Package: crmPack (via r-universe)

September 25, 2024

License GPL (>= 2)

Title Object-Oriented Implementation of CRM Designs

LazyLoad yes

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.

Version 1.0.6

Copyright F. Hoffmann-La Roche Ltd

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

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

VignetteBuilder knitr

Depends R (>= 3.0.0), ggplot2 (>= 2.0.0), graphics

Imports methods, grid, gridExtra, GenSA, mvtnorm, parallel, rjags, utils, tools, MASS

Suggests ggmcmc, knitr, Rcpp, RcppArmadillo

Collate 'helpers.R' 'Data-class.R' 'Data-methods.R' 'Rules-class.R' 'Model-class.R' 'Design-class.R' 'writeModel.R' 'McmcOptions-methods.R' 'McmcOptions-class.R' 'Samples-class.R' 'mcmc.R' 'Simulations-class.R' 'Model-methods.R' 'Rules-methods.R' 'Design-methods.R' 'fromQuantiles.R' 'Samples-methods.R' 'Simulations-methods.R' 'crmPack-package.R'

RoxygenNote 7.2.1

Encoding UTF-8

NeedsCompilation no

Author Daniel Sabanes Bove [aut, cre], Wai Yin Yeung [aut], Giuseppe Palermo [aut], Thomas Jaki [aut]

Maintainer Daniel Sabanes Bove <daniel.sabanes_bove@rconis.com>

Repository CRAN

Date/Publication 2024-06-26 15:00:14 UTC

Contents

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

Description

Object-oriented implementation of CRM designs

Author(s)

Daniel Sabanes Bove <daniel.sabanes_bove@rconis.com>, Wai Yin Yeung <winnie.yeung@roche.com>, Giuseppe Palermo <giuseppe.palermo@roche.com>, Thomas Jaki <jaki.thomas@gmail.com>

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).

`AllModels-class` *Class for All models This is a class where all models inherit.*

Description

Class for All models This is a class where all models inherit.

Slots

`datanames` The names of all data slots that are used in all models. In particularly, those are also used in the `datamodel` and/or `priormodel` definition for `GeneralModel`.

See Also

`GeneralModel`, `ModelPseudo`

approximate *Approximate posterior with (log) normal distribution*

Description

It is recommended to use `set.seed` before, in order to be able to reproduce the resulting approximating model exactly.

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,
  ...
)
```

Arguments

`object`	the `Samples` object
`model`	the `Model` 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 and plot)? (default)

Value

the approximation model

Functions

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

Examples

```
# 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),
                           refDose = 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)
posterior <- approximate(object = samples,
                         model = model,
                         data = data,
                         logNormal=TRUE,
                         control = list(threshold.stop = 0.1,
                                        max.time = 1,
                                        maxit = 1))
```

`as.list,GeneralData-method`

as.list method for the "GeneralData" class

Description

as.list method for the "GeneralData" class

Usage

```
## S4 method for signature 'GeneralData'
as.list(x, ...)
```

Arguments

x	the GeneralData object we want to convert
...	unused

Value

a list of all slots in x

Examples

```
# Create some data of class 'Data'
myData <- 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)))

# Converting Data object to list
as.list(myData)
```

---

biomLevel — *Compute the biomarker level for a given dose, given model and samples*

Description

Compute the biomarker level for a given dose, given model and samples

Usage

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

## S4 method for signature 'numeric,DualEndpoint,Samples'
biomLevel(dose, model, samples, xLevel, ...)
```

Arguments

dose	the dose
model	the DualEndpoint object
samples	the Samples object
...	unused
xLevel	the grid index of dose

Functions

- biomLevel(dose = numeric, model = DualEndpoint, samples = Samples): Here it is very easy, we just return the corresponding column (index xLevel) of the biomarker samples matrix, since we save that in the samples

Examples

```
# 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(mu = c(0, 1),
                        Sigma = 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),
                        smooth = "RW1")

# 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)

# Obtain the biomarker level for a given dose, given model and samples
biomLevel(dose = 0.5,
          model = model,
          samples = samples,
          xLevel = 2)
```

`CohortSize-class` *The virtual class for cohort sizes*

Description

The virtual class for cohort sizes

See Also

`CohortSizeMax`, `CohortSizeMin`, `CohortSizeRange`, `CohortSizeDLT`, `CohortSizeConst`, `CohortSizeParts`

CohortSizeConst *Initialization function for "CohortSizeConst"*

Description

Initialization function for "CohortSizeConst"

Usage

```
CohortSizeConst(size)
```

Arguments

size	see CohortSizeConst

Value

the CohortSizeConst object

CohortSizeConst-class *Constant cohort size*

Description

This class is used when the cohort size should be kept constant.

Slots

size the constant integer size

Examples

```
# This is to have along the study a constant cohort size of 3
mySize <- CohortSizeConst(size=3)
```

CohortSizeDLT *Initialization function for "CohortSizeDLT"*

Description

Initialization function for "CohortSizeDLT"

Usage

```
CohortSizeDLT(DLTintervals, cohortSize)
```

Arguments

DLTintervals	see CohortSizeDLT
cohortSize	see CohortSizeDLT

Value

the CohortSizeDLT object

CohortSizeDLT-class *Cohort size based on number of DLTs*

Description

Cohort size based on number of DLTs

Slots

DLTintervals an integer vector with the left bounds of the relevant DLT intervals

cohortSize an integer vector of the same length with the cohort sizes in the DLTintervals

Examples

```
# As example, 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

mySize <- CohortSizeDLT(DLTintervals = c(0, 1),
                        cohortSize = c(1, 3))
```

CohortSizeMax *Initialization function for "CohortSizeMax"*

Description

Initialization function for "CohortSizeMax"

Usage

```
CohortSizeMax(cohortSizeList)
```

Arguments

`cohortSizeList` see `CohortSizeMax`

Value

the `CohortSizeMax` object

CohortSizeMax-class *Size based on maximum of multiple cohort size rules*

Description

This class can be used to combine multiple cohort size rules with the MAX operation.

Details

`cohortSizeList` contains all cohort size rules, which are again objects of class `CohortSize`. The maximum of these individual cohort sizes is taken to give the final cohort size.

Slots

`cohortSizeList` list of cohort size rules

Examples

```
# 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),
                           cohortSize = 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(DLTintervals=c(0, 1),
                         cohortSize=c(1, 3))
```

```
# Create a list of cohort size rules of class 'CohortSizeMax' which will then be
# combined with the 'max' operation
mySize <- CohortSizeMax(cohortSizeList=list(mySize1,
                                            mySize2))
```

`CohortSizeMin` *Initialization function for "CohortSizeMin"*

Description

Initialization function for "CohortSizeMin"

Usage

```
CohortSizeMin(cohortSizeList)
```

Arguments

`cohortSizeList` see `CohortSizeMin`

Value

the `CohortSizeMin` object

`CohortSizeMin-class` *Size based on minimum of multiple cohort size rules*

Description

This class can be used to combine multiple cohort size rules with the MIN operation.

Details

`cohortSizeList` contains all cohort size rules, which are again objects of class `CohortSize`. The minimum of these individual cohort sizes is taken to give the final cohort size.

Slots

`cohortSizeList` list of cohort size rules

Examples

```
# 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),
                           cohortSize = 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(DLTintervals=c(0, 1),
                         cohortSize=c(1, 3))

# Create a list of cohort size rules of class 'CohortSizeMax' which will then be
# combined with the 'min' operation
mySize <- CohortSizeMin(cohortSizeList=list(mySize1,
                                            mySize2))
```

CohortSizeParts *Initialization function for "CohortSizeParts"*

Description

Initialization function for "CohortSizeParts"

Usage

```
CohortSizeParts(sizes)
```

Arguments

sizes	see CohortSizeParts

Value

the CohortSizeParts object

CohortSizeParts-class *Cohort size based on the parts*

Description

This class is used when the cohort size should change for the second part of the dose escalation. Only works in conjunction with DataParts objects.

