta') class.
dle_model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = emptydata
)

# The efficacy model of 'ModelEff' (e.g 'Effloglog') class.
eff_model <- Effloglog(
  eff = c(1.223, 2.513),
  eff_dose = c(25, 300),
  nu = c(a = 1, b = 0.025),
  data = emptydata
)

# The escalation rule using the 'NextBestMaxGain' class.
my_next_best <- NextBestMaxGain(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3
)
```

```
# Allow increase of 200%.
my_increments <- IncrementsRelative(intervals = 0, increments = 2)


# Cohort size of 3.
my_size <- CohortSizeConst(size = 3)

# Stop when 36 subjects are treated or next dose is NA.
my_stopping <- StoppingMinPatients(nPatients = 36) | StoppingMissingDose()

# Specify the design. (For details please refer to the 'DualResponsesDesign' example.)
my_design <- DualResponsesDesign(
  nextBest = my_next_best,
  model = dle_model,
  eff_model = eff_model,
  stopping = my_stopping,
  increments = my_increments,
  cohort_size = my_size,
  data = emptydata,
  startingDose = 25
)

# Specify the true DLE and efficacy curves.
my_truth_dle <- probFunction(dle_model, phi1 = -53.66584, phi2 = 10.50499)
my_truth_eff <- efficacyFunction(eff_model, theta1 = -4.818429, theta2 = 3.653058)

# For illustration purpose only 2 simulations are produced.
my_sim <- simulate(
  object = my_design,
  args = NULL,
  trueDLE = my_truth_dle,
  trueEff = my_truth_eff,
  trueNu = 1 / 0.025,
  nsim = 2,
  seed = 819,
  parallel = FALSE
)

# Summary of the simulations.
my_sum <- summary(
  my_sim,
  trueDLE = my_truth_dle,
  trueEff = my_truth_eff
)

# Show the summary of the simulations in a data frame.
show(my_sum)

# Example when DLE and efficacy samples are involved.

# The escalation rule using the 'NextBestMaxGainSamples' class.
my_next_best <- NextBestMaxGainSamples(
  prob_target_drt = 0.35,
```

```
  prob_target_eot = 0.3,
  derive = function(samples) {
    as.numeric(quantile(samples, prob = 0.3))
  },
  mg_derive = function(mg_samples) {
    as.numeric(quantile(mg_samples, prob = 0.5))
  }
)

# The design of 'DualResponsesSamplesDesign' class.
my_design <- DualResponsesSamplesDesign(
  nextBest = my_next_best,
  cohort_size = my_size,
  startingDose = 25,
  model = dle_model,
  eff_model = eff_model,
  data = emptydata,
  stopping = my_stopping,
  increments = my_increments
)

# MCMC options.
# For illustration purpose 50 burn-ins to generate 200 samples are used.
my_options <- McmcOptions(burnin = 50, step = 2, samples = 200)

# For illustration purpose 2 trials are simulated.
my_sim <- simulate(
  object = my_design,
  args = NULL,
  trueDLE = my_truth_dle,
  trueEff = my_truth_eff,
  trueNu = 1 / 0.025,
  nsim = 2,
  mcmcOptions = my_options,
  seed = 819,
  parallel = FALSE
)

# Produce a summary of the simulations.
my_sum <- summary(
  my_sim,
  trueDLE = my_truth_dle,
  trueEff = my_truth_eff
)

# Show the summary in data frame for the simulations.
show(my_sum)
```

---

show,PseudoSimulationsSummary-method

Show the summary of the simulations

Description

Show the summary of the simulations

Usage

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

Arguments

object	the PseudoSimulationsSummary object we want to print

Value

invisibly returns a data frame of the results with one row and appropriate column names

Examples

```
# Obtain the plot for the simulation results if only DLE responses are
# considered in the simulations.

# Specified simulations when no DLE samples are used.
emptydata <- Data(doseGrid = seq(25, 300, 25))

# The design only incorporate DLE responses and DLE samples are involved.
# Specify the model of 'ModelTox' class eg 'LogisticIndepBeta' class model.
my_model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = emptydata
)

# The escalation rule.
td_next_best <- NextBestTD(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3
)

# The cohort size is 3 subjects.
my_size <- CohortSizeConst(size = 3)

# Allow increase of 200%.
my_increments <- IncrementsRelative(intervals = 0, increments = 2)

# Specify the stopping rule with maximum sample size of 36 patients or when the
# next dose is NA.
my_stopping <- StoppingMinPatients(nPatients = 36) | StoppingMissingDose()

# Specify the design. (For details please refer to the 'TDDesign' example.)
my_design <- TDDesign(
  model = my_model,
```

```
  nextBest = td_next_best,
  stopping = my_stopping,
  increments = my_increments,
  cohort_size = my_size,
  data = emptydata,
  startingDose = 25
)

# Specify the truth of the DLE responses.
my_truth <- probFunction(my_model, phi1 = -53.66584, phi2 = 10.50499)

# For illustration purpose only 1 simulation is produced.
my_sim <- simulate(
  object = my_design,
  args = NULL,
  truth = my_truth,
  nsim = 1,
  seed = 819,
  parallel = FALSE
)

# Summary of the simulations.
my_sum <- summary(
  my_sim,
  truth = my_truth
)

# Show the summary of the simulated results in a data frame.
show(my_sum)

# Example where DLE samples are involved.

# The escalation rule.
td_next_best <- NextBestTDsamples(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3,
  derive = function(samples) {
    as.numeric(quantile(samples, probs = 0.3))
  }
)

# The design.
my_design <- TDsamplesDesign(
  model = my_model,
  nextBest = td_next_best,
  stopping = my_stopping,
  increments = my_increments,
  cohort_size = my_size,
  data = emptydata,
  startingDose = 25
)

# For illustration purposes 2 trails are simulated with 50 burn-ins to generate
```

```
# 200 samples.
my_options <- McmcOptions(burnin = 50, step = 2, samples = 200)

my_sim <- simulate(
  object = my_design,
  args = NULL,
  truth = my_truth,
  nsim = 2,
  seed = 819,
  mcmcOptions = my_options,
  parallel = FALSE
)

# Produce a summary of the simulations.
my_sum <- summary(
  my_sim,
  truth = my_truth
)

# Show the summary of the simulated results in a data frame.
show(my_sum)
```

`show,SimulationsSummary-method`

Show the summary of the simulations

Description

Show the summary of the simulations

Usage

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

Arguments

`object`	the `SimulationsSummary` object we want to print

Value

invisibly returns a data frame of the results with one row and appropriate column names

Examples

```
# nolint start

# Define the dose-grid
emptydata <- Data(doseGrid = c(1, 3, 5, 10, 15, 20, 25, 40, 50, 80, 100))
```

```
# Initialize the CRM model
model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov =
    matrix(c(1, -0.5, -0.5, 1),
      nrow = 2
    ),
  ref_dose = 56
)

# Choose the rule for selecting the next dose
myNextBest <- NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Choose the rule for the cohort-size
mySize1 <- CohortSizeRange(
  intervals = c(0, 30),
  cohort_size = c(1, 3)
)
mySize2 <- CohortSizeDLT(
  intervals = c(0, 1),
  cohort_size = c(1, 3)
)
mySize <- maxSize(mySize1, mySize2)

# Choose the rule for stopping
myStopping1 <- StoppingMinCohorts(nCohorts = 3)
myStopping2 <- StoppingTargetProb(
  target = c(0.2, 0.35),
  prob = 0.5
)
myStopping3 <- StoppingMinPatients(nPatients = 20)
myStopping <- (myStopping1 & myStopping2) | myStopping3

# Choose the rule for dose increments
myIncrements <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

# Initialize the design
design <- Design(
  model = model,
  nextBest = myNextBest,
  stopping = myStopping,
  increments = myIncrements,
  cohort_size = mySize,
  data = emptydata,
  startingDose = 3
)
```

```
## define the true function
myTruth <- probFunction(model, alpha0 = 7, alpha1 = 8)

# Run the simulation on the desired design
# We only generate 1 trial outcome here for illustration, for the actual study
# this should be increased of course
options <- McmcOptions(
  burnin = 100,
  step = 2,
  samples = 1000
)
time <- system.time(mySims <- simulate(design,
  args = NULL,
  truth = myTruth,
  nsim = 1,
  seed = 819,
  mcmcOptions = options,
  parallel = FALSE
))[3]

# Show the Summary of the Simulations
show(summary(mySims, truth = myTruth))

# nolint end
```

`simulate,DADesign-method`

Simulate outcomes from a time-to-DLT augmented CRM design (`DADesign`*)*

Description

Simulate outcomes from a time-to-DLT augmented CRM design (`DADesign`)

Usage

```
## S4 method for signature 'DADesign'
simulate(
  object,
  nsim = 1L,
  seed = NULL,
  truthTox,
  truthSurv,
  trueTmax = NULL,
  args = NULL,
  firstSeparate = FALSE,
  deescalate = TRUE,
  mcmcOptions = McmcOptions(),
```

```
  DA = TRUE,
  parallel = FALSE,
  nCores = min(parallel::detectCores(), 5),
  derive = list(),
  ...
)
```

Arguments

object	the DADesign object we want to simulate data from
nsim	the number of simulations (default: 1)
seed	see set_seed
truthTox	a function which takes as input a dose (vector) and returns the true probability (vector) for toxicity and the time DLT occurs. Additional arguments can be supplied in args.
truthSurv	a CDF which takes as input a time (vector) and returns the true cumulative probability (vector) that the DLT would occur conditioning on the patient has DLTs.
trueTmax	(number or NULL) the true maximum time at which DLTs can occur. Note that this must be larger thank Tmax from the object's base data, which is the length of the DLT window, i.e. until which time DLTs are officially declared as such and used in the trial.
args	data frame with arguments for the truth function. The column names correspond to the argument names, the rows to the values of the arguments. The rows are appropriately recycled in the nsim simulations. In order to produce outcomes from the posterior predictive distribution, e.g, pass an object that contains the data observed so far, truth contains the prob function from the model in object, and args contains posterior samples from the model.
firstSeparate	enroll the first patient separately from the rest of the cohort? (not default) If yes, the cohort will be closed if a DLT occurs in this patient.
deescalate	deescalation when a DLT occurs in cohorts with lower dose level.
mcmcOptions	object of class McmcOptions, giving the MCMC options for each evaluation in the trial. By default, the standard options are used.
DA	document or rename this parameter to make it more meaningful
parallel	should the simulation runs be parallelized across the clusters of the computer? (not default)
nCores	how many cores should be used for parallel computing? Defaults to the number of cores on the machine (maximum 5)
derive	a named list of functions which derives statistics, based on the vector of posterior MTD samples. Each list element must therefore accept one and only one argument, which is a numeric vector, and return a number.
...	not used

Value

an object of class Simulations

Examples

```
# nolint start

# Define the dose-grid and PEM parameters
emptydata <- DataDA(
  doseGrid = c(0.1, 0.5, 1, 1.5, 3, 6, seq(from = 10, to = 80, by = 2)),
  Tmax = 60
)

# Initialize the mDA-CRM model
npiece_ <- 10
Tmax_ <- 60

lambda_prior <- function(k) {
  npiece_ / (Tmax_ * (npiece_ - k + 0.5))
}

model <- DALogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 56,
  npiece = npiece_,
  l = as.numeric(t(apply(as.matrix(c(1:npiece_), 1, npiece_), 2, lambda_prior))),
  c_par = 2
)
# Choose the rule for dose increments
myIncrements <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

myNextBest <- NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Choose the rule for the cohort-size
mySize1 <- CohortSizeRange(
  intervals = c(0, 30),
  cohort_size = c(1, 3)
)
mySize2 <- CohortSizeDLT(
  intervals = c(0, 1),
  cohort_size = c(1, 3)
)
mySize <- maxSize(mySize1, mySize2)

# Choose the rule for stopping
myStopping1 <- StoppingTargetProb(
  target = c(0.2, 0.35),
  prob = 0.5
```

```
)
myStopping2 <- StoppingMinPatients(nPatients = 50)

myStopping <- (myStopping1 | myStopping2)

# Choose the safety window
mysafetywindow <- SafetyWindowConst(c(6, 2), 7, 7)

# Initialize the design
design <- DADesign(
  model = model,
  increments = myIncrements,
  nextBest = myNextBest,
  stopping = myStopping,
  cohort_size = mySize,
  data = emptydata,
  safetyWindow = mysafetywindow,
  startingDose = 3
)

## set up truth curves
myTruth <- probFunction(model, alpha0 = 2, alpha1 = 3)
curve(myTruth(x), from = 0, to = 100, ylim = c(0, 1))

exp_cond.cdf <- function(x, onset = 15) {
  a <- pexp(28, 1 / onset, lower.tail = FALSE)
  1 - (pexp(x, 1 / onset, lower.tail = FALSE) - a) / (1 - a)
}

# set up simulation settings
options <- McmcOptions(
  burnin = 10,
  step = 1,
  samples = 200
)

mySims <- simulate(design,
  args = NULL,
  truthTox = myTruth,
  truthSurv = exp_cond.cdf,
  trueTmax = 80,
  nsim = 2,
  seed = 819,
  mcmcOptions = options,
  firstSeparate = TRUE,
  deescalate = FALSE,
  parallel = FALSE
)

# nolint end
```

simulate,Design-method

Simulate outcomes from a CRM design

Description

Simulate outcomes from a CRM design

Usage

```
## S4 method for signature 'Design'
simulate(
  object,
  nsim = 1L,
  seed = NULL,
  truth,
  args = NULL,
  firstSeparate = FALSE,
  mcmcOptions = McmcOptions(),
  parallel = FALSE,
  nCores = min(parallel::detectCores(), 5),
  derive = list(),
  ...
)
```

Arguments

object	the Design object we want to simulate data from
nsim	the number of simulations (default: 1)
seed	see set_seed
truth	a function which takes as input a dose (vector) and returns the true probability (vector) for toxicity. Additional arguments can be supplied in args.
args	data frame with arguments for the truth function. The column names correspond to the argument names, the rows to the values of the arguments. The rows are appropriately recycled in the nsim simulations. In order to produce outcomes from the posterior predictive distribution, e.g, pass an object that contains the data observed so far, truth contains the prob function from the model in object, and args contains posterior samples from the model.
firstSeparate	enroll the first patient separately from the rest of the cohort? (not default) If yes, the cohort will be closed if a DLT occurs in this patient.
mcmcOptions	object of class McmcOptions, giving the MCMC options for each evaluation in the trial. By default, the standard options are used
parallel	should the simulation runs be parallelized across the clusters of the computer? (not default)

nCores	how many cores should be used for parallel computing? Defaults to the number of cores on the machine, maximum 5.
derive	a named list of functions which derives statistics, based on the vector of posterior MTD samples. Each list element must therefore accept one and only one argument, which is a numeric vector, and return a number.
...	not used

Value

an object of class Simulations

Examples

```
# nolint start

# Define the dose-grid
emptydata <- Data(doseGrid = c(1, 3, 5, 10, 15, 20, 25, 40, 50, 80, 100))

# Initialize the CRM model
model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov =
    matrix(c(1, -0.5, -0.5, 1),
      nrow = 2
    ),
  ref_dose = 56
)

# Choose the rule for selecting the next dose
myNextBest <- NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Choose the rule for the cohort-size
mySize1 <- CohortSizeRange(
  intervals = c(0, 30),
  cohort_size = c(1, 3)
)
mySize2 <- CohortSizeDLT(
  intervals = c(0, 1),
  cohort_size = c(1, 3)
)
mySize <- maxSize(mySize1, mySize2)

# Choose the rule for stopping
myStopping1 <- StoppingMinCohorts(nCohorts = 3)
myStopping2 <- StoppingTargetProb(
  target = c(0.2, 0.35),
  prob = 0.5
)
```

```
myStopping3 <- StoppingMinPatients(nPatients = 20)
myStopping <- (myStopping1 & myStopping2) | myStopping3

# Choose the rule for dose increments
myIncrements <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

# Initialize the design
design <- Design(
  model = model,
  nextBest = myNextBest,
  stopping = myStopping,
  increments = myIncrements,
  cohort_size = mySize,
  data = emptydata,
  startingDose = 3
)

## define the true function
myTruth <- probFunction(model, alpha0 = 7, alpha1 = 8)

# Run the simulation on the desired design
# We only generate 1 trial outcomes here for illustration, for the actual study
# this should be increased of course
options <- McmcOptions(
  burnin = 100,
  step = 1,
  samples = 2000
)

time <- system.time(mySims <- simulate(design,
  args = NULL,
  truth = myTruth,
  nsim = 1,
  seed = 819,
  mcmcOptions = options,
  parallel = FALSE
))[3]

# nolint end
```

`simulate,DesignGrouped-method`

Simulate Method for the `DesignGrouped` *Class*

Description

[Experimental]

A simulate method for `DesignGrouped` designs.

Usage

```
## S4 method for signature 'DesignGrouped'
simulate(
  object,
  nsim = 1L,
  seed = NULL,
  truth,
  combo_truth,
  args = data.frame(),
  firstSeparate = FALSE,
  mcmcOptions = McmcOptions(),
  parallel = FALSE,
  nCores = min(parallelly::availableCores(), 5),
  ...
)
```

Arguments

object	(DesignGrouped) the design we want to simulate trials from.
nsim	(number) how many trials should be simulated.
seed	(RNGstate) generated with set_seed().
truth	(function) a function which takes as input a dose (vector) and returns the true probability (vector) for toxicity for the mono arm. Additional arguments can be supplied in args.
combo_truth	(function) same as truth but for the combo arm.
args	(data.frame) optional data.frame with arguments that work for both the truth and combo_truth functions. The column names correspond to the argument names, the rows to the values of the arguments. The rows are appropriately recycled in the nsim simulations.
firstSeparate	(flag) whether to enroll the first patient separately from the rest of the cohort and close the cohort in case a DLT occurs in this first patient.
mcmcOptions	(McmcOptions) MCMC options for each evaluation in the trial.
parallel	(flag) whether the simulation runs are parallelized across the cores of the computer.
nCores	(number) how many cores should be used for parallel computing.
...	not used.

Value

A list of mono and combo simulation results as Simulations objects.

Examples

```
# Assemble ingredients for our group design.
my_stopping <- StoppingTargetProb(target = c(0.2, 0.35), prob = 0.5) |
  StoppingMinPatients(10) |
  StoppingMissingDose()
my_increments <- IncrementsDoseLevels(levels = 3L)
my_next_best <- NextBestNCRM(
  target = c(0.2, 0.3),
  overdose = c(0.3, 1),
  max_overdose_prob = 0.3
)
my_cohort_size <- CohortSizeConst(3)
empty_data <- Data(doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2)))
my_model <- LogisticLogNormalGrouped(
  mean = c(-4, -4, -4, -4),
  cov = diag(rep(6, 4)),
  ref_dose = 0.1
)

# Put together the design. Note that if we only specify the mono arm,
# then the combo arm is having the same settings.
my_design <- DesignGrouped(
  model = my_model,
  mono = Design(
    model = my_model,
    stopping = my_stopping,
    increments = my_increments,
    nextBest = my_next_best,
    cohort_size = my_cohort_size,
    data = empty_data,
    startingDose = 0.1
  ),
  first_cohort_mono_only = TRUE,
  same_dose_for_all = TRUE
)

# Set up a realistic simulation scenario.
my_truth <- function(x) plogis(-4 + 0.2 * log(x / 0.1))
my_combo_truth <- function(x) plogis(-4 + 0.5 * log(x / 0.1))
matplot(
  x = empty_data@doseGrid,
  y = cbind(
    mono = my_truth(empty_data@doseGrid),
    combo = my_combo_truth(empty_data@doseGrid)
  ),
  type = "l",
  ylab = "true DLT prob",
  xlab = "dose"
```

```
)
legend("topright", c("mono", "combo"), lty = c(1, 2), col = c(1, 2))

# Start the simulations.
set.seed(123)
my_sims <- simulate(
  my_design,
  nsim = 1, # This should be at least 100 in actual applications.
  seed = 123,
  truth = my_truth,
  combo_truth = my_combo_truth
)

# Looking at the summary of the simulations:
mono_sims_sum <- summary(my_sims$mono, truth = my_truth)
combo_sims_sum <- summary(my_sims$combo, truth = my_combo_truth)

mono_sims_sum
combo_sims_sum

plot(mono_sims_sum)
plot(combo_sims_sum)

# Looking at specific simulated trials:
trial_index <- 1
plot(my_sims$mono@data[[trial_index]])
plot(my_sims$combo@data[[trial_index]])
```

`simulate,DualDesign-method` *Simulate outcomes from a dual-endpoint design*

Description

Simulate outcomes from a dual-endpoint design

Usage

```
## S4 method for signature 'DualDesign'
simulate(
  object,
  nsim = 1L,
  seed = NULL,
  trueTox,
  trueBiomarker,
  args = NULL,
  sigma2W,
  rho = 0,
  firstSeparate = FALSE,
```

```
  mcmcOptions = McmcOptions(),
  parallel = FALSE,
  nCores = min(parallel::detectCores(), 5),
  derive = list(),
  ...
)
```

Arguments

object	the DualDesign object we want to simulate data from
nsim	the number of simulations (default: 1)
seed	see set_seed
trueTox	a function which takes as input a dose (vector) and returns the true probability (vector) for toxicity. Additional arguments can be supplied in args.
trueBiomarker	a function which takes as input a dose (vector) and returns the true biomarker level (vector). Additional arguments can be supplied in args.
args	data frame with arguments for the trueTox and trueBiomarker function. The column names correspond to the argument names, the rows to the values of the arguments. The rows are appropriately recycled in the nsim simulations.
sigma2W	variance for the biomarker measurements
rho	correlation between toxicity and biomarker measurements (default: 0)
firstSeparate	enroll the first patient separately from the rest of the cohort? (not default) If yes, the cohort will be closed if a DLT occurs in this patient.
mcmcOptions	object of class McmcOptions, giving the MCMC options for each evaluation in the trial. By default, the standard options are used
parallel	should the simulation runs be parallelized across the clusters of the computer? (not default)
nCores	how many cores should be used for parallel computing? Defaults to the number of cores on the machine, maximum 5.
derive	a named list of functions which derives statistics, based on the vector of posterior MTD samples. Each list element must therefore accept one and only one argument, which is a numeric vector, and return a number.
...	not used

Value

an object of class DualSimulations

Examples

```
# nolint start

# Define the dose-grid
emptydata <- DataDual(doseGrid = c(1, 3, 5, 10, 15, 20, 25, 40, 50, 80, 100))

# Initialize the CRM model
```

```
model <- DualEndpointRW(
  mean = c(0, 1),
  cov = matrix(c(1, 0, 0, 1), nrow = 2),
  sigma2betaW = 0.01,
  sigma2W = c(a = 0.1, b = 0.1),
  use_log_dose = TRUE,
  ref_dose = 2,
  rho = c(a = 1, b = 1),
  rw1 = TRUE
)

# Choose the rule for selecting the next dose
myNextBest <- NextBestDualEndpoint(
  target = c(0.9, 1),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Choose the rule for the cohort-size
mySize1 <- CohortSizeRange(
  intervals = c(0, 30),
  cohort_size = c(1, 3)
)
mySize2 <- CohortSizeDLT(
  intervals = c(0, 1),
  cohort_size = c(1, 3)
)
mySize <- maxSize(mySize1, mySize2)

# Choose the rule for stopping
myStopping4 <- StoppingTargetBiomarker(
  target = c(0.9, 1),
  prob = 0.5
)
myStopping <- myStopping4 | StoppingMinPatients(10)

# Choose the rule for dose increments
myIncrements <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

# Initialize the design
design <- DualDesign(
  model = model,
  data = emptydata,
  nextBest = myNextBest,
  stopping = myStopping,
  increments = myIncrements,
  cohort_size = mySize,
  startingDose = 3
)
```

```
# define scenarios for the TRUE toxicity and efficacy profiles
betaMod <- function(dose, e0, eMax, delta1, delta2, scal) {
  maxDens <- (delta1^delta1) * (delta2^delta2) / ((delta1 + delta2)^(delta1 + delta2))
  dose <- dose / scal
  e0 + eMax / maxDens * (dose^delta1) * (1 - dose)^delta2
}

trueBiomarker <- function(dose) {
  betaMod(dose, e0 = 0.2, eMax = 0.6, delta1 = 5, delta2 = 5 * 0.5 / 0.5, scal = 100)
}

trueTox <- function(dose) {
  pnorm((dose - 60) / 10)
}

# Draw the TRUE profiles
par(mfrow = c(1, 2))
curve(trueTox(x), from = 0, to = 80)
curve(trueBiomarker(x), from = 0, to = 80)

# Run the simulation on the desired design
# We only generate 1 trial outcome here for illustration, for the actual study
# this should be increased of course, similarly for the McmcOptions -
# they also need to be increased.
mySims <- simulate(design,
  trueTox = trueTox,
  trueBiomarker = trueBiomarker,
  sigma2W = 0.01,
  rho = 0,
  nsim = 1,
  parallel = FALSE,
  seed = 3,
  startingDose = 6,
  mcmcOptions =
    McmcOptions(
      burnin = 100,
      step = 1,
      samples = 300
    )
)

# nolint end
```

---

simulate,DualResponsesDesign-method

This is a methods to simulate dose escalation procedure using both DLE and efficacy responses. This is a method based on the DualResponsesDesign *where DLEmodel used are of* ModelTox *class object and efficacy model used are of* ModelEff *class object. In addition, no DLE and efficacy samples are involved or generated in the simulation process*

Description

This is a methods to simulate dose escalation procedure using both DLE and efficacy responses. This is a method based on the DualResponsesDesign where DLEmodel used are of ModelTox class object and efficacy model used are of ModelEff class object. In addition, no DLE and efficacy samples are involved or generated in the simulation process

Usage

```
## S4 method for signature 'DualResponsesDesign'
simulate(
  object,
  nsim = 1L,
  seed = NULL,
  trueDLE,
  trueEff,
  trueNu,
  args = NULL,
  firstSeparate = FALSE,
  parallel = FALSE,
  nCores = min(parallel::detectCores(), 5L),
  ...
)
```

Arguments

object	the DualResponsesDesign object we want to simulate the data from
nsim	the number of simulations (default :1)
seed	see set_seed
trueDLE	a function which takes as input a dose (vector) and returns the true probability (vector) of the occurrence of a DLE. Additional arguments can be supplied in args.
trueEff	a function which takes as input a dose (vector) and returns the expected efficacy responses (vector). Additional arguments can be supplied in args.
trueNu	the precision, the inverse of the variance of the efficacy responses
args	data frame with arguments for the trueDLE and trueEff function. The column names correspond to the argument names, the rows to the values of the arguments. The rows are appropriately recycled in the nsim simulations.
firstSeparate	enroll the first patient separately from the rest of the cohort? (not default) If yes, the cohort will be closed if a DLT occurs in this patient.
parallel	should the simulation runs be parallelized across the clusters of the computer? (not default)
nCores	how many cores should be used for parallel computing? Defaults to the number of cores on the machine, maximum 5.
...	not used

Value

an object of class PseudoDualSimulations

Examples

```
# nolint start

## Simulate dose-escalation procedure based on DLE and efficacy responses where no DLE
## and efficacy samples are used
## we need a data object with doses >= 1:
data <- DataDual(doseGrid = seq(25, 300, 25), placebo = FALSE)
## First for the DLE model
## The DLE model must be of 'ModelTox' (e.g 'LogisticIndepBeta') class
DLEmodel <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = data
)

## The efficacy model of 'ModelEff' (e.g 'Effloglog') class
Effmodel <- Effloglog(
  eff = c(1.223, 2.513), eff_dose = c(25, 300),
  nu = c(a = 1, b = 0.025), data = data
)

## The escalation rule using the 'NextBestMaxGain' class
mynextbest <- NextBestMaxGain(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3
)


## The increments (see Increments class examples)
## 200% allowable increase for dose below 300 and 200% increase for dose above 300
myIncrements <- IncrementsRelative(
  intervals = c(25, 300),
  increments = c(2, 2)
)
## cohort size of 3
mySize <- CohortSizeConst(size = 3)
## Stop only when 36 subjects are treated
myStopping <- StoppingMinPatients(nPatients = 36)
## Now specified the design with all the above information and starting with a dose of 25

## Specified the design(for details please refer to the 'DualResponsesDesign' example)
design <- DualResponsesDesign(
  nextBest = mynextbest,
  model = DLEmodel,
  eff_model = Effmodel,
  stopping = myStopping,
  increments = myIncrements,
```

```
  cohort_size = mySize,
  data = data, startingDose = 25
)
## Specify the true DLE and efficacy curves
myTruthDLE <- probFunction(DLEmodel, phi1 = -53.66584, phi2 = 10.50499)
myTruthEff <- efficacyFunction(Effmodel, theta1 = -4.818429, theta2 = 3.653058)

## The true gain curve can also be seen
myTruthGain <- function(dose) {
  return((myTruthEff(dose)) / (1 + (myTruthDLE(dose) / (1 - myTruthDLE(dose)))))
}


## Then specified the simulations and generate the trial
## For illustration purpose only 1 simulation is produced (nsim=1).
options <- McmcOptions(burnin = 100, step = 2, samples = 200)
mySim <- simulate(
  object = design,
  args = NULL,
  trueDLE = myTruthDLE,
  trueEff = myTruthEff,
  trueNu = 1 / 0.025,
  nsim = 1,
  seed = 819,
  parallel = FALSE
)

# nolint end
```

`simulate,DualResponsesSamplesDesign-method`

This is a methods to simulate dose escalation procedure using both DLE and efficacy responses. This is a method based on the `DualResponsesSamplesDesign` *where DLEmodel used are of* `ModelTox` *class object and efficacy model used are of* `ModelEff` *class object (special case is* `EffFlexi` *class model object). In addition, DLE and efficacy samples are involved or generated in the simulation process*

Description

This is a methods to simulate dose escalation procedure using both DLE and efficacy responses. This is a method based on the `DualResponsesSamplesDesign` where DLEmodel used are of `ModelTox` class object and efficacy model used are of `ModelEff` class object (special case is `EffFlexi` class model object). In addition, DLE and efficacy samples are involved or generated in the simulation process

Usage

```
## S4 method for signature 'DualResponsesSamplesDesign'
simulate(
  object,
  nsim = 1L,
  seed = NULL,
  trueDLE,
  trueEff,
  trueNu = NULL,
  trueSigma2 = NULL,
  trueSigma2betaW = NULL,
  args = NULL,
  firstSeparate = FALSE,
  mcmcOptions = McmcOptions(),
  parallel = FALSE,
  nCores = min(parallel::detectCores(), 5L),
  ...
)
```

Arguments

object	the DualResponsesSamplesDesign object we want to simulate the data from
nsim	the number of simulations (default :1)
seed	see set_seed
trueDLE	a function which takes as input a dose (vector) and returns the true probability (vector) of the occurrence of a DLE. Additional arguments can be supplied in args.
trueEff	a function which takes as input a dose (vector) and returns the expected efficacy responses (vector). Additional arguments can be supplied in args.
trueNu	(not with codeEffFlexi) the precision, the inverse of the variance of the efficacy responses
trueSigma2	(only with codeEffFlexi) the true variance of the efficacy responses which must be a single positive scalar.
trueSigma2betaW	(only with codeEffFlexi) the true variance for the random walk model used for smoothing. This must be a single positive scalar.
args	data frame with arguments for the trueDLE and trueEff function. The column names correspond to the argument names, the rows to the values of the arguments. The rows are appropriately recycled in the nsim simulations.
firstSeparate	enroll the first patient separately from the rest of the cohort? (not default) If yes, the cohort will be closed if a DLT occurs in this patient.
mcmcOptions	object of class McmcOptions, giving the MCMC options for each evaluation in the trial. By default, the standard options are used
parallel	should the simulation runs be parallelized across the clusters of the computer? (not default)

nCores	how many cores should be used for parallel computing? Defaults to the number of cores on the machine, maximum 5.
...	not used.

Value

an object of class PseudoDualSimulations or PseudoDualFlexiSimulations

Examples

```
# nolint start

## Simulate dose-escalation procedure based on DLE and efficacy responses where DLE
## and efficacy samples are used
data <- DataDual(doseGrid = seq(25, 300, 25), placebo = FALSE)
## First for the DLE model
## The DLE model must be of 'ModelTox' (e.g 'LogisticIndepBeta') class
DLEmodel <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = data
)

## The efficacy model of 'ModelEff' (e.g 'Effloglog') class
Effmodel <- Effloglog(
  eff = c(1.223, 2.513), eff_dose = c(25, 300),
  nu = c(a = 1, b = 0.025), data = data
)


## The escalation rule using the 'NextBestMaxGainSamples' class
mynextbest <- NextBestMaxGainSamples(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3,
  derive = function(samples) {
    as.numeric(quantile(samples, prob = 0.3))
  },
  mg_derive = function(mg_samples) {
    as.numeric(quantile(mg_samples, prob = 0.5))
  }
)

## The increments (see Increments class examples)
## 200% allowable increase for dose below 300 and 200% increase for dose above 300
myIncrements <- IncrementsRelative(
  intervals = c(25, 300),
  increments = c(2, 2)
)
## cohort size of 3
mySize <- CohortSizeConst(size = 3)
## Stop only when 10 subjects are treated (only for illustration such a low
```

```
## sample size)
myStopping <- StoppingMinPatients(nPatients = 10)
## Now specified the design with all the above information and starting with
## a dose of 25


## Specified the design
design <- DualResponsesSamplesDesign(
  nextBest = mynextbest,
  cohort_size = mySize,
  startingDose = 25,
  model = DLEmodel,
  eff_model = Effmodel,
  data = data,
  stopping = myStopping,
  increments = myIncrements
)
## specified the true DLE and efficacy curve
myTruthDLE <- probFunction(DLEmodel, phi1 = -53.66584, phi2 = 10.50499)
myTruthEff <- efficacyFunction(Effmodel, theta1 = -4.818429, theta2 = 3.653058)

## The true gain curve can also be seen
myTruthGain <- function(dose) {
  return((myTruthEff(dose)) / (1 + (myTruthDLE(dose) / (1 - myTruthDLE(dose)))))
}

## simulate the trial for 10 times involving samples
## for illustration purpose we use 10 burn-ins to generate 50 samples
options <- McmcOptions(burnin = 10, step = 1, samples = 50)
## For illustration purpose only 1 simulations are produced (nsim=1).
mySim <- simulate(design,
  args = NULL,
  trueDLE = myTruthDLE,
  trueEff = myTruthEff,
  trueNu = 1 / 0.025,
  nsim = 1,
  mcmcOptions = options,
  seed = 819,
  parallel = FALSE
)



## Simulate dose-escalation procedure based on DLE and efficacy responses where DLE
## and efficacy samples are used
## when the efficacy model is of 'EffFlexi' class
Effmodel <- EffFlexi(
  eff = c(1.223, 2.513), eff_dose = c(25, 300),
  sigma2W = c(a = 0.1, b = 0.1), sigma2betaW = c(a = 20, b = 50), rw1 = FALSE, data = data
)


## Specified the design
```

```
design <- DualResponsesSamplesDesign(
  nextBest = mynextbest,
  cohort_size = mySize,
  startingDose = 25,
  model = DLEmodel,
  eff_model = Effmodel,
  data = data,
  stopping = myStopping,
  increments = myIncrements
)
## specified the true DLE curve and the true expected efficacy values at all dose levels
myTruthDLE <- probFunction(DLEmodel, phi1 = -53.66584, phi2 = 10.50499)

myTruthEff <- c(
  -0.5478867, 0.1645417, 0.5248031, 0.7604467,
  0.9333009, 1.0687031, 1.1793942, 1.2726408,
  1.3529598, 1.4233411, 1.4858613, 1.5420182
)
## The true gain curve can also be seen
d1 <- data@doseGrid
myTruthGain <- (myTruthEff) / (1 + (myTruthDLE(d1) / (1 - myTruthDLE(d1))))


mySim <- simulate(
  object = design,
  args = NULL,
  trueDLE = myTruthDLE,
  trueEff = myTruthEff,
  trueSigma2 = 0.025,
  trueSigma2betaW = 1,
  mcmcOptions = options,
  nsim = 1,
  seed = 819,
  parallel = FALSE
)

# nolint end
```

`simulate,RuleDesign-method`

Simulate outcomes from a rule-based design

Description

Simulate outcomes from a rule-based design

Usage

```
## S4 method for signature 'RuleDesign'
simulate(
```

```
  object,
  nsim = 1L,
  seed = NULL,
  truth,
  args = NULL,
  parallel = FALSE,
  nCores = min(parallel::detectCores(), 5L),
  ...
)
```

Arguments

object	the RuleDesign object we want to simulate data from
nsim	the number of simulations (default: 1)
seed	see set_seed
truth	a function which takes as input a dose (vector) and returns the true probability (vector) for toxicity. Additional arguments can be supplied in args.
args	data frame with arguments for the truth function. The column names correspond to the argument names, the rows to the values of the arguments. The rows are appropriately recycled in the nsim simulations.
parallel	should the simulation runs be parallelized across the clusters of the computer? (not default)
nCores	how many cores should be used for parallel computing? Defaults to the number of cores on the machine, maximum 5.
...	not used

Value

an object of class GeneralSimulations

Examples

```
# nolint start

# Define the dose-grid
emptydata <- Data(doseGrid = c(5, 10, 15, 25, 35, 50, 80))

# inizialing a 3+3 design with constant cohort size of 3 and
# starting dose equal 5
myDesign <- RuleDesign(
  nextBest = NextBestThreePlusThree(),
  cohort_size = CohortSizeConst(size = 3L),
  data = emptydata,
  startingDose = 5
)

model <- LogisticLogNormal(
  mean = c(-0.85, 1),
```

```
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 50
)

## define the true function
myTruth <- probFunction(model, alpha0 = 7, alpha1 = 8)

# Perform the simulation
## For illustration purpose only 10 simulation is produced (nsim=10).
threeSims <- simulate(myDesign,
  nsim = 10,
  seed = 35,
  truth = myTruth,
  parallel = FALSE
)

# nolint end
```

---

simulate,TDDesign-method

This is a methods to simulate dose escalation procedure only using the DLE responses. This is a method based on the TDDesign *where model used are of* ModelTox *class object and no samples are involved.*

Description

This is a methods to simulate dose escalation procedure only using the DLE responses. This is a method based on the TDDesign where model used are of ModelTox class object and no samples are involved.

Usage

```
## S4 method for signature 'TDDesign'
simulate(
  object,
  nsim = 1L,
  seed = NULL,
  truth,
  args = NULL,
  firstSeparate = FALSE,
  parallel = FALSE,
  nCores = min(parallel::detectCores(), 5L),
  ...
)
```

Arguments

object	the TDDesign object we want to simulate the data from

nsim	the number of simulations (default :1)
seed	see set_seed
truth	a function which takes as input a dose (vector) and returns the true probability (vector) of the occurrence of a DLE. Additional arguments can be supplied in args.
args	data frame with arguments for the truth function. The column names correspond to the argument names, the rows to the values of the arguments. The rows are appropriately recycled in the nsim simulations. In order to produce outcomes from the posterior predictive distribution, e.g, pass an object that contains the data observed so far, truth contains the prob function from the model in object, and args contains posterior samples from the model.
firstSeparate	enroll the first patient separately from the rest of the cohort? (not default) If yes, the cohort will be closed if a DLT occurs in this patient.
parallel	should the simulation runs be parallelized across the clusters of the computer? (not default)
nCores	how many cores should be used for parallel computing? Defaults to the number of cores on the machine, maximum 5.
...	not used

Value

an object of class PseudoSimulations

@export @keywords methods

Examples

```
# nolint start

## Simulate dose-escalation procedure based only on DLE responses and no DLE samples are used

## The design comprises a model, the escalation rule, starting data,
## a cohort size and a starting dose
## Define your data set first using an empty data set
## with dose levels from 25 to 300 with increments 25
data <- Data(doseGrid = seq(25, 300, 25))

## The design only incorporate DLE responses and DLE samples are involved
## Specified the model of 'ModelTox' class eg 'LogisticIndepBeta' class model
model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = data
)
## Then the escalation rule
tdNextBest <- NextBestTD(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3
```

```
)
doseRecommendation <- nextBest(tdNextBest,
  doselimit = max(data@doseGrid),
  model = model,
  data = data
)
## Then the starting data, an empty data set
emptydata <- Data(doseGrid = seq(25, 300, 25))
## The cohort size, size of 3 subjects
mySize <- CohortSizeConst(size = 3)
## Deifne the increments for the dose-escalation process
## The maximum increase of 200% for doses up to the maximum of the dose specified in the doseGrid
## The maximum increase of 200% for dose above the maximum of the dose specified in the doseGrid
## This is to specified a maximum of 3-fold restriction in dose-esclation
myIncrements <- IncrementsRelative(
  intervals = c(min(data@doseGrid), max(data@doseGrid)),
  increments = c(2, 2)
)
## Specified the stopping rule e.g stop when the maximum sample size of 36 patients has been reached
myStopping <- StoppingMinPatients(nPatients = 36)


## Specified the design(for details please refer to the 'TDDesign' example)
design <- TDDesign(
  model = model,
  nextBest = tdNextBest,
  stopping = myStopping,
  increments = myIncrements,
  cohort_size = mySize,
  data = data, startingDose = 25
)

## Specify the truth of the DLE responses
myTruth <- probFunction(model, phi1 = -53.66584, phi2 = 10.50499)

## then plot the truth to see how the truth dose-DLE curve look like
curve(myTruth(x), from = 0, to = 300, ylim = c(0, 1))


## For illustration purpose only 1 simulation is produced (nsim=1).
mySim <- simulate(
  object = design,
  args = NULL,
  truth = myTruth,
  nsim = 1,
  seed = 819,
  parallel = FALSE
)

# nolint end
```

simulate,TDsamplesDesign-method

This is a methods to simulate dose escalation procedure only using the DLE responses. This is a method based on the TDsamplesDesign *where model used are of* ModelTox *class object DLE samples are also used*

Description

This is a methods to simulate dose escalation procedure only using the DLE responses. This is a method based on the TDsamplesDesign where model used are of ModelTox class object DLE samples are also used

Usage

```
## S4 method for signature 'TDsamplesDesign'
simulate(
  object,
  nsim = 1L,
  seed = NULL,
  truth,
  args = NULL,
  firstSeparate = FALSE,
  mcmcOptions = McmcOptions(),
  parallel = FALSE,
  nCores = min(parallel::detectCores(), 5L),
  ...
)
```

Arguments

object	the TDsamplesDesign object we want to simulate the data from
nsim	the number of simulations (default :1)
seed	see set_seed
truth	a function which takes as input a dose (vector) and returns the true probability (vector) of the occurrence of a DLE. Additional arguments can be supplied in args.
args	data frame with arguments for the truth function. The column names correspond to the argument names, the rows to the values of the arguments. The rows are appropriately recycled in the nsim simulations. In order to produce outcomes from the posterior predictive distribution, e.g, pass an object that contains the data observed so far, truth contains the prob function from the model in object, and args contains posterior samples from the model.
firstSeparate	enroll the first patient separately from the rest of the cohort? (not default) If yes, the cohort will be closed if a DLT occurs in this patient.

mcmcOptions	object of class McmcOptions, giving the MCMC options for each evaluation in the trial. By default, the standard options are used
parallel	should the simulation runs be parallelized across the clusters of the computer? (not default)
nCores	how many cores should be used for parallel computing? Defaults to the number of cores on the machine, maximum 5.
...	not used

Value

an object of class PseudoSimulations

@export @keywords methods

Examples

```
# nolint start

## Simulate dose-escalation procedure based only on DLE responses with DLE samples involved

## The design comprises a model, the escalation rule, starting data,
## a cohort size and a starting dose
## Define your data set first using an empty data set
## with dose levels from 25 to 300 with increments 25
data <- Data(doseGrid = seq(25, 300, 25))

## The design only incorporate DLE responses and DLE samples are involved
## Specified the model of 'ModelTox' class eg 'LogisticIndepBeta' class model
model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = data
)
## Then the escalation rule
tdNextBest <- NextBestTDsamples(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3,
  derive = function(samples) {
    as.numeric(quantile(samples, probs = 0.3))
  }
)

## The cohort size, size of 3 subjects
mySize <- CohortSizeConst(size = 3)
## Deifne the increments for the dose-escalation process
## The maximum increase of 200% for doses up to the maximum of the dose specified in the doseGrid
## The maximum increase of 200% for dose above the maximum of the dose specified in the doseGrid
## This is to specified a maximum of 3-fold restriction in dose-esclation
myIncrements <- IncrementsRelative(
  intervals = c(min(data@doseGrid), max(data@doseGrid)),
  increments = c(2, 2)
```

```
)
## Specified the stopping rule e.g stop when the maximum sample size of 36 patients has been reached
myStopping <- StoppingMinPatients(nPatients = 36)

## Specified the design(for details please refer to the 'TDsamplesDesign' example)
design <- TDsamplesDesign(
  model = model,
  nextBest = tdNextBest,
  stopping = myStopping,
  increments = myIncrements,
  cohort_size = mySize,
  data = data, startingDose = 25
)

## Specify the truth of the DLE responses
myTruth <- probFunction(model, phi1 = -53.66584, phi2 = 10.50499)

## then plot the truth to see how the truth dose-DLE curve look like
curve(myTruth(x), from = 0, to = 300, ylim = c(0, 1))

## Then specified the simulations and generate the trial
## options for MCMC
options <- McmcOptions(burnin = 100, step = 2, samples = 200)
## The simulations
## For illustration purpose only 1 simulation is produced (nsim=1).
mySim <- simulate(
  object = design,
  args = NULL,
  truth = myTruth,
  nsim = 1,
  seed = 819,
  mcmcOptions = options,
  parallel = FALSE
)

# nolint end
```

`Simulations-class` `Simulations`

Description

[Stable]

This class captures the trial simulations from model based designs. Additional slots `fit`, `stop_reasons`, `stop_report`,`additional_stats` compared to the general class `GeneralSimulations`.

Usage

```
Simulations(fit, stop_reasons, stop_report, additional_stats, ...)

.DefaultSimulations()
```

Arguments

`fit`	(`list`) see slot definition.
`stop_reasons`	(`list`) see slot definition.
`stop_report`	see `Simulations`
`additional_stats`	(`list`) see slot definition.
`...`	additional parameters from `GeneralSimulations`

Slots

`fit` (`list`)
: final fits

`stop_reasons` (`list`)
: stopping reasons for each simulation run

`stop_report` matrix of stopping rule outcomes

`additional_stats` list of additional statistical summary

Note

Typically, end users will not use the `.DefaultSimulations()` function.

Examples

```
data <- list(
  Data(
    x = 1:2,
    y = 0:1,
    doseGrid = 1:2,
    ID = 1L:2L,
    cohort = 1L:2L
  ),
  Data(
    x = 3:4,
    y = 0:1,
    doseGrid = 3:4,
    ID = 1L:2L,
    cohort = 1L:2L
  )
)

doses <- c(1, 2)

seed <- as.integer(123)

fit <- list(
  c(0.1, 0.2),
```

```
  c(0.3, 0.4)
)

stop_report <- matrix(c(TRUE, FALSE), nrow = 2)

stop_reasons <- list("A", "B")

additional_stats <- list(a = 1, b = 1)

simulations <- Simulations(
  fit = fit,
  stop_report = stop_report,
  stop_reasons = stop_reasons,
  additional_stats = additional_stats,
  data,
  doses,
  seed
)
```

`SimulationsSummary-class` `SimulationsSummary`

Description

[Stable]

In addition to the slots in the parent class `GeneralSimulationsSummary`, it contains two slots with model fit information.

Usage

```
.DefaultSimulationsSummary()
```

Slots

`stop_report` (`matrix`)
: matrix of stopping rule outcomes

`fit_at_dose_most_selected` (`numeric`)
: fitted toxicity rate at dose most often selected

`additional_stats` (`list`)
: list of additional statistical summary

`mean_fit` (`list`)
: list with the average, lower (2.5%) and upper (97.5%) quantiles of the mean fitted toxicity at each dose level

Note

Typically, end users will not use the `.DefaultSimulationsSummary()` function.

size *Size of an Object*

Description

[Stable]

A method that computes the size of a given object. This can be for instance a size of a MCMC sample, or the size of a cohort. See the help of a specific method for more details.

Usage

```
size(object, ...)