Slots

`sizes` the two sizes for part 1 and part 2

Examples

```
mySize <- CohortSizeParts(sizes=c(1,3))
```

CohortSizeRange *Initialization function for "CohortSizeRange"*

Description

Initialization function for "CohortSizeRange"

Usage

```
CohortSizeRange(intervals, cohortSize)
```

Arguments

`intervals`	see `CohortSizeRange`
`cohortSize`	see `CohortSizeRange`

Value

the `CohortSizeRange` object

CohortSizeRange-class *Cohort size based on dose range*

Description

Cohort size based on dose range

Slots

`intervals` a vector with the left bounds of the relevant dose intervals

`cohortSize` an integer vector of the same length with the cohort sizes in the `intervals`

Examples

```
# As example, here is the rule for:
#   having cohort of size 1 for doses <30
#   and having cohort of size 3 for doses >=30

mySize <- CohortSizeRange(intervals=c(0, 30),
                          cohortSize=c(1, 3))
```

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>

`Data` *Initialization function for the "Data" class*

Description

This is the function for initializing a "Data" class object.

Usage

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

Arguments

`x`	the doses for the patients
`y`	the vector of toxicity events (0 or 1 integers). You can also normal numeric vectors, but these will then be converted to integers.
`ID`	unique patient IDs (integer vector)
`cohort`	the cohort indices (sorted values from 0, 1, 2, ...)
`doseGrid`	the vector of all possible doses
`placebo`	logical value: if TRUE the first dose level in the grid is considered as PLACEBO
`...`	not used

Details

Note that `ID` and `cohort` can be missing, then a warning will be issued and the variables will be filled with default IDs and best guesses, respectively.

Value

the initialized `Data` object

Data-class — *Class for the data input*

Description

This class inherits from GeneralData.

Slots

x the doses for the patients

y the vector of toxicity events (0 or 1 integers)

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

nGrid number of gridpoints

xLevel the levels for the doses the patients have been given

placebo logical value: if TRUE the first dose level in the grid is considered as PLACEBO

Examples

```
# create some data from the class 'Data'
myData <- 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)))
```

DataDual — *Initialization function for the "DataDual" class*

Description

This is the function for initializing a "DataDual" class object.

Usage

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

Arguments

w	the continuous vector of biomarker values
...	additional parameters from Data

Value

the initialized DataDual object

DataDual-class *Class for the dual endpoint data input*

Description

This is a subclass of Data, so contains all slots from Data, and in addition biomarker values.

Slots

w the continuous vector of biomarker values

Examples

```
# Create some data from the class 'DataDual'
myData <- 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)))
```

DataMixture *Initialization function for the "DataMixture" class*

Description

This is the function for initializing a "DataMixture" class object.

Usage

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

Arguments

xshare	see DataMixture
yshare	see DataMixture
...	additional arguments for the underlying Data slots

Value

the initialized DataMixture object

DataMixture-class *Class for the data with mixture sharing*

Description

Class for the data with mixture sharing

Slots

`xshare` the doses for the share patients

`yshare` the vector of toxicity events (0 or 1 integers) for the share patients

`nObsshare` number of share patients

See Also

`LogisticLogNormalMixture` for the explanation how to use this data class

Examples

```
## decide on the dose grid:
doseGrid <- 1:80

## and MCMC options:
options <- McmcOptions()

## the classic model would be:
model <- LogisticLogNormal(mean = c(-0.85, 1),
                           cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
                           refDose = 50)

nodata <- Data(doseGrid=doseGrid)

priorSamples <- mcmc(nodata, model, options)
plot(priorSamples, model, nodata)

## set up the mixture model and data share object:
modelShare <- LogisticLogNormalMixture(shareWeight=0.1,
                                       mean = c(-0.85, 1),
                                       cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
                                       refDose = 50)

nodataShare <- DataMixture(doseGrid=doseGrid,
                           xshare=
                             c(rep(10, 4),
                               rep(20, 4),
                               rep(40, 4)),
                           yshare=
                             c(rep(0L, 4),
                               rep(0L, 4),
```

```
                                rep(0L, 4)))

## now compare with the resulting prior model:
priorSamplesShare <- mcmc(nodataShare, modelShare, options)
plot(priorSamplesShare, modelShare, nodataShare)
```

`DataParts` *Initialization function for the "DataParts" class*

Description

This is the function for initializing a `DataParts` object.

Usage

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

Arguments

`part`	which part does each of the patients belong to?
`nextPart`	what is the part for the next cohort? (1 or 2)
`part1Ladder`	what is the escalation ladder for part 1?
`...`	additional parameters from `Data`

Value

the initialized `DataParts` object

`DataParts-class` *Class for the data with two study parts*

Description

This is a subclass of `Data`, so contains all slots from `Data`, and in addition information on the two study parts.

Slots

`part` integer vector; which part does each of the patients belong to?

`nextPart` integer; what is the part for the next cohort?

`part1Ladder` sorted numeric vector; what is the escalation ladder for part 1? This shall be a subset of the `doseGrid`.

Examples

```
# create an object of class 'DataParts'
myData <- 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))
```

`Design` *Initialization function for "Design"*

Description

Initialization function for "Design"

Usage

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

Arguments

`model`	see `Design`
`stopping`	see `Design`
`increments`	see `Design`
`PLcohortSize`	see `Design`
`...`	additional arguments for `RuleDesign`

Value

the `Design` object

`Design-class` *Class for the CRM design*

Description

In addition to the slots in the more simple `RuleDesign`, objects of this class contain:

Slots

`model` the model to be used, an object of class `Model`

`stopping` stopping rule(s) for the trial, an object of class `Stopping`

`increments` how to control increments between dose levels, an object of class `Increments`

`PLcohortSize` rules for the cohort sizes for placebo, if any planned an object of class `CohortSize` (defaults to constant 0 placebo patients)

Examples

```
# 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),
                           refDose=56)

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

# Choose the rule for the cohort-size
mySize1 <- CohortSizeRange(intervals=c(0, 30),
                           cohortSize=c(1, 3))
mySize2 <- CohortSizeDLT(DLTintervals=c(0, 1),
                         cohortSize=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,
                 cohortSize=mySize,
                 data=emptydata,
                 startingDose=3)
```

dose *Compute the doses for a given probability, given model and samples*

Description

Compute the doses for a given probability, given model and samples

Usage

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

## S4 method for signature 'numeric,Model,Samples'
dose(prob, model, samples, ...)

## S4 method for signature 'numeric,ModelTox,Samples'
dose(prob, model, samples, ...)

## S4 method for signature 'numeric,ModelTox,missing'
dose(prob, model, samples, ...)
```

Arguments

prob	the probability
model	the Model
samples	the Samples
...	unused

Functions

- dose(prob = numeric, model = ModelTox, samples = Samples): Compute the doses for a given probability, given Pseudo DLE model with samples
- dose(prob = numeric, model = ModelTox, samples = missing): Compute the dose for a given probability and a given Pseudo DLE model without samples

Examples

```
# 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),
                            refDose=56)

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

# Posterior for the dose achieving Prob(DLE) = 0.45
TD45 <- dose(prob=0.45,model=model,samples=samples)


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

## Initialize a model 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)

options <- McmcOptions(burnin=100, step=2, samples=200)
DLEsamples <- mcmc(data=data,model=DLEmodel,options=options)

TD45 <- dose(prob=0.45, model = DLEmodel,samples = DLEsamples)


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

## Initialize a model 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)

TD45 <- dose(prob=0.45, model = DLEmodel)
```

DualDesign *Initialization function for "DualDesign"*

Description

Initialization function for "DualDesign"

Usage

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

Arguments

model	see DualDesign
data	see DualDesign
...	additional arguments for Design

Value

the DualDesign object

DualDesign-class *Class for the dual-endpoint CRM design*

Description

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

Slots

model the model to be used, an object of class DualEndpoint

data what is the dose grid, any previous data, etc., contained in an object of class DataDual

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

Examples

```
# 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(mu = c(0, 1),
                        Sigma = 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),
                        smooth="RW1")

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

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

```
mySize1 <- CohortSizeRange(intervals=c(0, 30),
                           cohortSize=c(1, 3))
mySize2 <- CohortSizeDLT(DLTintervals=c(0, 1),
                         cohortSize=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)

# 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,
                     cohortSize = mySize,
                     startingDose = 3)
```

DualEndpoint *Initialization function for the "DualEndpoint" class*

Description

Initialization function for the "DualEndpoint" class

Usage

```
DualEndpoint(mu, Sigma, refDose = 1, useLogDose = FALSE, sigma2W, rho)
```

Arguments

mu	see DualEndpoint
Sigma	see DualEndpoint
refDose	see DualEndpoint (default: 1)
useLogDose	see DualEndpoint (default: FALSE)
sigma2W	see DualEndpoint
rho	see DualEndpoint

Value

the DualEndpoint object

`DualEndpoint-class` *General class for the dual endpoint model*

Description

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 subclasses of this class detail how the dose-biomarker relationship is parametrized and are those to be used. This class here shall contain all the common features to reduce duplicate code. (However, this class must not be virtual, because we need to create objects of it during the construction of subclass objects.)

Details

Currently a probit regression model

$$probit[p(x)] = \beta_{Z1} + \beta_{Z2} \cdot x/x^*$$

or

$$probit[p(x)] = \beta_{Z1} + \beta_{Z2} \cdot \log(x/x^*)$$

in case that the option `useLogDose` is `TRUE`. Here $p(x)$ is the probability of observing a DLT for a given dose x, Φ is the standard normal cdf, and x^* is the reference dose.

The prior is

$$(\beta_{Z1}, log(\beta_{Z2})) \sim Normal(\mu, \Sigma)$$

.

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

$$w(x) \sim Normal(f(x), \sigma_W^2)$$

and $f(x)$ is a function of the dose x, which is further specified in the subclasses. The biomarker variance σ_W^2 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 ρ 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 Hive page for more details on the model and the example vignette by typing `crmPackExample()` for a full example.

Slots

`mu` For the probit toxicity model, `mu` contains the prior mean vector

`Sigma` For the probit toxicity model, contains the prior covariance matrix

`refDose` For the probit toxicity model, the reference dose

`useLogDose` For the probit toxicity model, whether a log transformation of the (standardized) dose should be used?

`sigma2W` Either a fixed value for the biomarker variance, or a vector with elements `a` and `b` for the inverse-gamma prior parameters.

`rho` Either a fixed value for the correlation (between -1 and 1), or a vector with elements `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.

`useFixed` a list with logical value for each of the parameters indicating whether a fixed value is used or not; this slot is needed for internal purposes and not to be touched by the user.

See Also

Current subclasses: `DualEndpointRW`, `DualEndpointBeta`

`DualEndpointBeta` *Initialization function for the "DualEndpointBeta" class*

Description

Initialization function for the "DualEndpointBeta" class

Usage

```
DualEndpointBeta(E0, Emax, delta1, mode, refDoseBeta, ...)
```

Arguments

`E0`	see `DualEndpointBeta`
`Emax`	see `DualEndpointBeta`
`delta1`	see `DualEndpointBeta`
`mode`	see `DualEndpointBeta`
`refDoseBeta`	see `DualEndpointBeta`
`...`	additional parameters, see `DualEndpoint`

Value

the `DualEndpointBeta` object

DualEndpointBeta-class

Dual endpoint model with beta function for dose-biomarker relationship

Description

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

Details

$$f(x) = E_0 + (E_{max} - E_0) * Beta(\delta_1, \delta_2) * (x/x^*)^{\delta_1} * (1 - x/x^*)^{\delta_2}$$

where x^* is the maximum dose (end of the dose range to be considered), δ_1 and δ_2 are the two beta parameters, and E_0 and E_{max} are the minimum and maximum levels, respectively. For ease of interpretation, we parametrize with δ_1 and the mode of the curve instead, where

$$mode = \delta_1/(\delta_1 + \delta_2),$$

and multiplying this with 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` either a fixed number or the two uniform distribution parameters

`Emax` either a fixed number or the two uniform distribution parameters

`delta1` either a fixed number or the two uniform distribution parameters

`mode` either a fixed number or the two uniform distribution parameters

`refDoseBeta` the reference dose x^* (note that this is different from the `refDose` in the inherited DualEndpoint model)

Examples

```
model <- DualEndpointBeta(E0 = c(0, 100),
                          Emax = c(0, 500),
                          delta1 = c(0, 5),
                          mode = c(1, 15),
                          refDose=10,
                          useLogDose=TRUE,
                          refDoseBeta = 1000,
                          mu = c(0, 1),
                          Sigma = matrix(c(1, 0, 0, 1), nrow=2),
                          sigma2W = c(a=0.1, b=0.1),
```

```
                    rho = c(a=1, b=1))
```

`DualEndpointEmax` *Initialization function for the "DualEndpointEmax" class*

Description

Initialization function for the "DualEndpointEmax" class

Usage

```
DualEndpointEmax(E0, Emax, ED50, refDoseEmax, ...)
```

Arguments

`E0`	see `DualEndpointEmax`
`Emax`	see `DualEndpointEmax`
`ED50`	see `DualEndpointEmax`
`refDoseEmax`	see `DualEndpointEmax`
`...`	additional parameters, see `DualEndpoint`

Value

the `DualEndpointEmax` object

`DualEndpointEmax-class` *Dual endpoint model with emax function for dose-biomarker relationship*

Description

This class extends the `DualEndpoint` class. Here the dose-biomarker relationship $f(x)$ is modelled by a parametric EMAX function:

Details

$$f(x) = E_0 + \frac{(E_{max} - E_0) * (x/x^*)}{ED_{50} + (x/x^*)}$$

where x^* is a reference dose, E_0 and E_{max} are the minimum and maximum levels for the biomarker and ED_{50} is the dose achieving half of the maximum effect $0.5 * E_{max}$.

All parameters can currently be assigned uniform distributions or be fixed in advance.

Slots

`E0` either a fixed number or the two uniform distribution parameters

`Emax` either a fixed number or the two uniform distribution parameters

`ED50` either a fixed number or the two uniform distribution parameters

`refDoseEmax` the reference dose x^*

Examples

```
model <- DualEndpointEmax(E0 = c(0, 100),
                          Emax = c(0, 500),
                          ED50 = c(10,200),
                          refDoseEmax = 1000,
                          mu = c(0, 1),
                          Sigma = matrix(c(1, 0, 0, 1), nrow=2),
                          sigma2W = c(a=0.1, b=0.1),
                          rho = c(a=1, b=1))
```

DualEndpointRW *Initialization function for the "DualEndpointRW" class*

Description

Initialization function for the "DualEndpointRW" class

Usage

```
DualEndpointRW(sigma2betaW, smooth = c("RW1", "RW2"), ...)
```

Arguments

`sigma2betaW`	see `DualEndpointRW`
`smooth`	either “RW1” (default) or “RW2”, for specifying the random walk prior on the biomarker level.
`...`	additional parameters, see `DualEndpoint`

Value

the `DualEndpointRW` object

DualEndpointRW-class *Dual endpoint model with RW prior for biomarker*

Description

This class extends the DualEndpoint class. Here the dose-biomarker relationship $f(x)$ is modelled by a non-parametric random-walk of first (RW1) or second order (RW2).

Details

That means, for the RW1 we assume

$$\beta_{W,i} - \beta_{W,i-1} \sim Normal(0, (x_i - x_{i-1})\sigma^2_{\beta_W}),$$

where $\beta_{W,i} = f(x_i)$ is the biomarker mean at the i-th dose gridpoint x_i. For the RW2, the second-order differences instead of the first-order differences of the biomarker means follow the normal distribution.