## S4 method for signature 'McmcOptions'
size(object, ...)

## S4 method for signature 'CohortSizeRange'
size(object, dose, data)

## S4 method for signature 'CohortSizeDLT'
size(object, dose, data)

## S4 method for signature 'CohortSizeMax'
size(object, dose, data)

## S4 method for signature 'CohortSizeMin'
size(object, dose, data)

## S4 method for signature 'CohortSizeConst'
size(object, dose, ...)

## S4 method for signature 'CohortSizeParts'
size(object, dose, data)

## S4 method for signature 'CohortSizeOrdinal'
size(object, dose, data, ...)

## S4 method for signature 'Samples'
size(object, ...)
```

Arguments

object	(McmcOptions or Samples or CohortSize) an object for which the size is computed.
...	not used.
dose	(numeric) the next dose.
data	the data input, an object of class DataOrdinal.

Value

A size of a given object.

Functions

- `size(McmcOptions)`: compute the number of MCMC samples based on `McmcOptions` object.
- `size(CohortSizeRange)`: Determines the size of the next cohort based on the range into which the next dose falls into.
- `size(CohortSizeDLT)`: Determines the size of the next cohort based on the number of DLTs so far.
- `size(CohortSizeMax)`: Determines the size of the next cohort based on maximum of multiple cohort size rules.
- `size(CohortSizeMin)`: Determines the size of the next cohort based on minimum of multiple cohort size rules.
- `size(CohortSizeConst)`: Constant cohort size.
- `size(CohortSizeParts)`: Determines the size of the next cohort based on the parts.
- `size(CohortSizeOrdinal)`: Determines the size of the next cohort in a ordinal CRM trial.
- `size(Samples)`: get the number of MCMC samples from `Samples` object.

Examples

```
# Set up the MCMC option in order to have a burn-in of 10000 iterations and
# then take every other iteration up to a collection of 10000 samples.
my_options <- McmcOptions(burnin = 10000, step = 2, samples = 10000)

size(my_options)
# nolint start

# Create the data
data <- Data(x=c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
             y=c(0, 0, 0, 0, 0, 0, 1, 0),
             cohort=c(0, 1, 2, 3, 4, 5, 5, 5),
             doseGrid=
               c(0.1, 0.5, 1.5, 3, 6,
                 seq(from=10, to=80, by=2)))

# Initialize the CRM model used to model the data
model <- LogisticLogNormal(mean=c(-0.85, 1),
                           cov=
                             matrix(c(1, -0.5, -0.5, 1),
                                    nrow=2),
                           ref_dose=56)

# Set-up some MCMC parameters and generate samples from the posterior
options <- McmcOptions(burnin=100,
                       step=2,
                       samples=2000)
set.seed(94)
```

```
samples <- mcmc(data, model, options)

# Define the rule for dose increments and calculate the maximum dose allowed
myIncrements <- IncrementsRelative(intervals=c(0, 20),
                                   increments=c(1, 0.33))
nextMaxDose <- maxDose(myIncrements,
                       data=data)

# Define the rule which will be used to select the next best dose
# based on the class 'NextBestNCRM'
myNextBest <- NextBestNCRM(target=c(0.2, 0.35),
                           overdose=c(0.35, 1),
                           max_overdose_prob=0.25)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
                               doselimit=nextMaxDose,
                               samples=samples, model=model, data=data)

# Rule for the cohort size:
#   - having cohort of size 1 for doses <10
#   - and having cohort of size 3 for doses >=10
mySize <- CohortSizeRange(intervals = c(0, 10), cohort_size = c(1, 3))

# Determine the cohort size for the next cohort
size(mySize, dose=doseRecommendation$value, data = data)

# nolint end
# nolint start

# Create the data
data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  cohort = c(0, 1, 2, 3, 4, 5, 5, 5),
  doseGrid =
    c(
      0.1, 0.5, 1.5, 3, 6,
      seq(from = 10, to = 80, by = 2)
    )
)

# Initialize the CRM model used to model the data
model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov =
    matrix(c(1, -0.5, -0.5, 1),
      nrow = 2
    ),
  ref_dose = 56
)

# Set-up some MCMC parameters and generate samples from the posterior
```

```
options <- McmcOptions(
  burnin = 100,
  step = 2,
  samples = 2000
)
set.seed(94)
samples <- mcmc(data, model, options)

# Define the rule for dose increments and calculate the maximum dose allowed
myIncrements <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)
nextMaxDose <- maxDose(myIncrements,
  data = data
)

# Define the rule which will be used to select the next best dose
# based on the class 'NextBestNCRM'
myNextBest <- NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
  doselimit = nextMaxDose,
  samples = samples, model = model, data = data
)

# Rule for the cohort size:
#   - having cohort of size 1 if no DLTs were yet observed
#   - and having cohort of size 3 if at least 1 DLT was already observed
mySize <- CohortSizeDLT(
  intervals = c(0, 1),
  cohort_size = c(1, 3)
)

# Determine the cohort size for the next cohort
size(mySize, dose = doseRecommendation$value, data = data)

# nolint end
# nolint start

# Create the data
data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  cohort = c(0, 1, 2, 3, 4, 5, 5, 5),
  doseGrid =
    c(
      0.1, 0.5, 1.5, 3, 6,
```

```
      seq(from = 10, to = 80, by = 2)
    )
)

# Initialize the CRM model used to model the data
model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov =
    matrix(c(1, -0.5, -0.5, 1),
      nrow = 2
    ),
  ref_dose = 56
)

# Set-up some MCMC parameters and generate samples from the posterior
options <- McmcOptions(
  burnin = 100,
  step = 2,
  samples = 2000
)
set.seed(94)
samples <- mcmc(data, model, options)

# Define the rule for dose increments and calculate the maximum dose allowed
myIncrements <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)
nextMaxDose <- maxDose(myIncrements,
  data = data
)

# Define the rule which will be used to select the next best dose
# based on the class 'NextBestNCRM'
myNextBest <- NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
  doselimit = nextMaxDose,
  samples = samples, model = model, data = data
)

# Rule for having cohort of size 1 for doses <30
#      and having cohort of size 3 for doses >=30
mySize1 <- CohortSizeRange(
  intervals = c(0, 10),
  cohort_size = c(1, 3)
)
```

```
# Rule for having cohort of size 1 until no DLT were observed
#      and having cohort of size 3 as soon as 1 DLT is observed
mySize2 <- CohortSizeDLT(
  intervals = c(0, 1),
  cohort_size = c(1, 3)
)

# Combining the two rules for cohort size by taking the maximum of the sample sizes
# of the single rules
mySize <- maxSize(mySize1, mySize2)

# Determine the cohort size for the next cohort
size(mySize, dose = doseRecommendation$value, data = data)

# nolint end
# nolint start

# Create the data
data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  cohort = c(0, 1, 2, 3, 4, 5, 5, 5),
  doseGrid =
    c(
      0.1, 0.5, 1.5, 3, 6,
      seq(from = 10, to = 80, by = 2)
    )
)

# Initialize the CRM model used to model the data
model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov =
    matrix(c(1, -0.5, -0.5, 1),
      nrow = 2
    ),
  ref_dose = 56
)

# Set-up some MCMC parameters and generate samples from the posterior
options <- McmcOptions(
  burnin = 100,
  step = 2,
  samples = 2000
)
set.seed(94)
samples <- mcmc(data, model, options)

# Define the rule for dose increments and calculate the maximum dose allowed
myIncrements <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)
```

```
nextMaxDose <- maxDose(myIncrements,
  data = data
)

# Define the rule which will be used to select the next best dose
# based on the class 'NextBestNCRM'
myNextBest <- NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
  doselimit = nextMaxDose,
  samples = samples, model = model, data = data
)

# Rule for having cohort of size 1 for doses <30
#      and having cohort of size 3 for doses >=30
mySize1 <- CohortSizeRange(
  intervals = c(0, 30),
  cohort_size = c(1, 3)
)

# Rule for having cohort of size 1 until no DLT were observed
#      and having cohort of size 3 as soon as 1 DLT is observed
mySize2 <- CohortSizeDLT(
  intervals = c(0, 1),
  cohort_size = c(1, 3)
)

# Combining the two rules for cohort size by taking the minimum of the sample sizes
# of the single rules
mySize <- minSize(mySize1, mySize2)

# Determine the cohort size for the next cohort
size(mySize, dose = doseRecommendation$value, data = data)

# nolint end
# nolint start

# Create the data
data <- Data(x=c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
             y=c(0, 0, 0, 0, 0, 0, 1, 0),
             cohort=c(0, 1, 2, 3, 4, 5, 5, 5),
             doseGrid=
               c(0.1, 0.5, 1.5, 3, 6,
                 seq(from=10, to=80, by=2)))

# Initialize the CRM model used to model the data
model <- LogisticLogNormal(mean=c(-0.85, 1),
                           cov=
```

```
                              matrix(c(1, -0.5, -0.5, 1),
                                     nrow=2),
                            ref_dose=56)

# Set-up some MCMC parameters and generate samples from the posterior
options <- McmcOptions(burnin=100,
                       step=2,
                       samples=2000)
set.seed(94)
samples <- mcmc(data, model, options)

# Define the rule for dose increments and calculate the maximum dose allowed
myIncrements <- IncrementsRelative(intervals=c(0, 20),
                                   increments=c(1, 0.33))
nextMaxDose <- maxDose(myIncrements,
                       data=data)

# Define the rule which will be used to select the next best dose
# based on the class 'NextBestNCRM'
myNextBest <- NextBestNCRM(target=c(0.2, 0.35),
                           overdose=c(0.35, 1),
                           max_overdose_prob=0.25)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
                               doselimit=nextMaxDose,
                               samples=samples, model=model, data=data)

# Rule for having cohorts with constant cohort size of 3
mySize <- CohortSizeConst(size=3)

# Determine the cohort size for the next cohort
size(mySize, dose=doseRecommendation$value)

# nolint end
# nolint start

# create an object of class 'DataParts'
data <- DataParts(
  x = c(0.1, 0.5, 1.5),
  y = c(0, 0, 0),
  doseGrid = c(
    0.1, 0.5, 1.5, 3, 6,
    seq(from = 10, to = 80, by = 2)
  ),
  part = c(1L, 1L, 1L),
  nextPart = 1L,
  part1Ladder = c(0.1, 0.5, 1.5, 3, 6, 10)
)

# Initialize the CRM model used to model the data
model <- LogisticLogNormal(
  mean = c(-0.85, 1),
```

```
  cov =
    matrix(c(1, -0.5, -0.5, 1),
      nrow = 2
    ),
  ref_dose = 56
)

# Set-up some MCMC parameters and generate samples from the posterior
options <- McmcOptions(
  burnin = 100,
  step = 2,
  samples = 2000
)
set.seed(94)
samples <- mcmc(data, model, options)

myIncrements <- IncrementsRelativeParts(
  dlt_start = 0,
  clean_start = 1
)
nextMaxDose <- maxDose(myIncrements,
  data = data
)

# Define the rule which will be used to select the next best dose
# based on the class 'NextBestNCRM'
myNextBest <- NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
  doselimit = nextMaxDose,
  samples = samples,
  model = model,
  data = data
)

# Rule for the cohort size:
mySize <- CohortSizeParts(cohort_sizes = c(1, 3))

# Determine the cohort size for the next cohort
size(mySize, dose = doseRecommendation$value, data = data)

# nolint end
CohortSizeOrdinal(
  grade = 1L,
  rule = CohortSizeRange(intervals = c(0L, 30L), cohort_size = c(1L, 3L))
)
# Set up the MCMC option in order to have a burn-in of 100 iterations and
# then take every other iteration up to a collection of 200 samples.
```

```
my_options <- McmcOptions(burnin = 100, step = 2, samples = 200)

my_samples <- Samples(
  data = list(alpha = rnorm(200), beta = rnorm(200)),
  options = my_options
)

size(my_samples)
```

Stopping-class Stopping

Description

[Stable]

Stopping is a class for stopping rules.

Slots

report_label (string)
: a label for the stopping report. The meaning of this parameter is twofold. If it is equal to NA_character_ (default), the report_label will not be used in the report at all. Otherwise, if it is specified as an empty character (i.e. character(0)) in a user constructor, then a default, class-specific label will be created for this slot. Finally, for the remaining cases, a user can provide a custom label.

See Also

StoppingList, StoppingCohortsNearDose, StoppingPatientsNearDose, StoppingMinCohorts, StoppingMinPatients, StoppingTargetProb, StoppingMTDdistribution, StoppingTargetBiomarker, StoppingHighestDose StoppingMTDCV, StoppingLowestDoseHSRBeta, StoppingSpecificDose.

StoppingAll-class StoppingAll

Description

[Stable]

StoppingAll is the class for testing a stopping rule that consists of many single stopping rules that are in turn the objects of class Stopping. All single stopping rules must be satisfied in order the result of this rule to be TRUE.

Usage

```
StoppingAll(stop_list, report_label = NA_character_)

.DefaultStoppingAll()
```

Arguments

stop_list	(list) see slot definition.
report_label	(string) see slot definition.

Slots

stop_list (list)
: list of stopping rules.

report_label label for reporting

Note

Typically, end users will not use the .DefaultStoppingAll() function.

Examples

```
# Define some stopping rules.
my_stopping1 <- StoppingMinCohorts(nCohorts = 3)
my_stopping2 <- StoppingTargetProb(target = c(0.2, 0.35), prob = 0.5)
my_stopping3 <- StoppingMinPatients(nPatients = 20)

# Create a list of stopping rules (of class `StoppingAll`) which would then be
# summarized by the `all` function, meaning that the study would be stopped only
# if all of the single stopping rules are `TRUE`.
my_stopping <- StoppingAll(
  stop_list = c(my_stopping1, my_stopping2, my_stopping3)
)
```

StoppingAny-class StoppingAny

Description

[Stable]

StoppingAny is the class for testing a stopping rule that consists of many single stopping rules that are in turn the objects of class Stopping. At least one single stopping rule must be satisfied in order the result of this rule to be TRUE.

Usage

```
StoppingAny(stop_list, report_label = NA_character_)

.DefaultStoppingAny()
```

Arguments

stop_list	(list) see slot definition.
report_label	(string) see slot definition.

Slots

stop_list (list)
: list of stopping rules.

report_label label for reporting

Note

Typically, end users will not use the .DefaultStoppingAny() function.

Examples

```
# Define some stopping rules.
my_stopping1 <- StoppingMinCohorts(nCohorts = 3)
my_stopping2 <- StoppingTargetProb(target = c(0.2, 0.35), prob = 0.5)
my_stopping3 <- StoppingMinPatients(nPatients = 20)

# Create a list of stopping rules (of class `StoppingAny`) which would then be
# summarized by the `any` function, meaning that the study would be stopped if
# any of the single stopping rules is `TRUE`.
my_stopping <- StoppingAny(
  stop_list = c(my_stopping1, my_stopping2, my_stopping3)
)
```

StoppingCohortsNearDose-class
StoppingCohortsNearDose

Description

[Stable]

StoppingCohortsNearDose is the class for stopping based on number of cohorts near to next best dose.

Usage

```
StoppingCohortsNearDose(
  nCohorts = 2L,
  percentage = 50,
  report_label = NA_character_
)

.DefaultStoppingCohortsNearDose()
```

Arguments

nCohorts	(number) see slot definition.
percentage	(number) see slot definition.
report_label	(string or NA) see slot definition.

Slots

nCohorts (number)
: number of required cohorts.

percentage (number)
: percentage (between and including 0 and 100) within the next best dose the cohorts must lie.

Note

Typically, end users will not use the .DefaultStoppingCohortsNearDose() function.

Examples

```
# Here, is the rule for stopping the study if at least 3 cohorts were dosed
# at a dose within (1 +/- 0.2) of the next best dose.
my_stopping <- StoppingCohortsNearDose(
  nCohorts = 3,
  percentage = 0.2
)
```

StoppingExternal-class

StoppingExternal

Description

[Experimental]

StoppingExternal is the class for stopping based on an external flag.

Usage

```
StoppingExternal(report_label = NA_character_)

.DefaultStoppingExternal(report_label = NA_character_)
```

Arguments

report_label	(string or NA) see slot definition.

Note

Typically, end users will not use the `.DefaultStoppingExternal()` function.

Examples

```
my_stopping <- StoppingExternal()
```

StoppingHighestDose-class
StoppingHighestDose

Description

[Experimental]

StoppingHighestDose is the class for stopping based on the highest dose. That is, the stopping occurs when the highest dose is reached.

Usage

```
StoppingHighestDose(report_label = NA_character_)

.DefaultStoppingHighestDose()
```

Arguments

report_label	(string or NA) see slot definition.

Note

Typically, end users will not use the `.DefaultStoppingHighestDose()` function.

Examples

```
# The following stopping rule is met when:
# - next proposed dose is highest dose, and
# - there are already at least 3 patients on that dose, and
# - we are sure that this dose is safe, e.g. the probability to be in (0%, 20%)
# interval of the DLT rate is above 50%.
my_stopping <- StoppingHighestDose() &
  StoppingPatientsNearDose(nPatients = 3, percentage = 0) &
  StoppingTargetProb(target = c(0, 0.2), prob = 0.5)

# We note that this rule would then need to be combined with the other standard
# stopping rules, when the MTD is found based on being near e.g. a 30% DLT
# probability or having reached maximal sample size, in the manner of:
# stop_rule <- stop_high | stop_low | stop_sample_size # nolintr.
```

StoppingList-class StoppingList

Description

[Stable]

StoppingList is the class for testing a stopping rule that consists of many single stopping rules that are in turn the objects of class Stopping. The summary slot stores a function that takes a logical vector of the size of stop_list and returns a single logical value. For example, if the function all is specified as a summary function, then that all stopping rules defined in stop_list must be satisfied in order the result of this rule to be TRUE.

Usage

```
StoppingList(stop_list, summary)

.DefaultStoppingList()
```

Arguments

stop_list	(list) see slot definition.
summary	(function) see slot definition.

Slots

stop_list (list)
: list of stopping rules.

summary (function)
: a summary function to combine the results of the stopping rules into a single result.

Note

Typically, end users will not use the .DefaultStoppingList() function.

Examples

```
# Define some stopping rules.
my_stopping1 <- StoppingMinCohorts(nCohorts = 3)
my_stopping2 <- StoppingTargetProb(target = c(0.2, 0.35), prob = 0.5)
my_stopping3 <- StoppingMinPatients(nPatients = 20)

# Create a list of stopping rules (of class `StoppingList`) which will then be
# summarized (in this specific example) with the `any` function, meaning that
# the study would be stopped if any of the single stopping rules is `TRUE`.
my_stopping <- StoppingList(
```

```
  stop_list = c(my_stopping1, my_stopping2, my_stopping3),
  summary = any
)
```

`StoppingLowestDoseHSRBeta-class` `StoppingLowestDoseHSRBeta`

Description

[Experimental]

`StoppingLowestDoseHSRBeta` is a class for stopping based on a Hard Safety Rule using the Beta posterior distribution with Beta(a,b) prior and a Bin-Beta model based on the observed data at the lowest dose level. The rule is triggered when the first dose is considered to be toxic (i.e. above threshold probability) based on the observed data at the lowest dose level and a Beta(a,b) prior distribution. The default prior is Beta(1,1). In case that placebo is used, the rule is evaluated at the second dose of the dose grid, i.e. at the lowest non-placebo dose.

Usage

```
StoppingLowestDoseHSRBeta(
  target = 0.3,
  prob = 0.95,
  a = 1,
  b = 1,
  report_label = NA_character_
)

.DefaultStoppingLowestDoseHSRBeta()
```

Arguments

`target`	(`proportion`) see slot definition.
`prob`	(`proportion`) see slot definition.
`a`	(`number`) see slot definition.
`b`	(`number`) see slot definition.
`report_label`	(`string` or `NA`) see slot definition.

Slots

`target` (`proportion`)
: the target toxicity.

`prob` (`proportion`)
: the threshold probability for the lowest dose being toxic.

`a` (`number`)
: shape parameter $a > 0$ of probability distribution Beta (a,b).

`b` (`number`)
: shape parameter $b > 0$ of probability distribution Beta (a,b).

Note

This stopping rule is independent from the underlying model.

Typically, end users will not use the `.DefaultStoppingLowestDoseHSRBeta()` function.

Examples

```
# Stopping the study if the first dose is toxic with more than 90%
# probability based on a Beta posterior distribution with Beta(1,1) prior.
my_stopping <- StoppingLowestDoseHSRBeta(
  target = 0.3,
  prob = 0.9
)

# Stopping the study if the first dose is toxic with more than 90%
# probability based on a Beta posterior distribution with Beta(0.5,0.5) prior.
my_stopping <- StoppingLowestDoseHSRBeta(
  target = 0.3,
  prob = 0.9,
  a = 0.5,
  b = 0.5
)
```

StoppingMaxGainCIRatio-class
StoppingMaxGainCIRatio

Description

[Stable]

`StoppingMaxGainCIRatio` is the class for testing a stopping rule that is based on a target ratio of the 95% credibility interval. Specifically, this is the ratio of the upper to the lower bound of the 95% credibility interval's estimate of the: (1) target dose (i.e. a dose that corresponds to a given target probability of the occurrence of a DLT `prob_target`), or (2) max gain dose (i.e. a dose which gives the maximum gain), depending on which one out of these two is smaller.

Usage

```
StoppingMaxGainCIRatio(
  target_ratio = 5,
  prob_target = 0.3,
  report_label = NA_character_
)

.DefaultStoppingMaxGainCIRatio()
```

Arguments

target_ratio	(numeric) see slot definition.
prob_target	(proportion) see slot definition.
report_label	(string or NA) see slot definition.

Slots

`target_ratio` (numeric)
: target for the ratio of the 95% credibility interval's estimate, that is required to stop a trial.

`prob_target` (proportion)
: the target probability of the occurrence of a DLT.

Examples

```
# Define the target stopping ratio (5) and the target probability of a DLT to
# be used (0.3).
my_stopping <- StoppingMaxGainCIRatio(target_ratio = 5, prob_target = 0.3)
.DefaultStoppingMaxGainCIRatio()
```

StoppingMinCohorts-class

StoppingMinCohorts

Description

[Stable]

StoppingMinCohorts is the class for stopping based on minimum number of cohorts.

Usage

```
StoppingMinCohorts(nCohorts = 2L, report_label = NA_character_)

.DefaultStoppingMinCohorts()
```

Arguments

nCohorts	(number) see slot definition.
report_label	(string or NA) see slot definition.

Slots

nCohorts (number)
: minimum required number of cohorts.

Note

Typically, end users will not use the .DefaultStoppingMinCohorts() function.

Examples

```
# As example, here is the rule for stopping the study if at least 6 cohorts
# were already dosed.
my_stopping <- StoppingMinCohorts(nCohorts = 6)
```

StoppingMinPatients-class
StoppingMinPatients

Description

[Stable]

StoppingMinPatients is the class for stopping based on minimum number of patients

Usage