The variance parameter $\sigma^2_{\beta_W}$ is important because it steers the smoothness of the function f(x): 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 fixed or assigned an inverse gamma prior distribution.

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 RW 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.

Slots

`sigma2betaW` Contains the prior variance factor of the random walk prior for the biomarker model. If it is not a single number, it can also contain a vector with elements `a` and `b` for the inverse-gamma prior on `sigma2betaW`.

`useRW1` for specifying the random walk prior on the biomarker level: if `TRUE`, RW1 is used, otherwise RW2.

Examples

```
model <- DualEndpointRW(mu = c(0, 1),
                        Sigma = 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),
                        smooth="RW1")
```

DualResponsesDesign *Initialization function for 'DualResponsesDesign"*

Description

Initialization function for 'DualResponsesDesign"

Usage

```
DualResponsesDesign(Effmodel, data, ...)
```

Arguments

Effmodel	please refer to DualResponsesDesign class object
data	please refer to DualResponsesDesign class object
...	additional arguments for TDDesign

Value

the DualResponsesDesign class object

DualResponsesDesign-class *This is a class of design based on DLE responses using the LogisticIndepBeta model model and efficacy responses using ModelEff model class without DLE and efficacy samples. It contain all slots in RuleDesign and TDDesign class object*

Description

This is a class of design based on DLE responses using the LogisticIndepBeta model model and efficacy responses using ModelEff model class without DLE and efficacy samples. It contain all slots in RuleDesign and TDDesign class object

Slots

data the data set of DataDual class object

Effmodel the pseudo efficacy model to be used, an object class of ModelEff

Examples

```
##Construct the DualResponsesDesign for simulations
##The design comprises the DLE and efficacy models, 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 <- 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),Effdose=c(25,300),
                    nu=c(a=1,b=0.025),data=data,c=0)

##The escalation rule using the 'NextBestMaxGain' class
mynextbest<-NextBestMaxGain(DLEDuringTrialtarget=0.35,
                            DLEEndOfTrialtarget=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

design <- DualResponsesDesign(nextBest=mynextbest,
                              model=DLEmodel,
                              Effmodel=Effmodel,
                              stopping=myStopping,
                              increments=myIncrements,
                              cohortSize=mySize,
                              data=data,startingDose=25)
```

DualResponsesSamplesDesign

Initialization function for 'DualResponsesSamplesDesign"

Description

Initialization function for 'DualResponsesSamplesDesign"

Usage

```
DualResponsesSamplesDesign(Effmodel, data, ...)
```

Arguments

Effmodel	please refer to DualResponsesSamplesDesign class object
data	please refer to DualResponsesSamplesDesign class object
...	additional arguments for TDsamplesDesign

Value

the DualResponsesSamplesDesign class object

DualResponsesSamplesDesign-class

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

Description

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

Slots

data the data set of DataDual class object

Effmodel the pseudo efficacy model to be used, an object class of ModelEff

Examples

```
##Construct the DualResponsesSamplesDesign for simulations
##The design comprises the DLE and efficacy models, 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 <- DataDual(doseGrid=seq(25,300,25),placebo=FALSE)

## First for the DLE model and DLE samples
## The DLE model must be of 'ModelTox'
## (e.g 'LogisticIndepBeta') class and
## DLEsamples of 'Samples' class
options<-McmcOptions(burnin=100,step=2,samples=200)
DLEmodel <- LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),
                              DLEdose=c(25,300),data=data)
```

```
DLEsamples<-mcmc(data,DLEmodel,options)
##The efficacy model of 'ModelEff' (e.g 'Effloglog') class and the efficacy samples
Effmodel<-Effloglog(Eff=c(1.223,2.513),Effdose=c(25,300),nu=c(a=1,b=0.025),data=data,c=0)
Effsamples<-mcmc(data,Effmodel,options)
##The escalation rule using the 'NextBestMaxGainSamples' class
mynextbest<-NextBestMaxGainSamples(DLEDuringTrialtarget=0.35,
                                   DLEEndOfTrialtarget=0.3,
                                   TDderive=function(TDsamples){
                                     quantile(TDsamples,prob=0.3)},
                                   Gstarderive=function(Gstarsamples){
                                     quantile(Gstarsamples,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 36 subjects are treated
myStopping <- StoppingMinPatients(nPatients=36)
##Now specified the design with all the above information and starting with a dose of 25

design <- DualResponsesSamplesDesign(nextBest=mynextbest,
                                     cohortSize=mySize,
                                     startingDose=25,
                                     model=DLEmodel,
                                     Effmodel=Effmodel,
                                     data=data,
                                     stopping=myStopping,
                                     increments=myIncrements)
```

`DualSimulations` *Initialization function for "DualSimulations"*

Description

Initialization function for "DualSimulations"

Usage

```
DualSimulations(rhoEst, sigma2West, fitBiomarker, ...)
```

Arguments

`rhoEst`	see `DualSimulations`
`sigma2West`	see `DualSimulations`
`fitBiomarker`	see `DualSimulations`
`...`	additional parameters from `Simulations`

Value

the DualSimulations object

DualSimulations-class *Class for the simulations output from dual-endpoint model based designs*

Description

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.

Slots

`rhoEst` the vector of final posterior median rho estimates

`sigma2West` the vector of final posterior median sigma2W estimates

`fitBiomarker` list with the final dose-biomarker curve fits

DualSimulationsSummary-class *Class for the summary of dual-endpoint simulations output*

Description

In addition to the slots in the parent class SimulationsSummary, it contains two slots for the biomarker model fit information.

Details

Note that objects should not be created by users, therefore no initialization function is provided for this class.

Slots

`biomarkerFitAtDoseMostSelected` fitted biomarker level at dose most often selected

`meanBiomarkerFit` list with the average, lower (2.5 quantiles of the mean fitted biomarker level at each dose level

EffFlexi — *Initialization function for the "EffFlexi" class*

Description

Initialization function for the "EffFlexi" class

Usage

```
EffFlexi(Eff, Effdose, sigma2, sigma2betaW, smooth = c("RW1", "RW2"), data)
```

Arguments

`Eff`	the pseudo efficacy responses
`Effdose`	the corresponding dose levels for the pseudo efficacy responses
`sigma2`	the prior variance of the efficacy responses which can be specified with a single positive scalar or with two positive scalar values fot the shape `a` and the rate `b` parameters of the inverse gamma distribtuion.
`sigma2betaW`	the prior variance of the random walk model used for smoothing which can be specified with a single postive scalar or with two positive scalars representing the shape `a` and the rate `b` parameter of the inverse gamma distribution.
`smooth`	used for smoothing data for this efficacy model. That is either the "RW1", the first-order random walk model or "RW2", the second-order random walk model is used of the mean efficacy responses.
`data`	the input data to update estimates of model parameters and follow the `DataDual` object class specification

Value

the `EffFlexi` class object

EffFlexi-class — *Class for the efficacy model in flexible form for prior expressed in form of pseudo data*

Description

This is a class where a flexible form is used to describe the realtionship between the efficacy responses and the dose levels. This flexible form aims to capture different shape for the dose-efficacy curve and the mean efficacy responses at each dose level are estimated using MCMC. 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 differnece of the mean efficacy responses to its neighbouring dose levels of their mean efficacy responses. The flexible form is specified as

$$\mathbf{W}|\boldsymbol{\beta_w}, \sigma^2 \sim Normal(\mathbf{X}_w\boldsymbol{\beta_w}, \sigma^2\mathbf{I})$$

where $\mathbf{W}$ represent the column vector of the efficacy responses, $\boldsymbol{\beta_w}$ is th column vector of the mean efficacy responses for all dose levels, $\mathbf{X_w}$ is the design matrix with entries $I_{i(j)}$ which gives a value 1 if subject i is allocated to dose j. The σ^2 (sigma2) is the variance of the efficacy responses which can be either fixed or from an inverse gamma distribution.

Details

The RW1 model is given as

$$\beta_{W,(j)} - \beta_{W,(j-1)} \sim Normal(0, \sigma^2_{\beta_W})$$

where $\beta_{W,(j)}$ is the mean efficacy responses at dose j For the RW2 is given as

$$\beta_{W,(j-2)} - 2\beta_{W,(j-1)} + \beta_{W,(j)} \sim Normal(0, \sigma^2_{\beta_W})$$

The variance parameter $\sigma^2_{\beta_W}$. The variance $\sigma^2_{\beta_W}$ (sigma2betaW) will be the same at all dose levels and can either be fixed or assigned an inverse gamma prior distribution.

The `Eff` and `Effdose` are the pseduo efficacy responses and dose levels at which these pseudo efficacy responses are observed at. (see more details for `Effloglog` class) `Eff` and `Effdose` must be vector of at least length 2. The values or elements in vectors `Eff` or `Effdose` must put in the same position with its corresponding value in the other vector. The `sigma2` is the prior variance of the flexible efficacy form. The variance is either specified with a single scalar value (fixed) or postive scalar value have to be specified for the `a` shape and `b` slope parameter for th inverse gamme distribtuion. Similarly, `sigma2betaW` is the prior variance of the random walk model which can be specified with a single scalar (fixed) value or specifying positive scalar values for the shape `a` and rate `b` parameters for the inverse gamma distributions. This model will output the updated value or the updated values of the paramters of the inverse gamma distributions for $sigma^2$ (sigma2) and $\sigma^2_{\beta_W}$ (sigma2betaW)

Slots

`Eff` the pseudo efficacy responses. A vector of at least length 2 with the elements here and its corresponding value in `Effdose` must be specified in the same position. (see dtails above)

`Effdose` the dose levels at which the pseudo efficacy responses are observed. This is a vector of at least length 2 and the elements here and its corrresponding value in `Eff` must be specified in the same postion. (see details from above)

`sigma2` the prior variance of the flexible efficacy form. It can be specified with a single positive scalar or specifying `a`, the shape and `b`, the rate parameter of the inverse gamma distribution. (see details from above)

`sigma2betaW` the prior variance of the random walk model for the mean efficact responses. A single positve scalar can be specified or specifying `a`, the shape and `b`, the rate parameter of the inverse gamma distribution (see details from above)

`useFixed` a list of with logical value to each of the parameters `sigma2` and `sigma2betaw` indicating whether a fixed value is used or not; this slot is needed for internal purposes and not to be touched by the user.

`useRW1` for specifying the random walk model for the mean efficacy responses; if `TRUE`, first order random walk model is used, otherwise the second-order random walk model.

designW is the design matrix for the efficacy responses. If only the pseudo efficacy responses are used, this will be the design matrix of the pseudo efficacy responses. If there are some observed efficacy responses available. It will be the design matrix based on both the pseudo and the observed efficacy responses.

RWmat is the the difference matrix for the random walk model. This slot is needed for internal purposes and not to be touched by the user.

RWmatRank is the rank of the difference matrix. This slot is needed for internal purposes and not to be touched by the user.

Examples

```
##Obtain prior estimates for the EffFlexi (efficacy model) given the pseudo data.
##First define an empty data set by only define the dose levels used in the study
## 12 dose levels are usesd from 25 to 300 mg with increments of 25.
emptydata<-DataDual(doseGrid=seq(25,300,25))
data<-emptydata
## define the pseudo data as first fixed 2 dose levels 25 and 300 mg and
## specified in (Effdose slot).
## Then the efficacy responses observed at these two dose levels are 1.223 and 2.513 and
## specified in (Eff slot).
## The prior variance of the pseudo efficay responses. This can be either a fixed value of
## specifying the shape (a) and the rate (b) parameters for the inverse gamma distribution
## in (sigma2 slot). The prior variance of the random walk model which can be a fixed value or
## two value for the shape (a) and rate (b) parameter of the inverse gamma distribution in
## (sigma2betaW slot). The data are specified in (data slot)


Effmodel<- EffFlexi(Eff=c(1.223, 2.513),Effdose=c(25,300),
                    sigma2=c(a=0.1,b=0.1),sigma2betaW=c(a=20,b=50),
                    smooth="RW2",data=data)

##Obtain estimates from the model given some observed responses
## first specified 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))

Effmodel<- EffFlexi(Eff=c(1.223, 2.513),Effdose=c(25,300),
                    sigma2=c(a=0.1,b=0.1),sigma2betaW=c(a=20,b=50),
                    smooth="RW2",data=data)
```

Effloglog *Initialization function for the "Effloglog" class*

Description

Initialization function for the "Effloglog" class

Usage

```
Effloglog(Eff, Effdose, nu, data, c = 0)
```

Arguments

Eff	the pseudo efficacy responses
Effdose	the corresponding dose levels for the pseudo efficacy responses
nu	the precision (inverse of the variance) of the efficacy responses
data	the input data of DataDual class to update model estimates
c	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).).

Value

the Effloglog object

Effloglog-class *Class for the linear log-log efficacy model using pseudo data prior*

Description

This is the efficacy model which describe the relationship of the continuous efficacy responses and the dose levels. More specifically, this is a model to describe the linear relationship between the continuous efficacy responses and its coressponding dose level in log-log scale. The efficacy log-log model is given as

$$y_i = \theta_1 + theta_2 log(log(d_i)) + \epsilon_i$$

where y_i is the efficacy responses for subject i, d_i is the dose level treated for subject i and ϵ_i is the random error term of efficacy model at subject i such that ϵ_i has a normal distribution of mean 0 and variance $\sigma^2 = \nu^{-1}$. This variance is assumed to be the same for all subjects.

Details

There are three parameters in this model which is to intercept θ_1, the slope θ_2 and the precision ν of the efficay responses. It inherit all slots from ModelEff

The prior of this model is specified in form of pseudo data. First at least two dose levels are fixed. Then ask for experts' opinion about the efficacy values that can be obtained at each of the dose levels if one subject is treated at each of these dose levels. The prior modal estimates (same as the maximum likelihood estimates) can be obtained for the intercept and slope paramters in this model.

The Eff and Effdose are used to represent the prior in form of the pseudo data. The Eff represents the pseudo scalar efficacy values. The Effdose represents the dose levels at which these pseudo efficacy values are observed. These pseudo efficay values are always specified by assuming one subject are treated in each of the dose levels. Since at least 2 pseudo efficacy values are needed to obtain modal estimates of the intercept and slope parameters, both Eff and Effdose must be vector

of at least length 2. The position of the values or elements specified in `Eff` or `Effdose` must be corresponds to the same elements or values in the other vector.

The `nu` represents the prior presion ν of the pseudo efficacy responses. It is also known as the inverse of the variance of the pseduo efficacy responses. The precision can be a fixed constant or having a gamma distribution. Therefore, single scalar value, a fixed value of the precision can be specified. If not, two positive scalar values must be specified as the shape and rate parameter of the gamma distribution. 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.