```
StoppingMinPatients(nPatients = 20L, report_label = NA_character_)

.DefaultStoppingMinPatients()
```

Arguments

nPatients	(number) see slot definition.
report_label	(string or NA) see slot definition.

Slots

nPatients (number)
: minimum allowed number of patients.

Note

Typically, end users will not use the `.DefaultStoppingMinPatients()` function.

Examples

```
# As example, here is the rule for stopping the study if at least 20 patients
# were already dosed
my_stopping <- StoppingMinPatients(nPatients = 20)
```

StoppingMissingDose-class
StoppingMissingDose

Description

[Experimental]

`StoppingMissingDose` is the class for stopping based on NA returned by next best dose.

Usage

```
StoppingMissingDose(report_label = NA_character_)

.DefaultStoppingMissingDose()
```

Arguments

`report_label`	(`string` or `NA`) see slot definition.

Note

Typically, end users will not use the `.DefaultStoppingMissingDose()` function.

Examples

```
# The rule for stopping the study if NA or Placebo is returned as
# next best dose.
my_stopping <- StoppingMissingDose()
```

`StoppingMTDCV-class` `StoppingMTDCV`

Description

[Experimental]

StoppingMTDCV is a class for stopping rule based on precision of MTD which is calculated as the coefficient of variation (CV) of the MTD. Here, the MTD is defined as the dose that reaches a specific `target` probability of the occurrence of a DLT.

Usage

```
StoppingMTDCV(target = 0.3, thresh_cv = 40, report_label = NA_character_)

.DefaultStoppingMTDCV()
```

Arguments

`target`	(`proportion`) see slot definition.
`thresh_cv`	(`number`) see slot definition.
`report_label`	(`string` or `NA`) see slot definition.

Slots

`target` (`proportion`)
: toxicity target of MTD (except 0 or 1).

`thresh_cv` (`number`)
: threshold (percentage > 0) for CV to be considered accurate enough to stop the trial. The stopping occurs when the CV is less than or equal to `tresh_cv`.

Note

Typically, end users will not use the `.DefaultStoppingMTDCV()` function.

Examples

```
# Stopping the study if the MTD estimation is precise enough, i.e. if robust
# coefficient of variation of the MTD is below 40%.
my_stopping <- StoppingMTDCV(target = 0.3, thresh_cv = 40)
```

StoppingMTDdistribution-class
StoppingMTDdistribution

Description

[Stable]

StoppingMTDdistribution is the class for stopping based on the posterior distribution of the MTD. It is used for the cases where the stopping occurs when the probability of MTD > thresh * next_dose is greater than or equal to prob, where the next_dose is the recommended next best dose. Here, the MTD is defined as the dose that reaches a specific target probability of the occurrence of a DLT.

Usage

```
StoppingMTDdistribution(
  target = 0.33,
  thresh = 0.5,
  prob = 0.9,
  report_label = NA_character_
)

.DefaultStoppingMTDdistribution()
```

Arguments

target	(proportion) see slot definition.
thresh	(proportion) see slot definition.
prob	(proportion) see slot definition.
report_label	(string or NA) see slot definition.

Slots

target (proportion)
: the target toxicity probability (except 0 or 1) defining the MTD.

thresh (proportion)
: the threshold (except 0 or 1) relative to the recommended next best dose.

prob (proportion)
: required minimum probability, except 0 or 1.

Note

Typically, end users will not use the `.DefaultStoppingMTDDistribution()` function.

Examples

```
# As example, here is the rule for stopping the study if there is at least 0.9
# probability that MTD > 0.5 * next_dose. Here MTD is defined as the dose for
# which prob(DLT) = 0.33
my_stopping <- StoppingMTDdistribution(
  target = 0.33,
  thresh = 0.5,
  prob = 0.9
)
```

`StoppingOrdinal-class  StoppingOrdinal`

Description

[Experimental]

`StoppingOrdinal` is the class for stopping based on a Stopping rule applied to a specific toxicity grade in an ordinal CRM trial

Usage

```
StoppingOrdinal(grade, rule)

.DefaultStoppingOrdinal()
```

Arguments

`grade`	`(integer)` see slot definition.
`rule`	`(Stopping)` see slot definition.

Slots

`grade (integer)`
: the grade to which the rule should be applied

`rule (Stopping)`
: the rule to apply

Note

Typically, end users will not use the `.DefaultStoppingOrdinal()` function.

Examples

```
StoppingOrdinal(
  1L,
  StoppingTargetProb(target = c(0.2, 0.35), prob = 0.6)
)
```

StoppingPatientsNearDose-class
StoppingPatientsNearDose

Description

[Stable]

StoppingPatientsNearDose is the class for stopping based on number of patients near to next best dose.

Usage

```
StoppingPatientsNearDose(
  nPatients = 10L,
  percentage = 50,
  report_label = NA_character_
)

.DefaultStoppingPatientsNearDose()
```

Arguments

nPatients	(number) see slot definition.
percentage	(number) see slot definition.
report_label	(string or NA) see slot definition.

Slots

`nPatients` (number)
: number of required patients.

`percentage` (number)
: percentage (between and including 0 and 100) within the next best dose the patients must lie.

Note

Typically, end users will not use the `.DefaultStoppingPatientsNearDose()` function.

Examples

```
# As example, here is the rule for stopping the study if at least 9 patients
# were dosed at a dose within (1 +/- 0.2) of the next best dose.

my_stopping <- StoppingPatientsNearDose(
  nPatients = 9,
  percentage = 20
)
```

StoppingSpecificDose-class
StoppingSpecificDose

Description

[Experimental]

StoppingSpecificDose is the class for testing a stopping rule at specific dose of the dose grid and not at the next best dose.

Usage

```
StoppingSpecificDose(
  rule = StoppingTargetProb(target = c(0, 0.3), prob = 0.8),
  dose = 80,
  report_label = NA_character_
)

.DefaultStoppingSpecificDose()
```

Arguments

rule	(Stopping) see slot definition.
dose	(number) see slot definition.
report_label	(string or NA) see slot definition.

Slots

`rule` (Stopping)
: a stopping rule available in this package.

`dose` (positive_number)
: a dose that is defined as part of the dose grid of the data.

Note

Typically, end users will not use the `.DefaultStoppingSpecificDose()` function.

Examples

```
# Stop if highest dose 80 is safe.
highest_dose_safe <- StoppingSpecificDose(
  rule = StoppingTargetProb(target = c(0, 0.3), prob = 0.8),
  dose = 80
)

# Stop if lowest dose 10 is toxic.
lowest_dose_toxic <- StoppingSpecificDose(
  rule = StoppingTargetProb(target = c(0.3, 1), prob = 0.8),
  dose = 10
)
```

StoppingTargetBiomarker-class
StoppingTargetBiomarker

Description

[Stable]

`StoppingTargetBiomarker` is a class for stopping based on probability of target biomarker.

Usage

```
StoppingTargetBiomarker(
  target = c(0.9, 1),
  prob = 0.3,
  is_relative = TRUE,
  report_label = NA_character_
)

.DefaultStoppingTargetBiomarker()
```

Arguments

`target`	(`numeric`) see slot definition.
`prob`	(`proportion`) see slot definition.
`is_relative`	(`flag`) see slot definition.
`report_label`	(`string` or `NA`) see slot definition.

Slots

`target` (`numeric`)
: the biomarker target range that needs to be reached. For example, `target = c(0.8, 1.0)` with `is_relative = TRUE` means that we target a dose with at least 80% of maximum biomarker level.

`is_relative` (`flag`)
: is target relative? If it so (default), then the `target` is interpreted relative to the maximum, so it must be a probability range. Otherwise, the `target` is interpreted as absolute biomarker range.

`prob` (`proportion`)
: required target probability (except 0 or 1) for reaching sufficient precision.

Note

Typically, end users will not use the `.DefaultStoppingTargetBiomarker()` function.

Examples

```
# Stopping the study if there is at least 0.5 probability that the biomarker
# (efficacy) is within the biomarker target range of [0.9, 1.0] (relative to the
# maximum for the biomarker).

my_stopping <- StoppingTargetBiomarker(target = c(0.9, 1), prob = 0.5)
```

StoppingTargetProb-class

StoppingTargetProb

Description

[Stable]

`StoppingTargetProb` is the class for stopping based on the probability of the DLT rate being in the target toxicity interval.

Usage

```
StoppingTargetProb(
  target = c(0.2, 0.35),
  prob = 0.4,
  report_label = NA_character_
)

.DefaultStoppingTargetProb()
```

Arguments

target	(number) see slot definition.
prob	(proportion) see slot definition.
report_label	(string or NA) see slot definition.

Slots

target (number)
: the target toxicity interval, e.g. c(0.2, 0.35).

prob (proportion)
: required target toxicity probability (except 0 or 1) for reaching sufficient precision.

Note

Typically, end users will not use the .DefaultStoppingTargetProb() function.

Examples

```
# As example, here is the rule for stopping the study if the posterior
# probability that [0.2 =< Prob(DLT | dose) <= 0.35] for the next best dose
# is above 0.5.
my_stopping <- StoppingTargetProb(target = c(0.2, 0.35), prob = 0.5)
```

StoppingTDCIRatio-class
StoppingTDCIRatio

Description

[Stable]

StoppingTDCIRatio is the class for testing a stopping rule that is based on a target ratio of the 95% credibility interval. Specifically, this is the ratio of the upper to the lower bound of the 95% credibility interval's estimate of the target dose (i.e. a dose that corresponds to a given target probability of the occurrence of a DLT prob_target).

Usage

```
StoppingTDCIRatio(
  target_ratio = 5,
  prob_target = 0.3,
  report_label = NA_character_
)

.DefaultStoppingTDCIRatio()
```

Arguments

target_ratio	(numeric) see slot definition.
prob_target	(proportion) see slot definition.
report_label	(string or NA) see slot definition.

Slots

target_ratio (numeric)
: target for the ratio of the 95% credibility interval's estimate, that is required to stop a trial.

prob_target (proportion)
: the target probability of the occurrence of a DLT.

Note

Typically, end users will not use the .DefaultStoppingTDCIRatio() function.

Examples

```
# Define the target stopping ratio (5) and the target probability of a DLT to
# be used (0.3).
my_stopping <- StoppingTDCIRatio(
  target_ratio = 5,
  prob_target = 0.3
)
```

---

stopTrial *Stop the trial?*

Description

[Stable]

This function returns whether to stop the trial.

[Experimental]

Stopping rule based precision of the MTD estimation. The trial is stopped, when the MTD can be estimated with sufficient precision. The criteria is based on the robust coefficient of variation (CV) calculated from the posterior distribution. The robust CV is defined mad(MTD) / median(MTD), where mad is the median absolute deviation.

Stopping based based on the lowest non placebo dose. The trial is stopped when the lowest non placebo dose meets the Hard Safety Rule, i.e. it is deemed to be overly toxic. Stopping is based on the observed data at the lowest dose level using a Bin-Beta model based on DLT probability.

[Experimental]

[Experimental]

[Experimental]

Usage

```
stopTrial(stopping, dose, samples, model, data, ...)

## S4 method for signature 'StoppingMissingDose,numeric,ANY,ANY,Data'
stopTrial(stopping, dose, samples, model, data, ...)

## S4 method for signature 'StoppingList,ANY,ANY,ANY,ANY'
stopTrial(stopping, dose, samples, model, data, ...)

## S4 method for signature 'StoppingAll,ANY,ANY,ANY,ANY'
stopTrial(stopping, dose, samples, model, data, ...)

## S4 method for signature 'StoppingAny,ANY,ANY,ANY,ANY'
stopTrial(stopping, dose, samples, model, data, ...)

## S4 method for signature 'StoppingCohortsNearDose,numeric,ANY,ANY,Data'
stopTrial(stopping, dose, samples, model, data, ...)

## S4 method for signature 'StoppingPatientsNearDose,numeric,ANY,ANY,Data'
stopTrial(stopping, dose, samples, model, data, ...)

## S4 method for signature 'StoppingMinCohorts,ANY,ANY,ANY,Data'
stopTrial(stopping, dose, samples, model, data, ...)

## S4 method for signature 'StoppingMinPatients,ANY,ANY,ANY,Data'
stopTrial(stopping, dose, samples, model, data, ...)

## S4 method for signature
## 'StoppingTargetProb,numeric,Samples,GeneralModel,ANY'
stopTrial(stopping, dose, samples, model, data, ...)

## S4 method for signature
## 'StoppingMTDdistribution,numeric,Samples,GeneralModel,ANY'
stopTrial(stopping, dose, samples, model, data, ...)

## S4 method for signature 'StoppingMTDCV,numeric,Samples,GeneralModel,ANY'
stopTrial(stopping, dose, samples, model, data, ...)

## S4 method for signature 'StoppingLowestDoseHSRBeta,numeric,Samples,ANY,ANY'
stopTrial(stopping, dose, samples, model, data, ...)

## S4 method for signature
## 'StoppingTargetBiomarker,numeric,Samples,DualEndpoint,ANY'
stopTrial(stopping, dose, samples, model, data, ...)

## S4 method for signature 'StoppingSpecificDose,numeric,ANY,ANY,Data'
stopTrial(stopping, dose, samples, model, data, ...)
```

```
## S4 method for signature 'StoppingHighestDose,numeric,ANY,ANY,Data'
stopTrial(stopping, dose, samples, model, data, ...)

## S4 method for signature
## 'StoppingOrdinal,numeric,ANY,LogisticLogNormalOrdinal,DataOrdinal'
stopTrial(stopping, dose, samples, model, data, ...)

## S4 method for signature 'StoppingOrdinal,numeric,ANY,ANY,ANY'
stopTrial(stopping, dose, samples, model, data, ...)

## S4 method for signature 'StoppingExternal,numeric,ANY,ANY,ANY'
stopTrial(stopping, dose, samples, model, data, external, ...)

## S4 method for signature 'StoppingTDCIRatio,ANY,Samples,ModelTox,ANY'
stopTrial(stopping, dose, samples, model, data, ...)

## S4 method for signature 'StoppingTDCIRatio,ANY,missing,ModelTox,ANY'
stopTrial(stopping, dose, samples, model, data, ...)

## S4 method for signature
## 'StoppingMaxGainCIRatio,ANY,Samples,ModelTox,DataDual'
stopTrial(
  stopping,
  dose,
  samples,
  model,
  data,
  TDderive,
  Effmodel,
  Effsamples,
  Gstarderive,
  ...
)

## S4 method for signature
## 'StoppingMaxGainCIRatio,ANY,missing,ModelTox,DataDual'
stopTrial(stopping, dose, model, data, Effmodel, ...)
```

Arguments

stopping	(Stopping) the rule for stopping the trial.
dose	the recommended next best dose.
samples	(Samples) the mcmc samples.
model	(GeneralModel) the model.

data	(Data) input data.
...	additional arguments without method dispatch.
external	(flag) whether to stop based on the external result or not.
TDderive	the function which derives from the input, a vector of the posterior samples called TDsamples of the dose which has the probability of the occurrence of DLE equals to either the targetDuringTrial or targetEndOfTrial, the final next best TDtargetDuringTrial (the dose with probability of the occurrence of DLE equals to the targetDuringTrial)and TDtargetEndOfTrial estimate.
Effmodel	the efficacy model of ModelEff class object
Effsamples	the efficacy samples of Samples class object
Gstarderive	the function which derives from the input, a vector of the posterior Gstar (the dose which gives the maximum gain value) samples called Gstarsamples, the final next best Gstar estimate.

Value

logical value: TRUE if the trial can be stopped, FALSE otherwise. It should have an attribute message which gives the reason for the decision.

Functions

- stopTrial(stopping = StoppingMissingDose, dose = numeric, samples = ANY, model = ANY, data = Data): Stop based on value returned by next best dose.
- stopTrial(stopping = StoppingList, dose = ANY, samples = ANY, model = ANY, data = ANY): Stop based on multiple stopping rules
- stopTrial(stopping = StoppingAll, dose = ANY, samples = ANY, model = ANY, data = ANY): Stop based on fulfillment of all multiple stopping rules
- stopTrial(stopping = StoppingAny, dose = ANY, samples = ANY, model = ANY, data = ANY): Stop based on fulfillment of any stopping rule
- stopTrial(stopping = StoppingCohortsNearDose, dose = numeric, samples = ANY, model = ANY, data = Data): Stop based on number of cohorts near to next best dose
- stopTrial(stopping = StoppingPatientsNearDose, dose = numeric, samples = ANY, model = ANY, data = Data): Stop based on number of patients near to next best dose
- stopTrial(stopping = StoppingMinCohorts, dose = ANY, samples = ANY, model = ANY, data = Data): Stop based on minimum number of cohorts
- stopTrial(stopping = StoppingMinPatients, dose = ANY, samples = ANY, model = ANY, data = Data): Stop based on minimum number of patients
- stopTrial(stopping = StoppingTargetProb, dose = numeric, samples = Samples, model = GeneralModel, data = ANY): Stop based on probability of target tox interval
- stopTrial(stopping = StoppingMTDdistribution, dose = numeric, samples = Samples, model = GeneralModel, data = ANY): Stop based on MTD distribution

- stopTrial(stopping = StoppingTargetBiomarker, dose = numeric, samples = Samples, model = DualEndpoint, data = ANY): Stop based on probability of targeting biomarker
- stopTrial(stopping = StoppingSpecificDose, dose = numeric, samples = ANY, model = ANY, data = Data): if Stopping rule is met for specific dose of the planned dose grid and not just for the default next best dose.
- stopTrial(stopping = StoppingHighestDose, dose = numeric, samples = ANY, model = ANY, data = Data): Stop when the highest dose is reached
- stopTrial(stopping = StoppingOrdinal, dose = numeric, samples = ANY, model = LogisticLogNormalOrdina data = DataOrdinal): Stop based on value returned by next best dose.
- stopTrial(stopping = StoppingOrdinal, dose = numeric, samples = ANY, model = ANY, data = ANY): Stop based on value returned by next best dose.
- stopTrial(stopping = StoppingExternal, dose = numeric, samples = ANY, model = ANY, data = ANY): Stop based on an external flag.
- stopTrial(stopping = StoppingTDCIRatio, dose = ANY, samples = Samples, model = ModelTox, data = ANY): Stop based on 'StoppingTDCIRatio' class when reaching the target ratio of the upper to the lower 95% credibility interval of the estimate (TDtargetEndOfTrial). This is a stopping rule which incorporate only DLE responses and DLE samples are given
- stopTrial(stopping = StoppingTDCIRatio, dose = ANY, samples = missing, model = ModelTox, data = ANY): Stop based on 'StoppingTDCIRatio' class when reaching the target ratio of the upper to the lower 95% credibility interval of the estimate (TDtargetEndOfTrial). This is a stopping rule which incorporate only DLE responses and no DLE samples are involved
- stopTrial(stopping = StoppingMaxGainCIRatio, dose = ANY, samples = Samples, model = ModelTox, data = DataDual): Stop based on reaching the target ratio of the upper to the lower 95% credibility interval of the estimate (the minimum of Gstar and TDtargetEndOf-Trial). This is a stopping rule which incorporate DLE and efficacy responses and DLE and efficacy samples are also used.
- stopTrial(stopping = StoppingMaxGainCIRatio, dose = ANY, samples = missing, model = ModelTox, data = DataDual): Stop based on reaching the target ratio of the upper to the lower 95% credibility interval of the estimate (the minimum of Gstar and TDtargetEndOf-Trial). This is a stopping rule which incorporate DLE and efficacy responses without DLE and efficacy samples involved.

Examples

```
## Example of combining stopping rules with '&' and/or '|' operators

myStopping1 <- StoppingMinCohorts(nCohorts=3)
myStopping2 <- StoppingTargetProb(target=c(0.2, 0.35),
                                  prob=0.5)
myStopping3 <- StoppingMinPatients(nPatients=20)

myStopping <- (myStopping1 & myStopping2) | myStopping3




# Example of usage for `StoppingMissingDose` StopTrial class.
```

```
# Create the data.
my_data <- Data(
  x = c(0.01, 0.1, 0.5, 3, 6, 10, 10, 10),
  y = c(0, 1, 1, 0, 0, 0, 0, 1),
  cohort = c(1, 1, 2, 3, 4, 5, 5, 5),
  ID = 1:8,
  doseGrid = c(
    0.01, 0.1, 0.5, 1.5, 3, 6,
    seq(from = 10, to = 80, by = 2)
  ),
  placebo = TRUE
)

# Initialize the CRM model used to model the data.
my_model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov =
    matrix(c(1, -0.5, -0.5, 1),
      nrow = 2
    ),
  ref_dose = 56
)

# Set-up some MCMC parameters and generate samples from the posterior.
my_options <- McmcOptions(
  burnin = 100,
  step = 2,
  samples = 2000
)

my_samples <- mcmc(my_data, my_model, my_options)

# Define the rule for dose increments and calculate the maximum dose allowed.
my_increments <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

next_max_dose <- maxDose(my_increments,
  data = my_data
)

# Define the rule which will be used to select the next best dose based
# on the class 'NextBestNCRM'.
my_next_best <- NextBestNCRM(
  target = c(0.1, 0.25),
  overdose = c(0.2, 1),
  max_overdose_prob = 0.25
)

# Calculate the next best dose.
dose_recommendation <- nextBest(
```

```
  my_next_best,
  doselimit = next_max_dose,
  samples = my_samples, model = my_model, data = my_data
)

# Define the stopping rule such that the study would be stopped if there is
# no safe active dose returned from dose_recommendation.
my_stopping <- StoppingMissingDose()
my_stopping <- StoppingAny(
  stop_list = c(
    StoppingMinPatients(nPatients = 16),
    StoppingMissingDose()
  )
)

# Evaluate if to stop the trial.
stopTrial(
  stopping = my_stopping,
  dose = dose_recommendation$value,
  data = my_data,
  model = my_model
)
# nolint start

# Create some data
data <- Data(x=c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
             y=c(0, 0, 0, 0, 0, 0, 1, 0),
             cohort=c(0, 1, 2, 3, 4, 5, 5, 5),
             doseGrid=
               c(0.1, 0.5, 1.5, 3, 6,
                 seq(from=10, to=80, by=2)))

# Initialize the CRM model used to model the data
model <- LogisticLogNormal(mean=c(-0.85, 1),
                           cov=
                             matrix(c(1, -0.5, -0.5, 1),
                                    nrow=2),
                           ref_dose=56)

# Set-up some MCMC parameters and generate samples from the posterior
options <- McmcOptions(burnin=100,
                       step=2,
                       samples=2000)
set.seed(94)
samples <- mcmc(data, model, options)

# Define the rule for dose increments and calculate the maximum dose allowed
myIncrements <- IncrementsRelative(intervals=c(0, 20),
                                   increments=c(1, 0.33))
nextMaxDose <- maxDose(myIncrements,
                       data=data)

# Define the rule which will be used to select the next best dose
```

```
# based on the class 'NextBestNCRM'
myNextBest <- NextBestNCRM(target=c(0.2, 0.35),
                           overdose=c(0.35, 1),
                           max_overdose_prob=0.25)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
                               doselimit=nextMaxDose,
                               samples=samples, model=model, data=data)

# Define the stopping rules
myStopping1 <- StoppingMinCohorts(nCohorts=3)
myStopping2 <- StoppingTargetProb(target=c(0.2, 0.35),
                                  prob=0.5)
myStopping3 <- StoppingMinPatients(nPatients=20)

# Create a list of stopping rules (of class 'StoppingList') which will then be
# summarized (in this specific example) with the 'any' function, meaning that the study
# would be stopped if 'any' of the single stopping rules is TRUE.
mystopping <- StoppingList(stop_list=c(myStopping1,myStopping2,myStopping3),
                           summary=any)