Given the variance of the pseudo efficacy responses, the joint prior distribution of the intercept θ_1 (theta1) and the slope θ_2 (theta2) of this model is a bivariate normal distribution. A conjugate posterior joint distribution is also used for theta1 and theta2. The joint prior bivariate normal distribution has mean $\boldsymbol{\mu}_0$ and covariance matrix $(\nu\mathbf{Q}_0)^{-1}$. $\boldsymbol{\mu}_0$ is a 2×1 column vector contains the prior modal estimates of the intercept (theta1) and the slope (theta2). Based on r for $r \geq 2$ pseudo efficacy responses specified, $\mathbf{X}_0$ will be the $r \times 2$ design matrix obtained for these pseudo efficacy responses. the matrix $\mathbf{Q}_0$ will be calculated by $\mathbf{Q}_0 = \mathbf{X}_0\mathbf{X}_0^T$ where ν is the precision of the pseudo efficacy responses. For the joint posterior bivariate distribution, we have $\boldsymbol{\mu}$ as the mean and $(\nu\mathbf{Q}_0)^{-1}$ as the covariance matrix. Here, $\boldsymbol{\mu}$ is the column vector containing the posterior modal estimates of the intercept (theta1) and the slope (theta2). The design matrix $\mathbf{X}$ obtained based only on observed efficacy responses will give $\mathbf{Q} = \mathbf{X}\mathbf{X}^T$ with ν as the precision of the observed efficay responses. If no observed efficay responses are availble (i.e only pseudo efficay responses are used), the `vecmu`, `matX`, `matQ` and `vecY` represents $\boldsymbol{\mu}_0$, $\mathbf{X}_0$, $\mathbf{Q}_0$ and the column vector of pseudo efficay responses, respectively. If there are some observed efficacy responses, `vecmu`, `matX`, `matQ` and `vecY` will represent $\boldsymbol{\mu}$, $\mathbf{X}$, $\mathbf{Q}$ and the column vector contains all observed efficacy responses, respectively. (see details in about the form of prior and posterior distribution)

Slots

`Eff` the pseudo efficacy response, the scalar efficacy values. This must be a vector of at least length 2. Each element or value here must represents responses treated based on one subject. The order of its elements must corresponds to the values presented in vector `Effdose` (see details above)

`Effdose` the pseudo efficacy dose level. This is the dose levels at which the pseudo efficacy responses are observed at. This must be a vector of at least length 2 and the orde of its elements must corresponds to values presented in vector `Eff` (see detial above)

`nu` refers to the prior precision of pseudo efficacy responses. This is either a fixed value or a vector of elements `a`, a positive scalar for the shape parameter, and `b`, a positive scalar for the rate parameter for the gamma dsitribution. (see detail from above)

`useFixed` a logical value if `nu` specified is a fixed value or not. This slot is needed for internal purposes and not to be touched by the user.

`theta1` The intercept θ_1 parameter of 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 data and (if any) the observed data/ responses.

`theta2` The slope $theta_2$ parameter of the efficacy log-lgo model. This slot is used in output to display the resulting prior or posterior modal estimates obtained based on the pseudo data and (if any) the observed data/ responses.

`Pcov` refers to the covariance matrix of the intercept (phi1) and slope (phi2) paramters of this model. This slot is used in output to display the covariance matrix obtained based on the pseudo data and (if any) the observed data/responses. This slot is needed for internal purposes.

`vecmu` is the column vector of the prior or the posterior modal estimates of the intercept (phi1) and the slope (phi2). This slot is used in output to display as the mean of the prior or posterior bivariate normal distribtuion for phi1 and phi2. (see details from above)

`matX` is the design matrix based on either the pseudo or all observed efficacy response. This is used in output to display the design matrix for the pseudo or the observed efficacy responses (see details from above)

`matQ` is the square matrix of multiplying the the design matrix with its transponse. This is represented either using the only the pseudo efficay responses or only with the observed efficacy responses. This is display in the output (see details from above)

`vecY` is the column vector either contains the pseudo efficay responses or all the observed efficacy responses. This is used in output to display the pseudo or observed efficacy responses (see detail from above)

`c` is a constant value greater or equal to 0, with the default 0 leading to the model form described above. In general, the model has the form $y_i = \theta_1 + theta_2 log(log(d_i + c)) + \epsilon_i$, such that dose levels greater than $1 - c$ can be considered as described in Yeung et al. (2015).

Examples

```
##Obtain prior modal estimates for the Effloglog model (efficacy model) given the pseudo data.
##First define an empty data set by only define the dose levels used in the study,
## 12 dose levels are usesd from 25 to 300 mg with increments of 25.
emptydata<-DataDual(doseGrid=seq(25,300,25),placebo=FALSE)
data<-emptydata
## define the pseudo data as first fixed 2 dose levels 25 and 300 mg and specified in
## (Effdose slot).
## Then the efficacy responses observed at these two dose levels are 1.223 and 2.513 and
## specified in (Eff slot).
## The prior precision of the pseudo efficay responses. This can be either a fixed value of
## specifying the shape (a) and the rate (b) parameters for the gamma distribution in (nu slot).
## Then specify all data currentl available in (data slot).

Effmodel<-Effloglog(Eff=c(1.223,2.513),Effdose=c(25,300),nu=c(a=1,b=0.025),data=data,c=0)

##Obtain posterior modal estimates and other estimates from the model given some observed responses
## If there is some observations available
## first specified 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))

Effmodel<-Effloglog(Eff=c(1.223,2.513),Effdose=c(25,300),nu=c(a=1,b=0.025),data=data)
```

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)
```

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

Examples

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

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

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

# Choose the rule for the cohort-size
mySize1 <- CohortSizeRange(intervals=c(0, 30),
                           cohortSize=c(1, 3))
mySize2 <- CohortSizeDLT(DLTintervals=c(0, 1),
                         cohortSize=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,
                 cohortSize=mySize,
                 data=emptydata,
                 startingDose=3)

# Examine the design
set.seed(4235)
# MCMC parameters are set to small values only to show this example. They should be
# increased for a real case.
options <- McmcOptions(burnin=10,step=1,samples=20)
examine(design, options)

## example where examine stops because stopping rule already fulfilled
myStopping4 <- StoppingMinPatients(nPatients=3)
```

```
myStopping <- (myStopping1 & myStopping2) | myStopping4
design <- Design(model=model,
                 nextBest=myNextBest,
                 stopping=myStopping,
                 increments=myIncrements,
                 cohortSize=mySize,
                 data=emptydata,
                 startingDose=3)
examine(design,mcmcOptions=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)
myIncrements <- IncrementsRelative(intervals=c(0, 20),
                                   increments=c(1, 0.00001))
myStopping <- (myStopping1 & myStopping2)
design <- Design(model=model,
                 nextBest=myNextBest,
                 stopping=myStopping,
                 increments=myIncrements,
                 cohortSize=mySize,
                 data=emptydata,
                 startingDose=3)
examine(design, mcmcOptions=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(),
                       cohortSize = CohortSizeConst(size=3L),
                       data = emptydata,
                       startingDose = 5)

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

`ExpEff` *Compute the expected efficacy based on a given dose, a given pseudo Efficacy log-log model and a given efficacy sample*

Description

Compute the expected efficacy based on a given dose, a given pseudo Efficacy log-log model and a given efficacy sample

Usage

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

## S4 method for signature 'numeric,Effloglog,Samples'
ExpEff(dose, model, samples, ...)

## S4 method for signature 'numeric,Effloglog,missing'
ExpEff(dose, model, samples, ...)

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

Arguments

dose	the dose
model	the Effloglog class object
samples	the Samples class object (can also be missing)
...	unused

Functions

- ExpEff(dose = numeric, model = Effloglog, samples = Samples): Method for the Effloglog class
- ExpEff(dose = numeric, model = Effloglog, samples = missing): Compute the Expected Efficacy based a given dose and a given Pseudo Efficacy log log model without samples
- ExpEff(dose = numeric, model = EffFlexi, samples = Samples): Compute the Expected Efficacy based a given dose, Efficacy Flexible model with samples

Examples

```
##Obtain the expected efficacy value for a given dose, a given pseudo
## efficacy model and a given efficacy sample
##The efficacy model must be from 'ModelEff' class (model slot)
##The efficacy sample must be from 'Samples' class (sample slot)
emptydata<-DataDual(doseGrid=seq(25,300,25))
data<-emptydata
model<- EffFlexi(Eff=c(1.223, 2.513),Effdose=c(25,300),
                 sigma2=c(a=0.1,b=0.1),sigma2betaW=c(a=20,b=50),smooth="RW2",data=data)
options<-McmcOptions(burnin=100,step=2,samples=200)
set.seed(94)
samples<-mcmc(data=data,model=model,options=options)
## Given the dose 75 (dose slot)
ExpEff(dose=75,model=model,samples=samples)
##Obtain the expected efficacy value for a given dose and a given pseudo efficacy model