# Evaluate if to stop the Trial
stopTrial(stopping=myStopping, dose=doseRecommendation$value,
          samples=samples, model=model, data=data)

# nolint end
# nolint start

# Create some data
data <- Data(x=c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
             y=c(0, 0, 0, 0, 0, 0, 1, 0),
             cohort=c(0, 1, 2, 3, 4, 5, 5, 5),
             doseGrid=
               c(0.1, 0.5, 1.5, 3, 6,
                 seq(from=10, to=80, by=2)))

# Initialize the CRM model used to model the data
model <- LogisticLogNormal(mean=c(-0.85, 1),
                           cov=
                             matrix(c(1, -0.5, -0.5, 1),
                                    nrow=2),
                           ref_dose=56)

# Set-up some MCMC parameters and generate samples from the posterior
options <- McmcOptions(burnin=100,
                       step=2,
                       samples=2000)
set.seed(94)
samples <- mcmc(data, model, options)

# Define the rule for dose increments and calculate the maximum dose allowed
myIncrements <- IncrementsRelative(intervals=c(0, 20),
```

```
                                    increments=c(1, 0.33))
nextMaxDose <- maxDose(myIncrements,
                       data=data)

# Define the rule which will be used to select the next best dose
# based on the class 'NextBestNCRM'
myNextBest <- NextBestNCRM(target=c(0.2, 0.35),
                           overdose=c(0.35, 1),
                           max_overdose_prob=0.25)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
                               doselimit=nextMaxDose,
                               samples=samples, model=model, data=data)

# Define the stopping rules
myStopping1 <- StoppingMinCohorts(nCohorts=3)
myStopping2 <- StoppingTargetProb(target=c(0.2, 0.35),
                                  prob=0.5)
myStopping3 <- StoppingMinPatients(nPatients=20)

# Combine the stopping rules, obtaining (in this specific example) a list of stopping
# rules of class 'StoppingAll'
myStopping <- (myStopping1 | myStopping2) & myStopping3

# Evaluate if to stop the Trial
stopTrial(stopping=myStopping, dose=doseRecommendation$value,
          samples=samples, model=model, data=data)

# nolint end
# nolint start

# Create some data
data <- Data(x=c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
             y=c(0, 0, 0, 0, 0, 0, 1, 0),
             cohort=c(0, 1, 2, 3, 4, 5, 5, 5),
             doseGrid=
               c(0.1, 0.5, 1.5, 3, 6,
                 seq(from=10, to=80, by=2)))

# Initialize the CRM model used to model the data
model <- LogisticLogNormal(mean=c(-0.85, 1),
                           cov=
                             matrix(c(1, -0.5, -0.5, 1),
                                    nrow=2),
                           ref_dose=56)

# Set-up some MCMC parameters and generate samples from the posterior
options <- McmcOptions(burnin=100,
                       step=2,
                       samples=2000)
set.seed(94)
samples <- mcmc(data, model, options)
```

```
# Define the rule for dose increments and calculate the maximum dose allowed
myIncrements <- IncrementsRelative(intervals=c(0, 20),
                                   increments=c(1, 0.33))
nextMaxDose <- maxDose(myIncrements,
                       data=data)

# Define the rule which will be used to select the next best dose
# based on the class 'NextBestNCRM'
myNextBest <- NextBestNCRM(target=c(0.2, 0.35),
                           overdose=c(0.35, 1),
                           max_overdose_prob=0.25)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
                               doselimit=nextMaxDose,
                               samples=samples, model=model, data=data)

# Define the stopping rules
myStopping1 <- StoppingMinCohorts(nCohorts=3)
myStopping2 <- StoppingTargetProb(target=c(0.2, 0.35),
                                  prob=0.5)
myStopping3 <- StoppingMinPatients(nPatients=20)

# Combine the stopping rules, obtaining (in this specific example) a list of stopping
# rules of class 'StoppingAny'
myStopping <- (myStopping1 | myStopping2) | myStopping3

# Evaluate if to stop the Trial
stopTrial(stopping=myStopping, dose=doseRecommendation$value,
          samples=samples, model=model, data=data)

# nolint end
# nolint start

# Create the data
data <- Data(x=c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
             y=c(0, 0, 0, 0, 0, 0, 1, 0),
             cohort=c(0, 1, 2, 3, 4, 5, 5, 5),
             doseGrid=
               c(0.1, 0.5, 1.5, 3, 6,
                 seq(from=10, to=80, by=2)))

# Initialize the CRM model used to model the data
model <- LogisticLogNormal(mean=c(-0.85, 1),
                           cov=
                             matrix(c(1, -0.5, -0.5, 1),
                                    nrow=2),
                           ref_dose=56)

# Set-up some MCMC parameters and generate samples from the posterior
options <- McmcOptions(burnin=100,
                       step=2,
```

```
                       samples=2000)
set.seed(94)
samples <- mcmc(data, model, options)

# Define the rule for dose increments and calculate the maximum dose allowed
myIncrements <- IncrementsRelative(intervals=c(0, 20),
                                   increments=c(1, 0.33))
nextMaxDose <- maxDose(myIncrements,
                       data=data)

# Define the rule which will be used to select the next best dose
# based on the class 'NextBestNCRM'
myNextBest <- NextBestNCRM(target=c(0.2, 0.35),
                           overdose=c(0.35, 1),
                           max_overdose_prob=0.25)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
                               doselimit=nextMaxDose,
                               samples=samples, model=model, data=data)

# Define the stopping rule such that the study would be stopped if at least 3
# cohorts were already dosed within 1 +/- 0.2 of the next best dose
myStopping <- StoppingCohortsNearDose(nCohorts = 3,
                                      percentage = 0.2)

# Evaluate if to stop the trial
stopTrial(stopping=myStopping,
          dose=doseRecommendation$value,
          data=data)

# nolint end
# nolint start

# Create the data
data <- Data(x=c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
             y=c(0, 0, 0, 0, 0, 0, 1, 0),
             cohort=c(0, 1, 2, 3, 4, 5, 5, 5),
             doseGrid=
               c(0.1, 0.5, 1.5, 3, 6,
                 seq(from=10, to=80, by=2)))

# Initialize the CRM model used to model the data
model <- LogisticLogNormal(mean=c(-0.85, 1),
                           cov=
                             matrix(c(1, -0.5, -0.5, 1),
                                    nrow=2),
                           ref_dose=56)

# Set-up some MCMC parameters and generate samples from the posterior
options <- McmcOptions(burnin=100,
                       step=2,
                       samples=2000)
```

```
set.seed(94)
samples <- mcmc(data, model, options)

# Define the rule for dose increments and calculate the maximum dose allowed
myIncrements <- IncrementsRelative(intervals=c(0, 20),
                                   increments=c(1, 0.33))
nextMaxDose <- maxDose(myIncrements,
                       data=data)

# Define the rule which will be used to select the next best dose
# based on the class 'NextBestNCRM'
myNextBest <- NextBestNCRM(target=c(0.2, 0.35),
                           overdose=c(0.35, 1),
                           max_overdose_prob=0.25)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
                               doselimit=nextMaxDose,
                               samples=samples, model=model, data=data)

# Define the stopping rule such that the study would be stopped if at least 9
# patients were already dosed within 1 +/- 0.2 of the next best dose
myStopping <- StoppingPatientsNearDose(nPatients = 9,
                                       percentage = 0.2)

# Evaluate if to stop the trial
stopTrial(stopping=myStopping,
          dose=doseRecommendation$value,
          data=data)

# nolint end
# nolint start

# Create the data
data <- Data(x=c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
             y=c(0, 0, 0, 0, 0, 0, 1, 0),
             cohort=c(0, 1, 2, 3, 4, 5, 5, 5),
             doseGrid=
               c(0.1, 0.5, 1.5, 3, 6,
                 seq(from=10, to=80, by=2)))

# Initialize the CRM model used to model the data
model <- LogisticLogNormal(mean=c(-0.85, 1),
                           cov=
                             matrix(c(1, -0.5, -0.5, 1),
                                    nrow=2),
                           ref_dose=56)

# Set-up some MCMC parameters and generate samples from the posterior
options <- McmcOptions(burnin=100,
                       step=2,
                       samples=2000)
set.seed(94)
```

```
samples <- mcmc(data, model, options)

# Define the rule for dose increments and calculate the maximum dose allowed
myIncrements <- IncrementsRelative(intervals=c(0, 20),
                                   increments=c(1, 0.33))
nextMaxDose <- maxDose(myIncrements,
                       data=data)

# Define the rule which will be used to select the next best dose
# based on the class 'NextBestNCRM'
myNextBest <- NextBestNCRM(target=c(0.2, 0.35),
                           overdose=c(0.35, 1),
                           max_overdose_prob=0.25)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
                               doselimit=nextMaxDose,
                               samples=samples, model=model, data=data)

# Define the stopping rule such that the study would be stopped if at least 6
# cohorts were already dosed
myStopping <- StoppingMinCohorts(nCohorts = 6)

# Evaluate if to stop the trial
stopTrial(stopping=myStopping,
          dose=doseRecommendation$value,
          data=data)

# nolint end
# nolint start

# Create the data
data <- Data(x=c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
             y=c(0, 0, 0, 0, 0, 0, 1, 0),
             cohort=c(0, 1, 2, 3, 4, 5, 5, 5),
             doseGrid=
               c(0.1, 0.5, 1.5, 3, 6,
                 seq(from=10, to=80, by=2)))

# Initialize the CRM model used to model the data
model <- LogisticLogNormal(mean=c(-0.85, 1),
                           cov=
                             matrix(c(1, -0.5, -0.5, 1),
                                    nrow=2),
                           ref_dose=56)

# Set-up some MCMC parameters and generate samples from the posterior
options <- McmcOptions(burnin=100,
                       step=2,
                       samples=2000)
set.seed(94)
samples <- mcmc(data, model, options)
```

```
# Define the rule for dose increments and calculate the maximum dose allowed
myIncrements <- IncrementsRelative(intervals=c(0, 20),
                                   increments=c(1, 0.33))
nextMaxDose <- maxDose(myIncrements,
                       data=data)

# Define the rule which will be used to select the next best dose
# based on the class 'NextBestNCRM'
myNextBest <- NextBestNCRM(target=c(0.2, 0.35),
                           overdose=c(0.35, 1),
                           max_overdose_prob=0.25)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
                               doselimit=nextMaxDose,
                               samples=samples, model=model, data=data)

# Define the stopping rule such that the study would be stopped if at least 20
# patients were already dosed
myStopping <- StoppingMinPatients(nPatients = 20)

# Evaluate if to stop the trial
stopTrial(stopping=myStopping,
          dose=doseRecommendation$value,
          data=data)

# nolint end
# nolint start

# Create the data
data <- Data(x=c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
             y=c(0, 0, 0, 0, 0, 0, 1, 0),
             cohort=c(0, 1, 2, 3, 4, 5, 5, 5),
             doseGrid=
               c(0.1, 0.5, 1.5, 3, 6,
                 seq(from=10, to=80, by=2)))

# Initialize the CRM model used to model the data
model <- LogisticLogNormal(mean=c(-0.85, 1),
                           cov=
                             matrix(c(1, -0.5, -0.5, 1),
                                    nrow=2),
                           ref_dose=56)

# Set-up some MCMC parameters and generate samples from the posterior
options <- McmcOptions(burnin=100,
                       step=2,
                       samples=2000)
set.seed(94)
samples <- mcmc(data, model, options)

# Define the rule for dose increments and calculate the maximum dose allowed
myIncrements <- IncrementsRelative(intervals=c(0, 20),
```

```
                                    increments=c(1, 0.33))
nextMaxDose <- maxDose(myIncrements,
                       data=data)

# Define the rule which will be used to select the next best dose
# based on the class 'NextBestNCRM'
myNextBest <- NextBestNCRM(target=c(0.2, 0.35),
                           overdose=c(0.35, 1),
                           max_overdose_prob=0.25)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
                               doselimit=nextMaxDose,
                               samples=samples, model=model, data=data)

# Define the stopping rule such that the study would be stopped if there is at least
# 0.5 posterior probability that [0.2 =< Prob(DLT | next-best-dose) <= 0.35]
myStopping <- StoppingTargetProb(target=c(0.2, 0.35),
                                 prob=0.5)

# Evaluate if to stop the trial
stopTrial(stopping=myStopping,
          dose=doseRecommendation$value,
          samples=samples,
          model=model,
          data=data)

# nolint end
# nolint start

# Create the data
data <- Data(x=c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
             y=c(0, 0, 0, 0, 0, 0, 1, 0),
             cohort=c(0, 1, 2, 3, 4, 5, 5, 5),
             doseGrid=
               c(0.1, 0.5, 1.5, 3, 6,
                 seq(from=10, to=80, by=2)))

# Initialize the CRM model used to model the data
model <- LogisticLogNormal(mean=c(-0.85, 1),
                           cov=
                             matrix(c(1, -0.5, -0.5, 1),
                                    nrow=2),
                           ref_dose=56)

# Set-up some MCMC parameters and generate samples from the posterior
options <- McmcOptions(burnin=100,
                       step=2,
                       samples=2000)
set.seed(94)
samples <- mcmc(data, model, options)

# Define the rule for dose increments and calculate the maximum dose allowed
```

```
myIncrements <- IncrementsRelative(intervals=c(0, 20),
                                   increments=c(1, 0.33))
nextMaxDose <- maxDose(myIncrements,
                       data=data)

# Define the rule which will be used to select the next best dose
# based on the class 'NextBestNCRM'
myNextBest <- NextBestNCRM(target=c(0.2, 0.35),
                           overdose=c(0.35, 1),
                           max_overdose_prob=0.25)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
                               doselimit=nextMaxDose,
                               samples=samples, model=model, data=data)

# Define the stopping rule such that the study would be stopped if there is at least
# 0.9 probability that MTD > 0.5*next_best_dose. Here MTD is defined as the dose for
# which prob(DLE)=0.33
myStopping <- StoppingMTDdistribution(target = 0.33,
                                      thresh = 0.5,
                                      prob = 0.9)

# Evaluate if to stop the trial
stopTrial(stopping=myStopping,
          dose=doseRecommendation$value,
          samples=samples,
          model=model,
          data=data)

# nolint end
# Create the data.
my_data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  cohort = c(0, 1, 2, 3, 4, 5, 5, 5),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)

# Initialize the CRM model used to model the data.
my_model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 56
)

# Set-up some MCMC parameters and generate samples from the posterior.
my_options <- McmcOptions(
  burnin = 100, step = 2, samples = 2000, rng_kind = "Mersenne-Twister", rng_seed = 94
)
my_samples <- mcmc(my_data, my_model, my_options)

# Define the rule for dose increments and calculate the maximum dose allowed.
```

```
my_increments <- IncrementsRelative(intervals = c(0, 20), increments = c(1, 0.33))
next_max_dose <- maxDose(my_increments, data = my_data)

# Define the rule which will be used to select the next best dose
# based on the class 'NextBestNCRM'.
my_next_best <- NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Calculate the next best dose.
dose_recommendation <- nextBest(
  my_next_best,
  doselimit = next_max_dose,
  samples = my_samples,
  model = my_model,
  data = my_data
)

# Define the stopping rule such that the study would be stopped if the
# the MTD can be estimated with sufficient precision, i.e. if robust coefficient
# of variation is below 40%.
my_stopping <- StoppingMTDCV(target = 0.3, thresh_cv = 40)

# Evaluate if to stop the trial.
stopTrial(
  stopping = my_stopping,
  dose = dose_recommendation$value,
  samples = my_samples,
  model = my_model,
  data = my_data
)

# Create the data.
data <- Data(
  x = c(0.1, 0.1, 0.1),
  y = c(0, 0, 1),
  cohort = c(1, 1, 1),
  doseGrid = c(
    0.1, 0.5, 1.5, 3, 6,
    seq(from = 10, to = 80, by = 2)
  ),
  ID = 1:3
)

# Initialize the CRM model used to model the data.
model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 56
)
```

```
# Set-up some MCMC parameters and generate samples from the posterior.
options <- McmcOptions(
  burnin = 100,
  step = 2,
  samples = 2000
)
set.seed(94)
samples <- mcmc(data, model, options)

# Define the rule for dose increments and calculate the maximum dose allowed.
my_increments <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

next_max_dose <- maxDose(my_increments, data = data)

# Define the rule which will be used to select the next best dose
# based on the class 'NextBestNCRM'.
my_next_best <- NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Calculate the next best dose.
dose_recommendation <- nextBest(my_next_best,
  doselimit = next_max_dose,
  samples = samples, model = model, data = data
)

# Define the stopping rule such that the study would be stopped if first dose
# is toxic based on a Beta posterior distribution with Beta(1,1) prior.
my_stopping <- StoppingLowestDoseHSRBeta(
  target = 0.3,
  prob = 0.9
)

# Evaluate if the trial will be stopped.
stopTrial(
  stopping = my_stopping,
  dose = dose_recommendation$value,
  samples = samples,
  model = model,
  data = data
)
# nolint start

# Create the data
data <- DataDual(
  x=c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10,
      20, 20, 20, 40, 40, 40, 50, 50, 50),
  y=c(0, 0, 0, 0, 0, 0, 1, 0,
```

```
      0, 1, 1, 0, 0, 1, 0, 1, 1),
  w=c(0.31, 0.42, 0.59, 0.45, 0.6, 0.7, 0.55, 0.6,
      0.52, 0.54, 0.56, 0.43, 0.41, 0.39, 0.34, 0.38, 0.21),
  doseGrid=c(0.1, 0.5, 1.5, 3, 6,
             seq(from=10, to=80, by=2)))

# Initialize the Dual-Endpoint model (in this case RW1)
model <- DualEndpointRW(mean = c(0, 1),
                        cov = matrix(c(1, 0, 0, 1), nrow=2),
                        sigma2betaW = 0.01,
                        sigma2W = c(a=0.1, b=0.1),
                        rho = c(a=1, b=1),
                        rw1 = TRUE)

# Set-up some MCMC parameters and generate samples from the posterior
options <- McmcOptions(burnin=100,
                       step=2,
                       samples=500)
set.seed(94)
samples <- mcmc(data, model, options)

# Define the rule for dose increments and calculate the maximum dose allowed
myIncrements <- IncrementsRelative(intervals=c(0, 20),
                                   increments=c(1, 0.33))
nextMaxDose <- maxDose(myIncrements,
                       data=data)

# Define the rule which will be used to select the next best dose
# In this case target a dose achieving at least 0.9 of maximum biomarker level (efficacy)
# and with a probability below 0.25 that prob(DLT)>0.35 (safety)
myNextBest <- NextBestDualEndpoint(target=c(0.9, 1),
                                   overdose=c(0.35, 1),
                                   max_overdose_prob=0.25)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
                               doselimit=nextMaxDose,
                               samples=samples,
                               model=model,
                               data=data)

# Define the stopping rule such that the study would be stopped if if there is at
# least 0.5 posterior probability that the biomarker (efficacy) is within the
# biomarker target range of [0.9, 1.0] (relative to the maximum for the biomarker).
myStopping <- StoppingTargetBiomarker(target = c(0.9, 1),
                                      prob = 0.5)

# Evaluate if to stop the trial
stopTrial(stopping=myStopping,
          dose=doseRecommendation$value,
          samples=samples,
          model=model,
          data=data)
```

```
# nolint end

# Create some data.
my_data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  ID = 1:8,
  cohort = c(0, 1, 2, 3, 4, 5, 5, 5),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)

# Initialize the CRM model used to model the data.
my_model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
  ref_dose = 50
)

# Set-up some MCMC parameters and generate samples from the posterior.
my_options <- McmcOptions(burnin = 100, step = 2, samples = 500)
my_samples <- mcmc(my_data, my_model, my_options)

# Define the rule which will be used to select the next best dose
# based on the class 'NextBestNCRM'.
my_nb_ncrm <- NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Calculate the next best dose.
my_dose_recommendation <- nextBest(
  nextBest = my_nb_ncrm,
  doselimit = 100,
  samples = my_samples,
  model = my_model,
  data = my_data
)

# Define the stopping rules.
highest_dose_safe <- StoppingSpecificDose(
  rule = StoppingTargetProb(target = c(0, 0.3), prob = 0.8),
  dose = 80
)
max_patients <- StoppingMinPatients(nPatients = 20)
patients_near_dose <- StoppingPatientsNearDose(nPatients = 3, percentage = 0)

# Create a list of stopping rules (of class 'StoppingList') which will then be
# summarized (in this specific example) with the 'any' function, meaning that
# the study would be stopped if 'any' of the single stopping rules is TRUE.
my_stopping <- highest_dose_safe | max_patients | patients_near_dose
```

```
# Evaluate if to stop the Trial
stopTrial(
  stopping = my_stopping,
  dose = doseRecommendation$value,
  samples = samples,
  model = model,
  data = data
)
# nolint start

# Create the data
data <- Data(x=c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10, 20, 20, 20, 40, 40, 40,
                 80, 80, 80),
             y=c(0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
             cohort=c(0, 1, 2, 3, 4, 5, 5, 5, 6, 6, 6, 7, 7, 7, 8, 8, 8),
             doseGrid=
               c(0.1, 0.5, 1.5, 3, 6,
                 seq(from=10, to=80, by=2)))

# Initialize the CRM model used to model the data
model <- LogisticLogNormal(mean=c(-0.85, 1),
                           cov=
                             matrix(c(1, -0.5, -0.5, 1),
                                    nrow=2),
                           ref_dose=56)

# Set-up some MCMC parameters and generate samples from the posterior
options <- McmcOptions(burnin=100,
                       step=2,
                       samples=2000)
set.seed(94)
samples <- mcmc(data, model, options)

# Define the rule for dose increments and calculate the maximum dose allowed
myIncrements <- IncrementsRelative(intervals=c(0, 20),
                                   increments=c(1, 0.33))
nextMaxDose <- maxDose(myIncrements,
                       data=data)

# Define the rule which will be used to select the next best dose
# based on the class 'NextBestNCRM'
myNextBest <- NextBestNCRM(target=c(0.2, 0.35),
                           overdose=c(0.35, 1),
                           max_overdose_prob=0.25)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
                               doselimit=nextMaxDose,
                               samples=samples, model=model, data=data)

# Define the stopping rule such that the study would be stopped if there is at least
# 0.5 posterior probability that [0.2 =< Prob(DLT | next-best-dose) <= 0.35]
stopTarget <- StoppingTargetProb(target=c(0.2, 0.35),
```

```
                                prob=0.5)

## now use the StoppingHighestDose rule:
stopHigh <-
  StoppingHighestDose() &
  StoppingPatientsNearDose(nPatients=3, percentage=0) &
  StoppingTargetProb(target=c(0, 0.2),
                     prob=0.5)

## and combine everything:
myStopping <- stopTarget | stopHigh

# Then evaluate if to stop the trial
stopTrial(stopping=myStopping,
          dose=doseRecommendation$value,
          samples=samples,
          model=model,
          data=data)

# nolint end
data <- .DefaultDataOrdinal()
model <- .DefaultLogisticLogNormalOrdinal()
options <- .DefaultMcmcOptions()
samples <- mcmc(data, model, options)

myIncrements <- .DefaultIncrementsOrdinal()
nextMaxDose <- maxDose(myIncrements, data = data)

myNextBest <- .DefaultNextBestOrdinal()

doseRecommendation <- nextBest(
  myNextBest,
  doselimit = nextMaxDose,
  samples = samples,
  model = model,
  data = data
)

myStopping <- .DefaultStoppingOrdinal()

stopTrial(
  stopping = myStopping,
  dose = doseRecommendation$value,
  samples = samples,
  model = model,
  data = data
)
data <- .DefaultDataOrdinal()
model <- .DefaultLogisticLogNormalOrdinal()
options <- .DefaultMcmcOptions()
samples <- mcmc(data, model, options)

myIncrements <- .DefaultIncrementsOrdinal()
```

```
nextMaxDose <- maxDose(myIncrements, data = data)

myNextBest <- .DefaultNextBestOrdinal()

doseRecommendation <- nextBest(
  myNextBest,
  doselimit = nextMaxDose,
  samples = samples,
  model = model,
  data = data
)

myStopping <- .DefaultStoppingOrdinal()

stopTrial(
  stopping = myStopping,
  dose = doseRecommendation$value,
  samples = samples,
  model = model,
  data = data
)
my_rule <- StoppingExternal(report_label = "Based on combo stop")

stopTrial(my_rule, 5, .DefaultSamples(), .DefaultModelLogNormal(), .DefaultData(), external = TRUE)
# nolint start

##define the stopping rules based on the 'StoppingTDCIRatio' class
##Using only DLE responses with samples
## we need a data object with doses >= 1:
data<-Data(x=c(25,50,50,75,150,200,225,300),
           y=c(0,0,0,0,1,1,1,1),
           doseGrid=seq(from=25,to=300,by=25))

##model can be specified of 'Model' or 'ModelTox' class
##For example, the 'logisticIndepBeta' class model
model<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=data)
##define MCMC options
##for illustration purpose we use 10 burn-in and generate 50 samples
options<-McmcOptions(burnin=10,step=2,samples=50)
##samples of 'Samples' class
samples<-mcmc(data,model,options)
##define the 'StoppingTDCIRatio' class
myStopping <- StoppingTDCIRatio(target_ratio = 5, prob_target = 0.3)
##Find the next Recommend dose using the nextBest method (plesae refer to nextbest examples)
tdNextBest <- NextBestTDsamples(
  prob_target_drt = 0.35, prob_target_eot = 0.3,
  derive = function(samples) {
    as.numeric(quantile(samples, probs = 0.3))
  }
)

RecommendDose<-nextBest(tdNextBest,doselimit=max(data@doseGrid),samples=samples,
                        model=model,data=data)
```

```
##use 'stopTrial' to determine if the rule has been fulfilled
##use 0.3 as the target proability of DLE at the end of the trial

stopTrial(stopping=myStopping,dose=RecommendDose$next_dose_drt,
          samples=samples,model=model,data=data)

# nolint end
# nolint start

##define the stopping rules based on the 'StoppingTDCIRatio' class
##Using only DLE responses
## we need a data object with doses >= 1:
data<-Data(x=c(25,50,50,75,150,200,225,300),
           y=c(0,0,0,0,1,1,1,1),
           doseGrid=seq(from=25,to=300,by=25))

##model must be of 'ModelTox' class
##For example, the 'logisticIndepBeta' class model
model<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=data)
##define the 'StoppingTDCIRatio' class
myStopping <- StoppingTDCIRatio(target_ratio = 5, prob_target = 0.3)
##Find the next Recommend dose using the nextBest method (plesae refer to nextbest examples)
tdNextBest<-NextBestTD(prob_target_drt=0.35,prob_target_eot=0.3)


RecommendDose<-nextBest(tdNextBest,doselimit=max(data@doseGrid),model=model,data=data)
##use 'stopTrial' to determine if the rule has been fulfilled
##use 0.3 as the target proability of DLE at the end of the trial

stopTrial(stopping=myStopping,dose=RecommendDose$next_dose_drt,
          model=model,data=data)

# nolint end
# nolint start
##define the stopping rules based on the 'StoppingMaxGainCIRatio' class
##Using both DLE and efficacy responses
## we need a data object with doses >= 1:
data <-DataDual(x=c(25,50,25,50,75,300,250,150),
                y=c(0,0,0,0,0,1,1,0),
                w=c(0.31,0.42,0.59,0.45,0.6,0.7,0.6,0.52),
                doseGrid=seq(25,300,25),
                placebo=FALSE)

##DLEmodel must be of 'ModelTox' class
##For example, the 'logisticIndepBeta' class model
DLEmodel<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=data)

##Effmodel must be  of 'ModelEff' class
##For example, the 'Effloglog' class model
Effmodel<-Effloglog(eff=c(1.223,2.513),eff_dose=c(25,300),nu=c(a=1,b=0.025),data=data)
##for illustration purpose we use 10 burn-in and generate 50 samples
options<-McmcOptions(burnin=10,step=2,samples=50)
##DLE and efficacy samples must be of 'Samples' class
```

```
DLEsamples<-mcmc(data,DLEmodel,options)
Effsamples<-mcmc(data,Effmodel,options)

##define the 'StoppingMaxGainCIRatio' class
myStopping <- StoppingMaxGainCIRatio(target_ratio = 5, prob_target = 0.3)
##Find the next Recommend dose using the nextBest method (plesae refer to nextbest examples)
mynextbest <- NextBestMaxGainSamples(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3,
  derive = function(samples) {
    as.numeric(quantile(samples, prob = 0.3))
  },
  mg_derive = function(mg_samples) {
    as.numeric(quantile(mg_samples, prob = 0.5))
  }
)

RecommendDose<-nextBest(mynextbest,doselimit=max(data@doseGrid),samples=DLEsamples,model=DLEmodel,
                        data=data,model_eff=Effmodel,samples_eff=Effsamples)
##use 'stopTrial' to determine if the rule has been fulfilled
##use 0.3 as the target proability of DLE at the end of the trial

stopTrial(stopping=myStopping,
          dose=RecommendDose$next_dose,
          samples=DLEsamples,
          model=DLEmodel,
          data=data,
          TDderive=function(TDsamples){
            quantile(TDsamples,prob=0.3)},
          Effmodel=Effmodel,
          Effsamples=Effsamples,
          Gstarderive=function(Gstarsamples){
            quantile(Gstarsamples,prob=0.5)})

# nolint end
# nolint start

##define the stopping rules based on the 'StoppingMaxGainCIRatio' class
##Using both DLE and efficacy responses
## we need a data object with doses >= 1:
data <-DataDual(x=c(25,50,25,50,75,300,250,150),
                y=c(0,0,0,0,0,1,1,0),
                w=c(0.31,0.42,0.59,0.45,0.6,0.7,0.6,0.52),
                doseGrid=seq(25,300,25),
                placebo=FALSE)

##DLEmodel must be of 'ModelTox' class
##For example, the 'logisticIndepBeta' class model
DLEmodel<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=data)

##Effmodel must be  of 'ModelEff' class
##For example, the 'Effloglog' class model
Effmodel<-Effloglog(eff=c(1.223,2.513),eff_dose=c(25,300),nu=c(a=1,b=0.025),data=data)
```

```
##define the 'StoppingMaxGainCIRatio' class
myStopping <- StoppingMaxGainCIRatio(target_ratio = 5, prob_target = 0.3)
##Find the next Recommend dose using the nextBest method (plesae refer to nextbest examples)
mynextbest<-NextBestMaxGain(prob_target_drt=0.35, prob_target_eot=0.3)

RecommendDose<-nextBest(mynextbest,doselimit=max(data@doseGrid),model=DLEmodel,
                        model_eff=Effmodel,data=data)

##use 'stopTrial' to determine if the rule has been fulfilled
##use 0.3 as the target proability of DLE at the end of the trial


stopTrial(stopping=myStopping,dose=RecommendDose$next_dose,model=DLEmodel,
          data=data, Effmodel=Effmodel)

# nolint end
```

---

summary,DualSimulations-method

Summarize the dual-endpoint design simulations, relative to given true dose-toxicity and dose-biomarker curves

---

Description

Summarize the dual-endpoint design simulations, relative to given true dose-toxicity and dose-biomarker curves

Usage

```
## S4 method for signature 'DualSimulations'
summary(object, trueTox, trueBiomarker, target = c(0.2, 0.35), ...)
```

Arguments

object	the DualSimulations object we want to summarize
trueTox	a function which takes as input a dose (vector) and returns the true probability (vector) for toxicity.
trueBiomarker	a function which takes as input a dose (vector) and returns the true biomarker level (vector).
target	the target toxicity interval (default: 20-35%) used for the computations
...	Additional arguments can be supplied here for trueTox and trueBiomarker

Value

an object of class DualSimulationsSummary

Examples

```
# Define the dose-grid.
emptydata <- DataDual(doseGrid = c(1, 3, 5, 10, 15, 20, 25, 40, 50, 80, 100))


# Initialize the CRM model.
my_model <- DualEndpointRW(
  mean = c(0, 1),
  cov = matrix(c(1, 0, 0, 1), nrow = 2),
  sigma2betaW = 0.01,
  sigma2W = c(a = 0.1, b = 0.1),
  rho = c(a = 1, b = 1),
  rw1 = TRUE
)

# Choose the rule for selecting the next dose.
my_next_best <- NextBestDualEndpoint(
  target = c(0.9, 1),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Choose the rule for stopping.
my_stopping1 <- StoppingTargetBiomarker(
  target = c(0.9, 1),
  prob = 0.5
)

# For illustration stop with 6 subjects.
my_stopping <- my_stopping1 | StoppingMinPatients(6) | StoppingMissingDose()

# Choose the rule for dose increments.
my_increments <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

# Initialize the design.
design <- DualDesign(
  model = my_model,
  data = emptydata,
  nextBest = my_next_best,
  stopping = my_stopping,
  increments = my_increments,
  cohort_size = CohortSizeConst(3),
  startingDose = 3
)

# Define scenarios for the TRUE toxicity and efficacy profiles.
beta_mod <- function(dose, e0, eMax, delta1, delta2, scal) {
  maxDens <- (delta1^delta1) * (delta2^delta2) / ((delta1 + delta2)^(delta1 + delta2))
  dose <- dose / scal
```

```
  e0 + eMax / maxDens * (dose^delta1) * (1 - dose)^delta2
}

true_biomarker <- function(dose) {
  beta_mod(dose, e0 = 0.2, eMax = 0.6, delta1 = 5, delta2 = 5 * 0.5 / 0.5, scal = 100)
}

true_tox <- function(dose) {
  pnorm((dose - 60) / 10)
}

# Draw the TRUE profiles.
par(mfrow = c(1, 2))
curve(true_tox(x), from = 0, to = 80)
curve(true_biomarker(x), from = 0, to = 80)

# Run the simulation on the desired design.
# For illustration purposes 1 trial is simulated with 5 burn-ins to generate 20 samples.
my_sims <- simulate(
  object = design,
  trueTox = true_tox,
  trueBiomarker = true_biomarker,
  sigma2W = 0.01,
  rho = 0,
  nsim = 1,
  parallel = FALSE,
  seed = 3,
  startingDose = 6,
  mcmcOptions = McmcOptions(
    burnin = 5,
    step = 1,
    samples = 20
  )
)

# Summarize the results of the simulations.
summary(
  my_sims,
  trueTox = true_tox,
  trueBiomarker = true_biomarker
)
```

---

summary,GeneralSimulations-method

Summarize the simulations, relative to a given truth

Description

Summarize the simulations, relative to a given truth

Usage

```
## S4 method for signature 'GeneralSimulations'
summary(object, truth, target = c(0.2, 0.35), ...)
```

Arguments

object	the GeneralSimulations object we want to summarize
truth	a function which takes as input a dose (vector) and returns the true probability (vector) for toxicity
target	the target toxicity interval (default: 20-35%) used for the computations
...	Additional arguments can be supplied here for truth

Value

an object of class GeneralSimulationsSummary

---

summary,PseudoDualFlexiSimulations-method

Summary for Pseudo Dual responses simulations given a pseudo DLE model and the Flexible efficacy model.

Description

Summary for Pseudo Dual responses simulations given a pseudo DLE model and the Flexible efficacy model.

Usage

```
## S4 method for signature 'PseudoDualFlexiSimulations'
summary(
  object,
  trueDLE,
  trueEff,
  targetEndOfTrial = 0.3,
  targetDuringTrial = 0.35,
  ...
)
```

Arguments

object	the PseudoDualFlexiSimulations object we want to summarize
trueDLE	a function which takes as input a dose (vector) and returns the true probability of DLE (vector)
trueEff	a vector which takes as input the true mean efficacy values at all dose levels (in order)

`targetEndOfTrial`
: the target probability of DLE that are used at the end of a trial. Default at 0.3.

`targetDuringTrial`
: the target probability of DLE that are used during the trial. Default at 0.35.

`...`
: Additional arguments can be supplied here for `trueDLE` and `trueEff`

Value

an object of class `PseudoDualSimulationsSummary`

Examples

```
# nolint start

## If DLE and efficacy responses are considered in the simulations and the 'EffFlexi' class is used
## we need a data object with doses >= 1:
data <- DataDual(doseGrid = seq(25, 300, 25))
## First for the DLE model
## The DLE model must be of 'ModelTox' (e.g 'LogisticIndepBeta') class
DLEmodel <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = data
)

## for the efficacy model
Effmodel <- EffFlexi(
  eff = c(1.223, 2.513), eff_dose = c(25, 300),
  sigma2W = c(a = 0.1, b = 0.1), sigma2betaW = c(a = 20, b = 50), rw1 = FALSE, data = data
)


## specified the next best
mynextbest <- NextBestMaxGainSamples(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3,
  derive = function(samples) {
    as.numeric(quantile(samples, prob = 0.3))
  },
  mg_derive = function(mg_samples) {
    as.numeric(quantile(mg_samples, prob = 0.5))
  }
)

## The increments (see Increments class examples)
## 200% allowable increase for dose below 300 and 200% increase for dose above 300
myIncrements <- IncrementsRelative(
  intervals = c(25, 300),
  increments = c(2, 2)
)
## cohort size of 3
```

```
mySize <- CohortSizeConst(size = 3)
## Stop only when 10 subjects are treated:
## very low sample size is just for illustration here
myStopping <- StoppingMinPatients(nPatients = 10)

## Specified the design
design <- DualResponsesSamplesDesign(
  nextBest = mynextbest,
  cohort_size = mySize,
  startingDose = 25,
  model = DLEmodel,
  eff_model = Effmodel,
  data = data,
  stopping = myStopping,
  increments = myIncrements
)
## specified the true DLE curve and the true expected efficacy values at all dose levels
myTruthDLE <- probFunction(DLEmodel, phi1 = -53.66584, phi2 = 10.50499)

myTruthEff <- c(
  -0.5478867, 0.1645417, 0.5248031, 0.7604467,
  0.9333009, 1.0687031, 1.1793942, 1.2726408,
  1.3529598, 1.4233411, 1.4858613, 1.5420182
)


## specify the options for MCMC
# For illustration purpose, we use 10 burn-in and generate 100 samples
options <- McmcOptions(burnin = 10, step = 1, samples = 100)
## The simulation
## For illustration purpose only 1 simulation is produced (nsim=1).
mySim <- simulate(
  object = design,
  args = NULL,
  trueDLE = myTruthDLE,
  trueEff = myTruthEff,
  trueSigma2 = 0.025,
  trueSigma2betaW = 1,
  nsim = 1,
  seed = 819,
  parallel = FALSE,
  mcmcOptions = options
)
## summarize the simulation results
summary(mySim,
  trueDLE = myTruthDLE,
  trueEff = myTruthEff
)

# nolint end
```

summary,PseudoDualSimulations-method

Summary for Pseudo Dual responses simulations, relative to a given pseudo DLE and efficacy model (except the EffFlexi class model)

Description

Summary for Pseudo Dual responses simulations, relative to a given pseudo DLE and efficacy model (except the EffFlexi class model)

Usage

```
## S4 method for signature 'PseudoDualSimulations'
summary(
  object,
  trueDLE,
  trueEff,
  targetEndOfTrial = 0.3,
  targetDuringTrial = 0.35,
  ...
)
```

Arguments

`object`	the `PseudoDualSimulations` object we want to summarize
`trueDLE`	a function which takes as input a dose (vector) and returns the true probability (vector) of DLE
`trueEff`	a function which takes as input a dose (vector) and returns the mean efficacy value(s) (vector).
`targetEndOfTrial`	the target probability of DLE that are used at the end of a trial. Default at 0.3.
`targetDuringTrial`	the target probability of DLE that are used during the trial. Default at 0.35.
`...`	Additional arguments can be supplied here for `trueDLE` and `trueEff`

Value

an object of class `PseudoDualSimulationsSummary`

Examples

```
# Obtain the plot for the simulation results if DLE and efficacy responses
# are considered in the simulations.

# Specified simulations when no samples are used.
emptydata <- DataDual(doseGrid = seq(25, 300, 25))
```

```
# The DLE model must be of 'ModelTox' (e.g 'LogisticIndepBeta') class.
dle_model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = emptydata
)

# The efficacy model of 'ModelEff' (e.g 'Effloglog') class.
eff_model <- Effloglog(
  eff = c(1.223, 2.513),
  eff_dose = c(25, 300),
  nu = c(a = 1, b = 0.025),
  data = emptydata
)

# The escalation rule using the 'NextBestMaxGain' class.
my_next_best <- NextBestMaxGain(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3
)

# Allow increase of 200%.
my_increments <- IncrementsRelative(intervals = 0, increments = 2)

# Cohort size of 3.
my_size <- CohortSizeConst(size = 3)

# Stop when 36 subjects are treated or next dose is NA.
my_stopping <- StoppingMinPatients(nPatients = 36) | StoppingMissingDose()

# Specify the design. (For details please refer to the 'DualResponsesDesign' example.)
my_design <- DualResponsesDesign(
  nextBest = my_next_best,
  model = dle_model,
  eff_model = eff_model,
  stopping = my_stopping,
  increments = my_increments,
  cohort_size = my_size,
  data = emptydata,
  startingDose = 25
)

# Specify the true DLE and efficacy curves.
my_truth_dle <- probFunction(dle_model, phi1 = -53.66584, phi2 = 10.50499)
my_truth_eff <- efficacyFunction(eff_model, theta1 = -4.818429, theta2 = 3.653058)

# Specify the simulations and generate the 2 trials.
my_sim <- simulate(
  object = my_design,
  args = NULL,
  trueDLE = my_truth_dle,
```

```
  trueEff = my_truth_eff,
  trueNu = 1 / 0.025,
  nsim = 2,
  seed = 819,
  parallel = FALSE
)

# Produce a summary of the simulations.
summary(
  my_sim,
  trueDLE = my_truth_dle,
  trueEff = my_truth_eff
)

# Example where DLE and efficacy samples are involved.
# Please refer to design-method 'simulate DualResponsesSamplesDesign' examples for details.

# Specify the next best rule.
my_next_best <- NextBestMaxGainSamples(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3,
  derive = function(samples) {
    as.numeric(quantile(samples, prob = 0.3))
  },
  mg_derive = function(mg_samples) {
    as.numeric(quantile(mg_samples, prob = 0.5))
  }
)

# Specify the design.
my_design <- DualResponsesSamplesDesign(
  nextBest = my_next_best,
  cohort_size = my_size,
  startingDose = 25,
  model = dle_model,
  eff_model = eff_model,
  data = emptydata,
  stopping = my_stopping,
  increments = my_increments
)

# For illustration purpose 50 burn-ins to generate 200 samples are used.
my_options <- McmcOptions(burnin = 50, step = 2, samples = 200)

# For illustration purpose 2 simulation are created.
my_sim <- simulate(
  object = my_design,
  args = NULL,
  trueDLE = my_truth_dle,
  trueEff = my_truth_eff,
  trueNu = 1 / 0.025,
  nsim = 2,
  mcmcOptions = my_options,
```

```
  seed = 819,
  parallel = FALSE
)

# Produce a summary of the simulations.
summary(
  my_sim,
  trueDLE = my_truth_dle,
  trueEff = my_truth_eff
)
```

---

summary,PseudoSimulations-method
Summarize the simulations, relative to a given truth

Description

Summarize the simulations, relative to a given truth

Usage

```
## S4 method for signature 'PseudoSimulations'
summary(object, truth, targetEndOfTrial = 0.3, targetDuringTrial = 0.35, ...)
```

Arguments

`object`	the `PseudoSimulations` object we want to summarize
`truth`	a function which takes as input a dose (vector) and returns the true probability (vector) for toxicity
`targetEndOfTrial`	the target probability of DLE wanted to achieve at the end of a trial
`targetDuringTrial`	the target probability of DLE wanted to achieve during a trial
`...`	Additional arguments can be supplied here for `truth`

Value

an object of class `PseudoSimulationsSummary`

Examples

```
emptydata <- Data(doseGrid = seq(25, 300, 25))

# The design incorporate DLE responses and DLE samples.
# Specify the model of 'ModelTox' class eg 'LogisticIndepBeta' class model.
my_model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
```

```
  DLEdose = c(25, 300),
  data = emptydata
)

# The escalation rule.
td_next_best <- NextBestTD(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3
)

# Cohort size is 3 subjects.
my_size <- CohortSizeConst(size = 3)

# Allow increase of 200%.
my_increments <- IncrementsRelative(intervals = 0, increments = 2)

# Stopp when the maximum sample size of 36 patients has been reached or the next
# dose is NA.
my_stopping <- StoppingMinPatients(nPatients = 36) | StoppingMissingDose()

# Specify the design. (For details please refer to the 'TDDesign' example.)
my_design <- TDDesign(
  model = my_model,
  nextBest = td_next_best,
  stopping = my_stopping,
  increments = my_increments,
  cohort_size = my_size,
  data = emptydata,
  startingDose = 25
)

# Specify the truth of the DLE responses.
my_truth <- probFunction(my_model, phi1 = -53.66584, phi2 = 10.50499)

# For illustration purpose 50 burn-ins to generate 200 samples are used.
my_options <- McmcOptions(burnin = 50, step = 2, samples = 200)

# Refer to design-method 'simulate TDDesign' examples for details.
# For illustration purpose only 1 simulation is produced.
my_sim <- simulate(
  object = my_design,
  args = NULL,
  truth = my_truth,
  nsim = 1,
  seed = 819,
  parallel = FALSE,
  mcmcOptions = my_options
)

# Produce a summary of the simulations.
summary(
  my_sim,
  truth = my_truth
```

```
)

# Example where DLE samples are involved.

# Specify the next best rule.
td_next_best <- NextBestTDsamples(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3,
  derive = function(samples) {
    as.numeric(quantile(samples, probs = 0.3))
  }
)

# The design.
my_design <- TDsamplesDesign(
  model = my_model,
  nextBest = td_next_best,
  stopping = my_stopping,
  increments = my_increments,
  cohort_size = my_size,
  data = emptydata,
  startingDose = 25
)

# For illustration purpose 50 burn-ins to generate 200 samples are used.
my_options <- McmcOptions(burnin = 50, step = 2, samples = 200)

# For illustration purpose 2 trials are simulated.
my_sim <- simulate(
  object = my_design,
  args = NULL,
  truth = my_truth,
  nsim = 2,
  seed = 819,
  mcmcOptions = my_options,
  parallel = FALSE
)

# Produce a summary of the simulations.
summary(
  my_sim,
  truth = my_truth
)
```

---

summary,Simulations-method

Summarize the model-based design simulations, relative to a given truth

---

Description

Summarize the model-based design simulations, relative to a given truth

Usage

```
## S4 method for signature 'Simulations'
summary(object, truth, target = c(0.2, 0.35), ...)
```

Arguments

object	the Simulations object we want to summarize
truth	a function which takes as input a dose (vector) and returns the true probability (vector) for toxicity
target	the target toxicity interval (default: 20-35%) used for the computations
...	Additional arguments can be supplied here for truth

Value

an object of class SimulationsSummary

Examples

```
# nolint start

# Define the dose-grid
emptydata <- Data(doseGrid = c(1, 3, 5, 10, 15, 20, 25, 40, 50, 80, 100))

# Initialize the CRM model
model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov =
    matrix(c(1, -0.5, -0.5, 1),
      nrow = 2
    ),
  ref_dose = 56
)

# Choose the rule for selecting the next dose
myNextBest <- NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
)

# Choose the rule for the cohort-size
mySize1 <- CohortSizeRange(
  intervals = c(0, 30),
  cohort_size = c(1, 3)
)
mySize2 <- CohortSizeDLT(
```