##The efficacy model must be from 'ModelEff' class (model slot)
emptydata<-DataDual(doseGrid=seq(25,300,25),placebo=FALSE)
data<-emptydata
```

```
model<-Effloglog(Eff=c(1.223,2.513),Effdose=c(25,300),nu=c(a=1,b=0.025),data=data,c=0)


## Given the dose 75 (dose slot)
ExpEff(dose=75,model=model)
##Obtain the expected efficacy value for a given dose, the 'EffFlexi' efficacy model and
##samples generated from this efficacy model
##The efficacy model must be from 'EffFlexi' class (model slot)
##The efficacy samples must be from 'Samples' class (samples slot)
model<- EffFlexi(Eff=c(1.223, 2.513),Effdose=c(25,300),
                 sigma2=c(a=0.1,b=0.1),sigma2betaW=c(a=20,b=50),smooth="RW2",data=data)
set.seed(94)
samples<-mcmc(data=data,model=model,options=options)
## Given the dose 75 (dose slot)
ExpEff(dose=75,model=model,samples=samples)
```

`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,Model,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,
  ...
)
```

Arguments

`object`	the `Samples` object
`model`	the `Model` object
`data`	the `Data` object
`...`	unused
`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 = Model, 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

Examples

```
# 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),
                           refDose = 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)



# 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(mu = c(0, 1),
                        Sigma = 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),
                        smooth = "RW1")

# 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)
##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)
##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),Effdose=c(25,300),
                  sigma2=c(a=0.1,b=0.1),sigma2betaW=c(a=20,b=50),smooth="RW2",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)
```

`fitGain` *Get the fiited 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 fiited 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)
```

`gain` *Compute the gain value with a given dose level, given a pseudo DLE model, a DLE sample, a pseudo Efficacy log-log model and a Efficacy sample*

Description

Compute the gain value with a given dose level, given a pseudo DLE model, a DLE sample, a pseudo Efficacy log-log model and a Efficacy sample

Usage

```
gain(dose, DLEmodel, DLEsamples, Effmodel, Effsamples, ...)

## S4 method for signature 'numeric,ModelTox,Samples,Effloglog,Samples'
gain(dose, DLEmodel, DLEsamples, Effmodel, Effsamples, ...)

## S4 method for signature 'numeric,ModelTox,Samples,EffFlexi,Samples'
gain(dose, DLEmodel, DLEsamples, Effmodel, Effsamples, ...)

## S4 method for signature 'numeric,ModelTox,missing,Effloglog,missing'
gain(dose, DLEmodel, DLEsamples, Effmodel, Effsamples, ...)
```

Arguments

`dose`	the dose
`DLEmodel`	the `ModelTox` object
`DLEsamples`	the `Samples` object (can also be missing)
`Effmodel`	the `Effloglog` or the `EffFlexi` object
`Effsamples`	the `Samples` object (can also be missing)
`...`	unused

Functions

- `gain( dose = numeric, DLEmodel = ModelTox, DLEsamples = Samples, Effmodel = EffFlexi, Effsamples = Samples )`: Compute the gain given a dose level, a pseduo DLE model, a DLE sample, the pseudo EffFlexi model and an Efficacy sample
- `gain( dose = numeric, DLEmodel = ModelTox, DLEsamples = missing, Effmodel = Effloglog, Effsamples = missing )`: Compute the gain value given a dose level, a pseudo DLE model and a pseudo efficacy model of `Effloglog` class object without DLE and the efficacy sample

Examples

```
##Obtain the gain value for a given dose, a pseudo DLE model, a DLE sample,
## a pseudo efficacy model and an efficacy sample
##The DLE model must be from 'ModelTox' class (DLEmodel slot)
emptydata<- DataDual(doseGrid=seq(25,300,25),placebo=FALSE)
data<-emptydata
DLEmodel<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=data)
DLEsamples <- mcmc(data, DLEmodel, McmcOptions(burnin=100,step=2,samples=200))

##The efficacy model must be from 'ModelEff' class (Effmodel slot)
## The DLE and efficayc samples must be from 'Samples' class (DLEsamples and Effsamples slot)
Effmodel<-Effloglog(Eff=c(1.223,2.513),Effdose=c(25,300),nu=c(a=1,b=0.025),data=data,c=0)
Effsamples <- mcmc(data, Effmodel, McmcOptions(burnin=100,step=2,samples=200))

## Given a dose level 75,
gain(dose=75,DLEmodel=DLEmodel,DLEsamples=DLEsamples,Effmodel=Effmodel,Effsamples=Effsamples)
##Obtain the gain value for a given dose, a pseudo DLE model, a DLE sample,
## the 'EffFlexi' efficacy model and an efficacy sample
##The DLE model must be from 'ModelTox' class (DLEmodel slot)
emptydata<- DataDual(doseGrid=seq(25,300,25))
data<-emptydata
DLEmodel<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=data)
DLEsamples <- mcmc(data, DLEmodel, McmcOptions(burnin=100,step=2,samples=200))

##The efficacy model must be from 'EffFlexi' class (Effmodel slot)
## The DLE and efficayc samples must be from 'Samples' class (DLEsamples and Effsamples slot)
EffFleximodel <- EffFlexi(Eff=c(1.223, 2.513),Effdose=c(25,300),
                  sigma2=c(a=0.1,b=0.1),sigma2betaW=c(a=20,b=50),smooth="RW2",data=data)
Effsamples <- mcmc(data, EffFleximodel, McmcOptions(burnin=100,step=2,samples=200))

## Given a dose level 75,
gain(dose=75,DLEmodel=DLEmodel,DLEsamples=DLEsamples,Effmodel=EffFleximodel,Effsamples=Effsamples)
##Obtain the gain value for a given dose, a pseudo DLE model and  a pseudo efficacy model
## without samples
##The DLE model must be from 'ModelTox' class (DLEmodel slot)
emptydata<- DataDual(doseGrid=seq(25,300,25),placebo=FALSE)
data<-Data(doseGrid=seq(25,300,25),placebo=FALSE)

DLEmodel<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=data)
##The efficacy model must be from 'Effloglog' class  (Effmodel slot)
Effmodel<-Effloglog(Eff=c(1.223,2.513),Effdose=c(25,300),nu=c(a=1,b=0.025),data=emptydata,c=0)
## Given a dose level 75,
```

```
gain(dose=75,DLEmodel=DLEmodel,Effmodel=Effmodel)
```

GeneralData-class *Class for general data input*

Description

Class for general data input

Slots

`ID` unique patient IDs (integer vector)

`cohort` the cohort indices (sorted values from 0, 1, 2, ...)

`nObs` number of observations

GeneralModel-class *No Intitialization function for this General class for model input*

Description

This is the general model class, from which all other specific models for involving BUGS (the software) for computing result. It inherits all slots from `AllModels`

Details

The `datamodel` must obey the convention that the data input is called exactly 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.

Slots

`datamodel` a function representing the BUGS data model specification (see the details above)

`priormodel` a function representing the BUGS prior specification (see the details above)

`modelspecs` a function computing the list of the data model and prior model specifications that are required for fully specifying them (e.g. prior parameters, reference dose, etc.), based on the data slots that are then required as arguments of this function. This will then be passed to BUGS for the computations.

`init` 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 then required as arguments of this function

`sample` names of all parameters from which you would like to save the MCMC samples.

See Also

`Model`

GeneralSimulations *Initialization function for "GeneralSimulations"*

Description

Initialization function for "GeneralSimulations"

Usage