```
  intervals = c(0, 1),
  cohort_size = c(1, 3)
)
mySize <- maxSize(mySize1, mySize2)

# Choose the rule for stopping
myStopping1 <- StoppingMinCohorts(nCohorts = 3)
myStopping2 <- StoppingTargetProb(
  target = c(0.2, 0.35),
  prob = 0.5
)
myStopping3 <- StoppingMinPatients(nPatients = 20)
myStopping <- (myStopping1 & myStopping2) | myStopping3

# Choose the rule for dose increments
myIncrements <- IncrementsRelative(
  intervals = c(0, 20),
  increments = c(1, 0.33)
)

# Initialize the design
design <- Design(
  model = model,
  nextBest = myNextBest,
  stopping = myStopping,
  increments = myIncrements,
  cohort_size = mySize,
  data = emptydata,
  startingDose = 3
)

## define the true function
myTruth <- probFunction(model, alpha0 = 7, alpha1 = 8)

# Run the simulation on the desired design
# We only generate 1 trial outcomes here for illustration, for the actual study
# this should be increased of course
options <- McmcOptions(
  burnin = 100,
  step = 2,
  samples = 1000
)
time <- system.time(mySims <- simulate(design,
  args = NULL,
  truth = myTruth,
  nsim = 1,
  seed = 819,
  mcmcOptions = options,
  parallel = FALSE,
  derive = list(
    max_mtd = max,
    mean_mtd = mean,
    median_mtd = median
```

```
  ),
))[3]

# Summarize the Results of the Simulations
summary(mySims, truth = myTruth)

# nolint end
```

`TDDesign-class` `TDDesign`

Description

[Stable]

`TDDesign` is the class of design based only on DLT responses using `ModelTox` class model (i.e. `LogisticIndepBeta`) without MCMC samples.

Usage

```
TDDesign(
  model,
  stopping,
  increments,
  pl_cohort_size = CohortSizeConst(0L),
  ...
)

.DefaultTDDesign()
```

Arguments

`model`	(`ModelTox`) see slot definition.
`stopping`	(`Stopping`) see slot definition.
`increments`	(`Increments`) see slot definition.
`pl_cohort_size`	(`CohortSize`) see slot definition.
`...`	Arguments passed on to `RuleDesign` `nextBest` (`NextBest`) see slot definition. `cohort_size` (`CohortSize`) see slot definition. `data` (`Data`) see slot definition. `startingDose` (`number`) see slot definition.

Slots

`model` (`ModelTox`)
: the pseudo DLT model to be used.

`stopping` (`Stopping`)
: stopping rule(s) for the trial.

`increments` (`Increments`)
: how to control increments between dose levels.

`pl_cohort_size` (`CohortSize`)
: rules for the cohort sizes for placebo, if any planned (defaults to constant 0 placebo patients).

Note

Typically, end users will not use the `.DefaultTDDesign()` function.

Examples

```
empty_data <- Data(doseGrid = seq(25, 300, 25))

my_model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = empty_data
)

# The escalation rule.
my_next_best <- NextBestTD(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3
)

my_size <- CohortSizeConst(size = 3)

# The increments for the dose-escalation process:
#  the maximum increase of 200% for doses up to the maximum dose in grid,
#  the maximum increase of 200% for dose above the maximum dose in grid.
my_increments <- IncrementsRelative(
  intervals = range(empty_data@doseGrid),
  increments = c(2, 2)
)

# Stop when the maximum sample size of 36 patients is reached.
my_stopping <- StoppingMinPatients(nPatients = 36)

# The design with all the above information and starting with a dose of 25.
# This design incorporates only DLT responses and no DLT samples are involved
# during the simulation.
design <- TDDesign(
  model = my_model,
  stopping = my_stopping,
```

```
  increments = my_increments,
  nextBest = my_next_best,
  cohort_size = my_size,
  data = empty_data,
  startingDose = 25
)
```

TDsamplesDesign-class TDsamplesDesign

Description

[Stable]

TDsamplesDesign is the class of design based only on DLT responses using ModelTox class model (i.e. LogisticIndepBeta) as well as MCMC samples obtained for this model.

Usage

```
TDsamplesDesign(
  model,
  stopping,
  increments,
  pl_cohort_size = CohortSizeConst(0L),
  ...
)

.DefaultTDsamplesDesign()
```

Arguments

model	(ModelTox) see slot definition.
stopping	(Stopping) see slot definition.
increments	(Increments) see slot definition.
pl_cohort_size	(CohortSize) see slot definition.
...	Arguments passed on to RuleDesign nextBest (NextBest) see slot definition. cohort_size (CohortSize) see slot definition. data (Data) see slot definition. startingDose (number) see slot definition.

Slots

`model` (`ModelTox`)
: the pseudo DLT model to be used.

`stopping` (`Stopping`)
: stopping rule(s) for the trial.

`increments` (`Increments`)
: how to control increments between dose levels.

`pl_cohort_size` (`CohortSize`)
: rules for the cohort sizes for placebo, if any planned (defaults to constant 0 placebo patients).

Note

Typically, end users will not use the `.DefaultTDsamplesDesign()` function.

Examples

```
empty_data <- Data(doseGrid = seq(25, 300, 25))

my_model <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
  DLEdose = c(25, 300),
  data = empty_data
)

# The escalation rule.
my_next_best <- NextBestTDsamples(
  prob_target_drt = 0.35,
  prob_target_eot = 0.3,
  derive = function(samples) {
    as.numeric(quantile(samples, probs = 0.3))
  }
)

my_size <- CohortSizeConst(size = 3)

# The increments for the dose-escalation process:
#  the maximum increase of 200% for doses up to the maximum dose in grid,
#  the maximum increase of 200% for dose above the maximum dose in grid.
my_increments <- IncrementsRelative(
  intervals = range(empty_data@doseGrid),
  increments = c(2, 2)
)

# Stop when the maximum sample size of 36 patients is reached.
my_stopping <- StoppingMinPatients(nPatients = 36)

# The design with all the above information and starting with a dose of 25.
design <- TDsamplesDesign(
  model = my_model,
```

```
  stopping = my_stopping,
  increments = my_increments,
  nextBest = my_next_best,
  cohort_size = my_size,
  data = empty_data,
  startingDose = 25
)
```

`tidy` *Tidying* `CrmPackClass` *objects*

Description

[Experimental]

In the spirit of the `broom` package, provide a method to convert a `CrmPackClass` object to a (list of) tibbles.

Following the principles of the `broom` package, convert a `CrmPackClass` object to a (list of) tibbles. This is a basic, default representation.

[Experimental]

A method that tidies a `GeneralData` object.

[Experimental]

A method that tidies a `DataGrouped` object.

[Experimental]

A method that tidies a `DataDA` object.

[Experimental]

A method that tidies a `DataDual` object.

[Experimental]

A method that tidies a `DataParts` object.

[Experimental]

A method that tidies a `DataMixture` object.

[Experimental]

A method that tidies a `DataOrdinal` object.

[Experimental]

A method that tidies a `LogisticIndepBeta` object.

[Experimental]

A method that tidies a `Effloglog` object.

Usage

```
tidy(x, ...)

## S4 method for signature 'CrmPackClass'
tidy(x, ...)

## S4 method for signature 'GeneralData'
tidy(x, ...)

## S4 method for signature 'DataGrouped'
tidy(x, ...)

## S4 method for signature 'DataDA'
tidy(x, ...)

## S4 method for signature 'DataDual'
tidy(x, ...)

## S4 method for signature 'DataParts'
tidy(x, ...)

## S4 method for signature 'DataMixture'
tidy(x, ...)

## S4 method for signature 'DataOrdinal'
tidy(x, ...)

## S4 method for signature 'Simulations'
tidy(x, ...)

## S4 method for signature 'LogisticIndepBeta'
tidy(x, ...)

## S4 method for signature 'Effloglog'
tidy(x, ...)

## S4 method for signature 'IncrementsRelative'
tidy(x, ...)

## S4 method for signature 'CohortSizeDLT'
tidy(x, ...)

## S4 method for signature 'CohortSizeMin'
tidy(x, ...)

## S4 method for signature 'CohortSizeMax'
tidy(x, ...)
```

```
## S4 method for signature 'CohortSizeRange'
tidy(x, ...)

## S4 method for signature 'CohortSizeParts'
tidy(x, ...)

## S4 method for signature 'IncrementsMin'
tidy(x, ...)

## S4 method for signature 'IncrementsRelative'
tidy(x, ...)

## S4 method for signature 'IncrementsRelativeDLT'
tidy(x, ...)

## S4 method for signature 'IncrementsRelativeParts'
tidy(x, ...)

## S4 method for signature 'NextBestNCRM'
tidy(x, ...)

## S4 method for signature 'NextBestNCRMLoss'
tidy(x, ...)

## S4 method for signature 'DualDesign'
tidy(x, ...)

## S4 method for signature 'Samples'
tidy(x, ...)
```

Arguments

x	(CrmPackClass) the object to be tidied.
...	potentially used by class-specific methods.

Value

A (list of) tibble(s) representing the object in tidy form.

The tibble object.

The tibble object.

The tibble object.

The tibble object.

The tibble object.

The tibble object.

The tibble object.

The list of tibble objects.

The list of tibble objects.

Usage Notes

The prior observations are indicated by a Cohort value of 0 in the returned tibble.

Examples

```
CohortSizeConst(3) %>% tidy()
.DefaultData() %>% tidy()
.DefaultDataOrdinal() %>% tidy()
.DefaultDataGrouped() %>% tidy()
.DefaultDataDA() %>% tidy()
.DefaultData() %>% tidy()
.DefaultDataOrdinal() %>% tidy()
.DefaultDataGrouped() %>% tidy()
.DefaultDataDA() %>% tidy()
.DefaultData() %>% tidy()
.DefaultDataOrdinal() %>% tidy()
.DefaultDataGrouped() %>% tidy()
.DefaultDataDA() %>% tidy()
.DefaultData() %>% tidy()
.DefaultDataOrdinal() %>% tidy()
.DefaultDataGrouped() %>% tidy()
.DefaultDataDA() %>% tidy()
.DefaultData() %>% tidy()
.DefaultDataOrdinal() %>% tidy()
.DefaultDataGrouped() %>% tidy()
.DefaultDataDA() %>% tidy()
.DefaultData() %>% tidy()
.DefaultDataOrdinal() %>% tidy()
.DefaultDataGrouped() %>% tidy()
.DefaultDataDA() %>% tidy()
.DefaultData() %>% tidy()
.DefaultDataOrdinal() %>% tidy()
.DefaultDataGrouped() %>% tidy()
.DefaultDataDA() %>% tidy()
.DefaultSimulations() %>% tidy()
.DefaultLogisticIndepBeta() %>% tidy()
.DefaultEffloglog() %>% tidy()
CohortSizeRange(intervals = c(0, 20), cohort_size = c(1, 3)) %>% tidy()
.DefaultCohortSizeDLT() %>% tidy()
.DefaultCohortSizeMin() %>% tidy()
.DefaultCohortSizeMax() %>% tidy()
.DefaultCohortSizeRange() %>% tidy()
CohortSizeParts(cohort_sizes = c(1, 3)) %>% tidy()
.DefaultIncrementsMin() %>% tidy()
CohortSizeRange(intervals = c(0, 20), cohort_size = c(1, 3)) %>% tidy()
x <- .DefaultIncrementsRelativeDLT()
x %>% tidy()
.DefaultIncrementsRelativeParts() %>% tidy()
```

```
NextBestNCRM(
  target = c(0.2, 0.35),
  overdose = c(0.35, 1),
  max_overdose_prob = 0.25
) %>% tidy()
.DefaultNextBestNCRMLoss() %>% tidy()
.DefaultDualDesign() %>% tidy()
options <- McmcOptions(
  burnin = 100,
  step = 1,
  samples = 2000
)

emptydata <- Data(doseGrid = c(1, 3, 5, 10, 15, 20, 25, 40, 50, 80, 100))

model <- LogisticLogNormal(
  mean = c(-0.85, 1),
  cov =
    matrix(c(1, -0.5, -0.5, 1),
      nrow = 2
    ),
  ref_dose = 56
)

samples <- mcmc(emptydata, model, options)
samples %>% tidy()
```

`TITELogisticLogNormal-class` `TITELogisticLogNormal`

Description

[Stable]

`TITELogisticLogNormal` is the class for TITE-CRM based on a logistic regression model using a bivariate normal prior on the intercept and log slope parameters.

This class inherits from the `LogisticLogNormal`.

Usage

```
TITELogisticLogNormal(weight_method = "linear", ...)

.DefaultTITELogisticLogNormal()
```

Arguments

`weight_method`	(`string`) the weight function method: either linear or adaptive. This was used in Liu, Yin and Yuan's paper.

... Arguments passed on to LogisticLogNormal

mean (numeric)
: the prior mean vector.

cov (matrix)
: the prior covariance matrix. The precision matrix prec is internally calculated as an inverse of cov.

ref_dose (number)
: the reference dose $x*$ (strictly positive number).

Slots

weight_method (string)
: the weight function method: either linear or adaptive. This was used in Liu, Yin and Yuan's paper.

Note

Typically, end users will not use the .DefaultTITELogisticLogNormal() function.

See Also

DALogisticLogNormal.

Examples

```
my_model <- TITELogisticLogNormal(
  mean = c(0, 1),
  cov = diag(2),
  ref_dose = 1,
  weight_method = "linear"
)

my_model1 <- TITELogisticLogNormal(
  mean = c(0, 1),
  cov = diag(2),
  ref_dose = 1,
  weight_method = "adaptive"
)
```

---

update,Data-method *Updating* Data *Objects*

Description

[Stable]

A method that updates existing Data object with new data.

Usage

```
## S4 method for signature 'Data'
update(
  object,
  x,
  y,
  ID = length(object@ID) + seq_along(y),
  new_cohort = TRUE,
  check = TRUE,
  ...
)
```

Arguments

`object`	(`Data`) object you want to update.
`x`	(`number`) the dose level (one level only!).
`y`	(`integer`) the DLT vector (0/1 vector) for all patients in this cohort. You can also supply `numeric` vectors, but these will then be converted to `integer` internally.
`ID`	(`integer`) the patient IDs. You can also supply `numeric` vectors, but these will then be converted to `integer` internally.
`new_cohort`	(`flag`) if `TRUE` (default) the new data are assigned to a new cohort.
`check`	(`flag`) whether the validation of the updated object should be conducted. See details below.
`...`	not used.

Details

The current implementation of this `update` method allows for updating the `Data` class object by adding a single dose level `x` only. However, there might be some use cases where the new cohort to be added contains a placebo and active dose. Hence, such update would need to be performed iteratively by calling the `update` method twice. For example, in the first call a user can add a placebo, and then in the second call, an active dose. Since having a cohort with placebo only is not allowed, the `update` method would normally throw the error when attempting to add a placebo in the first call. To allow for such updates, the `check` parameter should be then set to `FALSE` for that first call.

Value

The new, updated `Data` object.

Examples

```
# Create some data of class 'Data'.
my_data <- Data(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)

# Update the data with a new cohort.
my_data1 <- update(my_data, x = 20, y = c(0L, 1L, 1L))
my_data1
```

---

update,DataDA-method *Updating* DataDA *Objects*

Description

[Stable]

A method that updates existing DataDA object with new data.

Usage

```
## S4 method for signature 'DataDA'
update(object, u, t0, trialtime, y, ..., check = TRUE)
```

Arguments

object	(DataDA) object you want to update.
u	(numeric) the new DLT free survival times for all patients, i.e. for existing patients in the object as well as for new patients.
t0	(numeric) the time that each patient starts DLT observation window. This parameter covers all patients, i.e. existing patients in the object as well as for new patients.
trialtime	(number) current time in the trial, i.e. a followup time.
y	(numeric) the new DLTs for all patients, i.e. for existing patients in the object as well as for new patients.
...	further arguments passed to Data update method update-Data. These are used when there are new patients to be added to the cohort.
check	(flag) whether the validation of the updated object should be conducted. See help for update-Data for more details on the use case of this parameter.

Value

The new, updated DataDA object.

Note

This function is capable of not only adding new patients but also updates existing ones with respect to y, t0, u slots.

Examples

```
# Create an object of class 'DataDA'.
my_data <- DataDA(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 1, 1, 0, 0, 1, 0),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2)),
  u = c(42, 30, 15, 5, 20, 25, 30, 60),
  t0 = c(0, 15, 30, 40, 55, 70, 75, 85),
  Tmax = 60
)

# Update the data.
my_data1 <- update(
  object = my_data,
  y = c(my_data@y, 0), # The 'y' will be updated according to 'u'.
  u = c(my_data@u, 20),
  t0 = c(my_data@t0, 95),
  x = 20,
  trialtime = 120 # This is the global timeline for a trial.
)
my_data1
```

---

update,DataDual-method

Updating DataDual *Objects*

Description

[Stable]

A method that updates existing DataDual object with new data.

Usage

```
## S4 method for signature 'DataDual'
update(object, w, ..., check = TRUE)
```

Arguments

object	(DataDual) object you want to update.
w	(numeric) the continuous vector of biomarker values for all the patients in this update.
...	further arguments passed to Data update method update-Data.
check	(flag) whether the validation of the updated object should be conducted. See help for update-Data for more details on the use case of this parameter.

Value

The new, updated DataDual object.

Examples

```
# Create some data of class 'DataDual'.
my_data <- DataDual(
  x = c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
  y = c(0, 0, 0, 0, 0, 0, 1, 0),
  w = rnorm(8),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2))
)

# Update the data with a new cohort.
my_data1 <- update(my_data, x = 20, y = c(0, 1, 1), w = c(0.4, 1.2, 2.2))
my_data1
```

---

update,DataOrdinal-method

Updating DataOrdinal *Objects*

Description

[Experimental]

A method that updates existing DataOrdinal object with new data.

Usage

```
## S4 method for signature 'DataOrdinal'
update(
  object,
  x,
  y,
  ID = length(object@ID) + seq_along(y),
  new_cohort = TRUE,
```

```
  check = TRUE,
  ...
)
```

Arguments

object	(DataOrdinal) object you want to update.
x	(number) the dose level (one level only!).
y	(integer) the vector of toxicity grades (0, 1, 2, ...) for all patients in this cohort. You can also supply numeric vectors, but these will then be converted to integer internally.
ID	(integer) the patient IDs. You can also supply numeric vectors, but these will then be converted to integer internally.
new_cohort	(flag) if TRUE (default) the new data are assigned to a new cohort.
check	(flag) whether the validation of the updated object should be conducted. See Details below.
...	not used.

Details

The current implementation of this update method allows for updating the DataOrdinal class object by adding a single dose level x only. However, there might be some use cases where the new cohort to be added contains a placebo and active dose. Hence, such update would need to be performed iteratively by calling the update method twice. For example, in the first call a user can add a placebo, and then in the second call, an active dose. Since having a cohort with placebo only is not allowed, the update method would normally throw the error when attempting to add a placebo in the first call. To allow for such updates, the check parameter should be then set to FALSE for that first call.

Value

The new, updated DataOrdinal object.

Examples

```
data <- DataOrdinal(
  x = c(10, 20, 30, 40, 50, 50, 50, 60, 60, 60),
  y = as.integer(c(0, 0, 0, 0, 0, 1, 0, 0, 1, 2)),
  ID = 1L:10L,
  cohort = as.integer(c(1:4, 5, 5, 5, 6, 6, 6)),
  doseGrid = c(seq(from = 10, to = 100, by = 10)),
  yCategories = c("No tox" = 0L, "Sub-tox AE" = 1L, "DLT" = 2L),
```

```
  placebo = FALSE
)

update(data, x = 70, y = c(1L, 2L, 1L))
```

---

update,DataParts-method

Updating DataParts *Objects*

Description

[Stable]

A method that updates existing DataParts object with new data.

Usage

```
## S4 method for signature 'DataParts'
update(object, x, y, ..., check = TRUE)
```

Arguments

object	(DataParts) object you want to update.
x	(number) the dose level (one level only!).
y	(integer) the DLT vector (0/1 vector) for all patients in this cohort. You can also supply numeric vectors, but these will then be converted to integer internally.
...	further arguments passed to Data update method update-Data.
check	(flag) whether the validation of the updated object should be conducted. See help for update-Data for more details on the use case of this parameter.

Value

The new, updated DataParts object.

Examples

```
# Create an object of class 'DataParts'.
my_data <- DataParts(
  x = c(0.1, 0.5, 1.5),
  y = c(0, 0, 0),
  doseGrid = c(0.1, 0.5, 1.5, 3, 6, seq(from = 10, to = 80, by = 2)),
  part = c(1L, 1L, 1L),
  nextPart = 1L,
  part1Ladder = c(0.1, 0.5, 1.5, 3, 6, 10)
```

```
)

# Update the data with a new cohort.
# Note that since we reached the last level from 'part1Ladder'
# then the 'nextPart' is switched from '1' to '2'.
my_data1 <- update(my_data, x = 10, y = 0L)
my_data1
```

---

update,ModelPseudo-method

Update method for the ModelPseudo model class. This is a method to update the model class slots (estimates, parameters, variables and etc.), when the new data (e.g. new observations of responses) are available. This method is mostly used to obtain new modal estimates for pseudo model parameters.

Description

Update method for the ModelPseudo model class. This is a method to update the model class slots (estimates, parameters, variables and etc.), when the new data (e.g. new observations of responses) are available. This method is mostly used to obtain new modal estimates for pseudo model parameters.

Usage

```
## S4 method for signature 'ModelPseudo'
update(object, data, ...)
```

Arguments

object	(ModelPseudo) the model to update.
data	(Data) all currently available of data.
...	not used.

Value

the new ModelPseudo class object.

Examples

```
# Update the 'LogisticIndepBeta' model with new data.
empty_data <- Data(doseGrid = seq(25, 300, 25))

my_model_lib <- LogisticIndepBeta(
  binDLE = c(1.05, 1.8),
  DLEweights = c(3, 3),
```

```
  DLEdose = c(25, 300),
  data = empty_data
)

# Then, we have some new observations data.
data <- Data(
  x = c(25, 50, 50, 75, 100, 100, 225, 300),
  y = c(0, 0, 0, 0, 1, 1, 1, 1),
  ID = 1:8,
  cohort = c(1L, 2L, 2L, 3L, 4L, 4L, 5L, 6L),
  doseGrid = empty_data@doseGrid
)

# Update the model to get new estimates.
new_model_lib <- update(object = my_model_lib, data = data)

# Update the 'Effloglog' model with new data.
empty_data_dual <- DataDual(doseGrid = seq(25, 300, 25), placebo = FALSE)

my_model_eff <- Effloglog(
  eff = c(1.223, 2.513),
  eff_dose = c(25, 300),
  nu = c(a = 1, b = 0.025),
  data = empty_data_dual,
  const = 0
)

# Data with new observations data.
my_data_dual <- DataDual(
  x = c(25, 50, 50, 75, 100, 100, 225, 300),
  y = c(0, 0, 0, 0, 1, 1, 1, 1),
  w = c(0.31, 0.42, 0.59, 0.45, 0.6, 0.7, 0.6, 0.52),
  ID = 1:8,
  cohort = c(1L, 2L, 2L, 3L, 4L, 4L, 5L, 6L),
  doseGrid = empty_data_dual@doseGrid
)

# Update the model to get new estimates.
new_model_eff <- update(object = my_model_eff, data = my_data_dual)
```

`Validate` `Validate`

Description

[Stable]

The `Validate` class is a Reference Class to help programming validation for new S4 classes.

Details

Starting from an empty `msg` vector, with each check that is returning `FALSE` the vector gets a new element - the string explaining the failure of the validation.

Fields

`msg` (`character`)
: the cumulative messages.

Methods

`check(test, string = "")` Check whether the `test` is `TRUE`; if so, return `NULL`. Otherwise, add the `string` message into the cumulative messages vector `msg`.

`result()` Return either cumulative messages vector `msg` (which contains the error messages from all the checks), or `NULL`, if `msg` is empty (i.e. all the checks were successful).

---

v_cohort_size *Internal Helper Functions for Validation of* `CohortSize` *Objects*

Description

[Stable]

These functions are only used internally to validate the format of an input `CohortSize` or inherited classes and therefore not exported.

Usage

```
v_cohort_size_range(object)

v_cohort_size_dlt(object)

v_cohort_size_const(object)

v_cohort_size_parts(object)

v_cohort_size_max(object)
```

Arguments

`object`	(`CohortSize`) object to validate.

Value

A `character` vector with the validation failure messages, or `TRUE` in case validation passes.

Functions

- v_cohort_size_range(): validates that the CohortSizeRange object contains valid intervals and cohort_size slots.
- v_cohort_size_dlt(): validates that the CohortSizeDLT object contains valid intervals and cohort_size slots.
- v_cohort_size_const(): validates that the CohortSizeConst object contains valid size slot.
- v_cohort_size_parts(): validates that the CohortSizeParts object contains valid sizes slot.
- v_cohort_size_max(): validates that the CohortSizeMax object contains valid cohort_sizes slot.

---

v_data_objects *Internal Helper Functions for Validation of* GeneralData *Objects*

Description

[Stable]

These functions are only used internally to validate the format of an input GeneralData or inherited classes and therefore not exported.

Usage

```
v_general_data(object)

h_doses_unique_per_cohort(dose, cohort)

v_data(object)

v_data_dual(object)

v_data_parts(object)

v_data_mixture(object)

v_data_da(object)

v_data_ordinal(object)

v_data_grouped(object)
```

Arguments

object	(GeneralData) object to validate.
dose	(numeric) dose values.
cohort	(integer) cohort indices parallel to doses.

Value

A character vector with the validation failure messages, or TRUE in case validation passes.

TRUE if dose is unique per cohort, otherwise FALSE.

Functions

- v_general_data(): validates that the GeneralData object contains unique ID, non-negative cohort indices and ID and cohort vectors are of the same length nObs.
- h_doses_unique_per_cohort(): helper function which verifies whether the dose values are unique in each and every different cohort.
- v_data(): validates that the Data object contains valid elements with respect to their types, dependency and length.
- v_data_dual(): validates that the DataDual object contains valid biomarker vector with respect to its type and the length.
- v_data_parts(): validates that the DataParts object contains valid elements with respect to their types, dependency and length.
- v_data_mixture(): validates that the DataMixture object contains valid elements with respect to their types, dependency and length.
- v_data_da(): validates that the DataDA object contains valid elements with respect to their types, dependency and length.
- v_data_ordinal(): validates that the DataOrdinal object contains valid elements with respect to their types, dependency and length.
- v_data_grouped(): validates that the DataGrouped object contains valid group information.

---

v_design *Internal Helper Functions for Validation of* RuleDesign *Objects*

Description

[Stable]

These functions are only used internally to validate the format of an input RuleDesign or inherited classes and therefore not exported.

[Experimental]

These functions are only used internally to validate the format of an input RuleDesignOrdinal or inherited classes and therefore not exported.

Usage

```
v_rule_design(object)

v_rule_design_ordinal(object)

v_design_grouped(object)
```

Arguments

object	(RuleDesignOrdinal) object to validate.

Value

A character vector with the validation failure messages, or TRUE in case validation passes.

A character vector with the validation failure messages, or TRUE in case validation passes.

Functions

- v_rule_design(): validates that the RuleDesign object contains valid startingDose.
- v_rule_design_ordinal(): validates that the RuleDesignOrdinal object contains valid starting_dose.
- v_design_grouped(): validates that the DesignGrouped object contains valid flags.

---

v_general_simulations *Internal Helper Functions for Validation of GeneralSimulations Objects*

Description

[Stable]

These functions are only used internally to validate the format of an input GeneralSimulations or inherited classes and therefore not exported.

Usage

```
v_general_simulations(object)

v_simulations(object)

v_dual_simulations(object)

v_da_simulations(object)
```

Arguments

object	(GeneralSimulations) object to validate.

Value

A character vector with the validation failure messages, or TRUE in case validation passes.

Functions

- v_general_simulations(): validates that the GeneralSimulations object contains valid data object and valid dose simulations.
- v_simulations(): validates that the Simulations object contains valid object fit, stop_reasons, stop_report, and additional_stats compared to the general class GeneralSimulations.
- v_dual_simulations(): validates that the DualSimulations object and capture the dose-biomarker fits, and the sigma2W and rho estimates.
- v_da_simulations(): validates that the DASimulations object contains valid trialduration the vector of trial duration values for all simulations.

---

v_increments *Internal Helper Functions for Validation of* Increments *Objects*

Description

[Stable]

These functions are only used internally to validate the format of an input Increments or inherited classes and therefore not exported.

Usage

```
v_increments_relative(object)

v_increments_relative_parts(object)

v_increments_relative_dlt(object)

v_increments_dose_levels(object)

v_increments_hsr_beta(object)

v_increments_min(object)

v_increments_ordinal(object)

v_cohort_size_ordinal(object)
```

Arguments

object	(Increments) object to validate.

Value

A character vector with the validation failure messages, or TRUE in case validation passes.

Functions

- v_increments_relative(): validates that the IncrementsRelative object contains valid intervals and increments parameters.
- v_increments_relative_parts(): validates that the IncrementsRelativeParts object contains valid dlt_start and clean_start parameters.
- v_increments_relative_dlt(): validates that the IncrementsRelativeDLT object contains valid intervals and increments parameters.
- v_increments_dose_levels(): validates that the IncrementsDoseLevels object contains valid levels and basis_level option.
- v_increments_hsr_beta(): validates that the IncrementsHSRBeta object contains valid probability target, threshold and shape parameters.
- v_increments_min(): validates that the IncrementsMin object contains a list with Increments objects.
- v_increments_ordinal(): validates that the IncrementsOrdinal object contains valid grade and standard Increments rule.
- v_cohort_size_ordinal(): validates that the CohortSizeOrdinal object contains valid grade and standard CohortSize rule.

---

v_mcmcoptions_objects *Internal Helper Functions for Validation of* McmcOptions *Objects*

Description

[Stable]

These functions are only used internally to validate the format of an input McmcOptions or inherited classes and therefore not exported.

Usage

```
v_mcmc_options(object)
```

Arguments

object	(McmcOptions) object to validate.

Value

A `character` vector with the validation failure messages, or `TRUE` in case validation passes.

Functions

- `v_mcmc_options()`: validates that the `McmcOptions` object contains valid integer scalars `iterations`, `burnin` and `step` as well as proper parameters for Random Number Generator.

`v_model_objects` *Internal Helper Functions for Validation of* `GeneralModel` *and* `ModelPseudo` *Objects*

Description

[Stable]

These functions are only used internally to validate the format of an input `GeneralModel` and `ModelPseudo` or inherited classes and therefore are not exported.

Usage

```
v_general_model(object)

v_model_logistic_kadane(object)

v_model_logistic_kadane_beta_gamma(object)

v_model_logistic_normal_mix(object)

v_model_logistic_normal_fixed_mix(object)

v_model_logistic_log_normal_mix(object)

v_model_dual_endpoint(object)

v_model_dual_endpoint_rw(object)

v_model_dual_endpoint_beta(object)

v_model_dual_endpoint_emax(object)

v_model_logistic_indep_beta(object)

v_model_eff_log_log(object)

v_model_eff_flexi(object)
```

```
v_model_da_logistic_log_normal(object)

v_model_tite_logistic_log_normal(object)

v_model_one_par_exp_normal_prior(object)

v_model_one_par_exp_prior(object)

v_logisticlognormalordinal(object)
```

Arguments

object	(GeneralModel) or (ModelPseudo) object to validate.

Value

A character vector with the validation failure messages, or TRUE in case validation passes.

Functions

- v_general_model(): validates that the names of the arguments in init function are included in datanames or datanames_prior slots.
- v_model_logistic_kadane(): validates that the logistic Kadane model parameters are valid.
- v_model_logistic_kadane_beta_gamma(): validates that the logistic Kadane model parameters with a beta and gamma prior are valid.
- v_model_logistic_normal_mix(): validates that weightpar is valid.
- v_model_logistic_normal_fixed_mix(): validates that component is a list with valid ModelParamsNormal objects as well as weights are correct.
- v_model_logistic_log_normal_mix(): validates that share_weight represents probability.
- v_model_dual_endpoint(): validates that DualEndpoint class slots are valid.
- v_model_dual_endpoint_rw(): validates that DualEndpointRW class slots are valid.
- v_model_dual_endpoint_beta(): validates that DualEndpointBeta class slots are valid.
- v_model_dual_endpoint_emax(): validates that DualEndpointEmax class slots are valid.
- v_model_logistic_indep_beta(): validates that LogisticIndepBeta class slots are valid.
- v_model_eff_log_log(): validates that Effloglog class slots are valid.
- v_model_eff_flexi(): validates that EffFlexi class slots are valid.
- v_model_da_logistic_log_normal(): validates that DALogisticLogNormal class slots are valid.
- v_model_tite_logistic_log_normal(): validates that TITELogisticLogNormal class slots are valid.
- v_model_one_par_exp_normal_prior(): validates that OneParLogNormalPrior class slots are valid.

- `v_model_one_par_exp_prior()`: validates that `OneParExpPrior` class slots are valid.
- `v_logisticlognormalordinal()`: confirms that cov is diagonal

`v_model_params` *Internal Helper Functions for Validation of Model Parameters Objects*

Description

[Experimental]

These functions are only used internally to validate the format of an object with model parameters or inherited classes and therefore not exported.

Usage

```
v_model_params_normal(object)
```

Arguments

`object`	(`ModelParamsNormal`) multivariate normal parameters object to validate.

Value

A `character` vector with the validation failure messages, or `TRUE` in case validation passes.

Functions

- `v_model_params_normal()`: a helper function that validates multivariate normal parameters.

`v_next_best` *Internal Helper Functions for Validation of* `NextBest` *Objects*

Description

[Stable]

These functions are only used internally to validate the format of an input `NextBest` or inherited classes and therefore not exported.

Usage

```
v_next_best_mtd(object)

v_next_best_ncrm(object)

v_next_best_ncrm_loss(object)

v_next_best_dual_endpoint(object)

v_next_best_min_dist(object)

v_next_best_inf_theory(object)

v_next_best_td(object)

v_next_best_td_samples(object)

v_next_best_max_gain_samples(object)

v_next_best_prob_mtd_lte(object)

v_next_best_prob_mtd_min_dist(object)

v_next_best_ordinal(object)
```

Arguments

object	(NextBest) object to validate.

Value

A character vector with the validation failure messages, or TRUE in case validation passes.

Functions

- v_next_best_mtd(): validates that the NextBestMTD object contains valid target probability and derive function.
- v_next_best_ncrm(): validates that the NextBestNCRM object contains valid target probability, overdose and max_overdose_prob probability ranges.
- v_next_best_ncrm_loss(): validates that the NextBestNCRMLoss object contains valid objects.
- v_next_best_dual_endpoint(): validates that the NextBestDualEndpoint object contains valid probability objects.
- v_next_best_min_dist(): validates that the NextBestMinDist object contains valid target object.

- `v_next_best_inf_theory()`: validates that the `NextBestInfTheory` object contains valid `target` and `asymmetry` objects.
- `v_next_best_td()`: validates that the `NextBestTD` object contains valid `prob_target_drt` and `prob_target_eot` probabilities.
- `v_next_best_td_samples()`: validates that the `NextBestTDsamples` object contains valid `derive` function.
- `v_next_best_max_gain_samples()`: validates that the `NextBestMaxGainSamples` object contains valid `derive` and `mg_derive` functions.
- `v_next_best_prob_mtd_lte()`: validates that the `NextBestProbMTDLTE` object contains valid `target` probability and `method` string value.
- `v_next_best_prob_mtd_min_dist()`: validates that the `NextBestProbMTDMinDist` object contains valid `target` probability and `method` string value.
- `v_next_best_ordinal()`: validates that the `NextBestOrdinal` object contains valid `grade` and standard `NextBest` rule.

`v_pseudo_simulations` *Internal Helper Functions for Validation of* `PseudoSimulations` *Objects*

Description

[Stable]

These functions are only used internally to validate the format of an input `PseudoSimulations` or inherited classes and therefore not exported.

Usage

```
v_pseudo_simulations(object)

v_pseudo_dual_simulations(object)

v_pseudo_dual_flex_simulations(object)
```

Arguments

`object`	(`PseudoSimulations`) object to validate.

Value

A `character` vector with the validation failure messages, or `TRUE` in case validation passes.

Functions

- v_pseudo_simulations(): validates that the PseudoSimulations object contains valid fit, FinalTDtargetEndOfTrialEstimates , FinalTDtargetDuringTrialAtDoseGrid,FinalTDtargetEndOfTrialAtDo , FinalTDEOTCIs, FinalTDEOTRatios, FinalCIs, FinalRatios, object and valid stopReasons simulations.
- v_pseudo_dual_simulations(): validates that the PseudoDualSimulations object contains valid fit_eff, final_gstar_estimates , final_gstar_at_dose_grid, final_gstar_cis , final_gstar_ratios, final_optimal_dose, final_optimal_dose_at_dose_grid object and valid sigma2_est simulations.
- v_pseudo_dual_flex_simulations(): validates that the PseudoDualFlexiSimulations object contains valid sigma2betaWest vector of the final posterior mean sigma2betaW estimates.FinalGstarEstimates , FinalGstarAtDoseGrid,

---

v_safety_window *Internal Helper Functions for Validation of* SafetyWindow *Objects*

Description

[Stable]

These functions are only used internally to validate the format of an input SafetyWindow or inherited classes and therefore not exported.

Usage

```
v_safety_window_size(object)

v_safety_window_const(object)
```

Arguments

object	(SafetyWindow) object to validate.

Value

A character vector with the validation failure messages, or TRUE in case validation passes.

Functions

- v_safety_window_size(): validates that the SafetyWindowSize object contains valid slots.
- v_safety_window_const(): validates that the SafetyWindowConst object contains valid slots.

v_samples_objects *Internal Helper Functions for Validation of* Samples *Objects*

Description

These functions are only used internally to validate the format of an input Samples or inherited classes and therefore not exported.

Usage

```
v_samples(object)
```

Arguments

object	(Samples) object to validate.

Value

A character vector with the validation failure messages, or TRUE in case validation passes.

Functions

- v_samples(): validates that the Samples object contains valid data slot.

v_stopping *Internal Helper Functions for Validation of* Stopping *Objects*

Description

[Stable]

These functions are only used internally to validate the format of an input Stopping or inherited classes and therefore not exported.

Usage

```
v_stopping_cohorts_near_dose(object)

v_stopping_patients_near_dose(object)

v_stopping_min_cohorts(object)

v_stopping_min_patients(object)

v_stopping_target_prob(object)
```

```
v_stopping_mtd_distribution(object)

v_stopping_mtd_cv(object)

v_stopping_target_biomarker(object)

v_stopping_list(object)

v_stopping_all(object)

v_stopping_tdci_ratio(object)
```

Arguments

object	(Stopping) object to validate.

Value

A character vector with the validation failure messages, or TRUE in case validation passes.

Functions

- v_stopping_cohorts_near_dose(): validates that the StoppingCohortsNearDose object contains valid nCohorts and percentage parameters.
- v_stopping_patients_near_dose(): validates that the StoppingPatientsNearDose object contains valid nPatients and percentage parameters.
- v_stopping_min_cohorts(): validates that the StoppingMinCohorts object contains valid nCohorts parameter.
- v_stopping_min_patients(): validates that the StoppingMinPatients object contains valid nPatients parameter.
- v_stopping_target_prob(): validates that the StoppingTargetProb object contains valid target and prob parameters.
- v_stopping_mtd_distribution(): validates that the StoppingMTDdistribution object contains valid target, thresh and prob parameters.
- v_stopping_mtd_cv(): validates that the StoppingMTDCV object contains valid probability target and percentage threshold.
- v_stopping_target_biomarker(): validates that the StoppingTargetBiomarker object contains valid target, is_relative and probslots.
- v_stopping_list(): validates that the StoppingList object contains valid stop_list, summary slots.
- v_stopping_all(): validates that the StoppingAll object contains valid stop_list slot.
- v_stopping_tdci_ratio(): validates that the StoppingTDCIRatio object contains valid target_ratio and prob_target slots.

windowLength *Determine the safety window length of the next cohort*

Description

This function determines the safety window length of the next cohort.

Usage

```
windowLength(safetyWindow, size, ...)

## S4 method for signature 'SafetyWindowSize'
windowLength(safetyWindow, size, data, ...)

## S4 method for signature 'SafetyWindowConst'
windowLength(safetyWindow, size, ...)
```

Arguments

safetyWindow	The rule, an object of class SafetyWindow
size	The next cohort size
...	additional arguments
data	The data input, an object of class DataDA

Value

the windowLength as a list of safety window parameters (gap, follow, follow_min)

Functions

- windowLength(SafetyWindowSize): Determine safety window length based on the cohort size
- windowLength(SafetyWindowConst): Constant safety window length

Examples

```
# nolint start

# Create the data
data <- DataDA(x=c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
               y=c(0, 0, 1, 1, 0, 0, 1, 0),
               doseGrid=
                 c(0.1, 0.5, 1.5, 3, 6,
                   seq(from=10, to=80, by=2)),
               u=c(42,30,15,5,20,25,30,60),
               t0=c(0,15,30,40,55,70,75,85),
               Tmax=60)
```

```
# Initialize the CRM model used to model the data
npiece_ <- 10
lambda_prior<-function(k){
  npiece_/(data@Tmax*(npiece_-k+0.5))
}

model<-DALogisticLogNormal(mean=c(-0.85,1),
                           cov=matrix(c(1,-0.5,-0.5,1),nrow=2),
                           ref_dose=56,
                           npiece=npiece_,
                  l=as.numeric(t(apply(as.matrix(c(1:npiece_),1,npiece_),2,lambda_prior))),
                           c_par=2)

# Set-up some MCMC parameters and generate samples from the posterior
options <- McmcOptions(burnin=100,
                       step=2,
                       samples=200)
set.seed(94)
samples <- mcmc(data, model, options)

# Define the rule for dose increments and calculate the maximum dose allowed
myIncrements <- IncrementsRelative(intervals=c(0, 20),
                                   increments=c(1, 0.33))
nextMaxDose <- maxDose(myIncrements,
                       data=data)

# Define the rule which will be used to select the next best dose
# based on the class 'NextBestNCRM'
myNextBest <- NextBestNCRM(target=c(0.2, 0.35),
                           overdose=c(0.35, 1),
                           max_overdose_prob=0.25)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
                               doselimit=nextMaxDose,
                               samples=samples, model=model, data=data)

# Define the rule which will be used to select the next cohort size
# based on the class 'CohortSizeConst'
mySize <- CohortSizeConst(size=3)

# Determine the cohort size for the next cohort
sizeRecommendation <- size(mySize, dose=doseRecommendation$value, data = data)

# Rule for the safety window length:
#   -having patientGap as (0,7,3,3,...) for cohort size <4
#   -and having patientGap as (0,9,5,5,...) for cohort size >=4
myWindowLength <- SafetyWindowSize(gap = list(c(7,3),c(9,5)),
                                   size = c(1,4),
                                   follow = 7,
                                   follow_min = 14)
```

```
# Determine the safety window parameters for the next cohort
windowLength(myWindowLength, size=sizeRecommendation)

# nolint end
# nolint start

# Create the data
data <- DataDA(x=c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
               y=c(0, 0, 1, 1, 0, 0, 1, 0),
               doseGrid=
                 c(0.1, 0.5, 1.5, 3, 6,
                   seq(from=10, to=80, by=2)),
               u=c(42,30,15,5,20,25,30,60),
               t0=c(0,15,30,40,55,70,75,85),
               Tmax=60)

# Initialize the CRM model used to model the data
npiece_ <- 10
lambda_prior<-function(k){
  npiece_/(data@Tmax*(npiece_-k+0.5))
}

model<-DALogisticLogNormal(mean=c(-0.85,1),
                           cov=matrix(c(1,-0.5,-0.5,1),nrow=2),
                           ref_dose=56,
                           npiece=npiece_,
                  l=as.numeric(t(apply(as.matrix(c(1:npiece_),1,npiece_),2,lambda_prior))),
                           c_par=2)

# Set-up some MCMC parameters and generate samples from the posterior
options <- McmcOptions(burnin=100,
                       step=2,
                       samples=200)
set.seed(94)
samples <- mcmc(data, model, options)

# Define the rule for dose increments and calculate the maximum dose allowed
myIncrements <- IncrementsRelative(intervals=c(0, 20),
                                   increments=c(1, 0.33))
nextMaxDose <- maxDose(myIncrements,
                       data=data)

# Define the rule which will be used to select the next best dose
# based on the class 'NextBestNCRM'
myNextBest <- NextBestNCRM(target=c(0.2, 0.35),
                           overdose=c(0.35, 1),
                           max_overdose_prob=0.25)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
                               doselimit=nextMaxDose,
                               samples=samples, model=model, data=data)
```

```
# Define the rule which will be used to select the next cohort size
# based on the class 'CohortSizeConst'
mySize <- CohortSizeConst(size=3)

# Determine the cohort size for the next cohort
sizeRecommendation <- size(mySize, dose=doseRecommendation$value, data = data)

# Rule for having safety window length with constant safety window parameters
myWindowLength <- SafetyWindowConst(gap = c(7,3),
                                    follow = 7,
                                    follow_min = 14)

# Determine the safety window parameters for the next cohort
windowLength(myWindowLength, size=sizeRecommendation)

# nolint end
```

`&,Stopping,Stopping-method`

The method combining two atomic stopping rules

Description

The method combining two atomic stopping rules

Usage

```
## S4 method for signature 'Stopping,Stopping'
e1 & e2
```

Arguments

e1	First Stopping object
e2	Second Stopping object

Value

The StoppingAll object

Examples

```
## Example of combining two atomic stopping rules with an AND ('&') operator

myStopping1 <- StoppingMinCohorts(nCohorts=3)
myStopping2 <- StoppingTargetProb(target=c(0.2, 0.35),
                                  prob=0.5)

myStopping <- myStopping1 & myStopping2
```

`&,Stopping,StoppingAll-method`
The method combining an atomic and a stopping list

Description

The method combining an atomic and a stopping list

Usage

```
## S4 method for signature 'Stopping,StoppingAll'
e1 & e2
```

Arguments

e1	Stopping object
e2	StoppingAll object

Value

The modified StoppingAll object

Examples

```
## Example of combining an atomic stopping rule with a list of stopping rules
## with an AND ('&') operator

myStopping1 <- StoppingMinCohorts(nCohorts=3)
myStopping2 <- StoppingTargetProb(target=c(0.2, 0.35),
                                  prob=0.5)

myStopping3 <- StoppingMinPatients(nPatients=20)

myStopping <-  myStopping3 & (myStopping1 | myStopping2 )
```

`&,StoppingAll,Stopping-method`
The method combining a stopping list and an atomic

Description

The method combining a stopping list and an atomic

Usage

```
## S4 method for signature 'StoppingAll,Stopping'
e1 & e2
```

Arguments

e1	StoppingAll object
e2	Stopping object

Value

The modified StoppingAll object

Examples

```
## Example of combining a list of stopping rules with an atomic stopping rule
## with an AND ('&') operator

myStopping1 <- StoppingMinCohorts(nCohorts=3)
myStopping2 <- StoppingTargetProb(target=c(0.2, 0.35),
                                  prob=0.5)

myStopping3 <- StoppingMinPatients(nPatients=20)

myStopping <- (myStopping1 | myStopping2 ) & myStopping3
```

Index