```
GeneralSimulations(data, doses, seed)
```

Arguments

data	see GeneralSimulations
doses	see GeneralSimulations
seed	see GeneralSimulations

Value

the GeneralSimulations object

GeneralSimulations-class *General class for the simulations output*

Description

This class captures trial simulations.

Details

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.

Slots

data list of produced Data objects

doses the vector of final dose recommendations

seed random generator state before starting the simulation

GeneralSimulationsSummary-class

Class for the summary of general simulations output

Description

Note that objects should not be created by users, therefore no initialization function is provided for this class.

Slots

`target` target toxicity interval

`targetDoseInterval` corresponding target dose interval

`nsim` number of simulations

`propDLTs` proportions of DLTs in the trials

`meanToxRisk` mean toxicity risks for the patients

`doseSelected` doses selected as MTD

`toxAtDosesSelected` true toxicity at doses selected

`propAtTarget` Proportion of trials selecting target MTD

`doseMostSelected` dose most often selected as MTD

`obsToxRateAtDoseMostSelected` observed toxicity rate at dose most often selected

`nObs` number of patients overall

`nAboveTarget` number of patients treated above target tox interval

`doseGrid` the dose grid that has been used

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

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

```
# 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),
                           refDose = 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")
```

---

getEff — *Extracting efficacy responses for subjects without or with a DLE. This is a class where we separate efficacy responses with or without a DLE. It outputs the efficacy responses and their corresponding dose levels treated at in two categories (with or without DLE)*

Description

Extracting efficacy responses for subjects without or with a DLE. This is a class where we separate efficacy responses with or without a DLE. It outputs the efficacy responses and their corresponding dose levels treated at in two categories (with or without DLE)

Usage

```
getEff(object, ...)

## S4 method for signature 'DataDual'
getEff(object, x, y, w, ...)
```

Arguments

object	for data input from DataDual object
...	unused
x	todo
y	todo
w	todo

Examples

```
## Separate the efficacy responses of subjects with or without DLE
## data must be specified for in '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))
##Display the efficacy response and their corresponding dose levels
## treated at when no or a DLE is observed
getEff(data)
```

---

getMinInfBeta *Get the minimal informative unimodal beta distribution*

Description

As defined in Neuenschwander et al (2008), this function computes the parameters of the minimal informative unimodal beta distribution, given the request that the p-quantile should be q, i.e. X ~ Be(a, b) with Pr(X <= q) = p.

Usage

```
getMinInfBeta(p, q)
```

Arguments

p	the probability (> 0 and < 1)
q	the quantile (> 0 and < 1)

Value

the two resulting beta parameters a and b in a list

IncrementMin *Initialization function for "IncrementMin"*

Description

Initialization function for "IncrementMin"

Usage

```
IncrementMin(IncrementsList)
```

Arguments

`IncrementsList` see `IncrementMin`

Value

the `IncrementMin` object

IncrementMin-class *Max increment based on minimum of multiple increment rules*

Description

This class can be used to combine multiple increment rules with the MIN operation.

Details

`IncrementsList` contains all increment rules, which are again objects of class `Increments`. The minimum of these individual increments is taken to give the final maximum increment.

Slots

`IncrementsList` list of increment rules

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

# 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
```

```
myIncrements1 <- IncrementsRelativeDLT(DLTintervals = c(0, 1, 3),
                                       increments = c(1, 0.33, 0.2))

myIncrements2 <- IncrementsRelative(intervals=c(0, 20),
                                    increments=c(1, 0.33))

# Now we combine the 2 rules
combIncrement <- IncrementMin(IncrementsList=
                              list(myIncrements1,myIncrements2))
```

`Increments-class` *The virtual class for controlling increments*

Description

The virtual class for controlling increments

See Also

`IncrementsRelative`, `IncrementsRelativeDLT`, `IncrementsRelativeParts`

`IncrementsNumDoseLevels` *Initialization function for "IncrementsNumDoseLevels"*

Description

Initialization function for "IncrementsNumDoseLevels"

Usage

```
IncrementsNumDoseLevels(maxLevels = 1)
```

Arguments

`maxLevels`	see `IncrementsNumDoseLevels`

Value

the `IncrementsNumDoseLevels` object

IncrementsNumDoseLevels-class

Increments control based on number of dose levels

Description

Increments control based on number of dose levels

Slots

`maxLevels` scalar positive integer for the number of maximum dose levels to increment for the next dose. It defaults to 1, which means that no dose skipping is allowed - the next dose can be maximum one level higher than the current dose.

Examples

```
# As example, here is the rule for:
# maximum skip one dose level, that is 2 dose levels higher is maximum
# increment
myIncrements <- IncrementsNumDoseLevels(maxLevels=2)
```

IncrementsRelative

Initialization function for "IncrementsRelative"

Description

Initialization function for "IncrementsRelative"

Usage

```
IncrementsRelative(intervals, increments)
```

Arguments

`intervals`	see `IncrementsRelative`
`increments`	see `IncrementsRelative`

Value

the `IncrementsRelative` object

IncrementsRelative-class

Increments control based on relative differences in intervals

Description

Note that `intervals` is to be read as follows. If for example, we want to specify three intervals: First 0 to less than 50, second at least 50 up to less than 100 mg, and third at least 100 mg, then we specify `intervals` to be `c(0, 50, 100)`. That means, the right bound of the intervals are exclusive to the interval, and the last interval goes from the last value until infinity.

Slots

`intervals` a vector with the left bounds of the relevant intervals

`increments` a vector of the same length with the maximum allowable relative increments in the `intervals`

Examples

```
# As example, here is the 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

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

IncrementsRelativeDLT *Initialization function for "IncrementsRelativeDLT"*

Description

Initialization function for "IncrementsRelativeDLT"

Usage

```
IncrementsRelativeDLT(DLTintervals, increments)
```

Arguments

`DLTintervals`	see `IncrementsRelativeDLT`
`increments`	see `IncrementsRelativeDLT`

Value

the `IncrementsRelativeDLT` object

IncrementsRelativeDLT-class

Increments control based on relative differences in terms of DLTs

Description

Note that `DLTintervals` is to be read as follows. If for example, we want to specify three intervals: First 0 DLTs, second 1 or 2 DLTs, and third at least 3 DLTs, then we specify `DLTintervals` to be `c(0, 1, 3)`. That means, the right bound of the intervals are exclusive to the interval – the vector only gives the left bounds of the intervals. The last interval goes from 3 to infinity.

Slots

`DLTintervals` an integer vector with the left bounds of the relevant DLT intervals

`increments` a vector of the same length with the maximum allowable relative increments in the `DLTintervals`

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

myIncrements <- IncrementsRelativeDLT(DLTintervals = c(0, 1, 3),
                                      increments = c(1, 0.33, 0.2))
```

IncrementsRelativeParts

Initialization function for "IncrementsRelativeParts"

Description

Initialization function for "IncrementsRelativeParts"

Usage

```
IncrementsRelativeParts(dltStart, cleanStart, ...)
```

Arguments

`dltStart`	see `IncrementsRelativeParts`
`cleanStart`	see `IncrementsRelativeParts`
`...`	additional slots from `IncrementsRelative`

Value

the IncrementsRelativeParts object

IncrementsRelativeParts-class

Increments control based on relative differences in intervals, with special rules for part 1 and beginning of part 2

Description

Note that this only works in conjunction with DataParts objects. If the part 2 will just be started in the next cohort, then the next maximum dose will be either dltStart (e.g. -1) shift of the last part 1 dose in case of a DLT in part 1, or cleanStart shift (e.g. 0) in case of no DLTs in part 1. 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 of the data object. If part 2 has been started before, the usual relative increment rules apply, see IncrementsRelative.

Slots

dltStart integer giving the dose level increment for starting part 2 in case of a DLT in part 1

cleanStart integer giving the dose level increment for starting part 2 in case of a DLT in part 1. If this is less or equal to 0, then the part 1 ladder will be used to find the maximum next dose. If this is larger than 0, then the relative increment rules will be applied to find the next maximum dose level.

Examples

```
myIncrements <- IncrementsRelativeParts(dltStart=0,
                                        cleanStart=1)
```

---

initialize,DualEndpointOld-method

Initialization method for the "DualEndpointOld" class

Description

Initialization method for the "DualEndpointOld" class

Usage

```
## S4 method for signature 'DualEndpointOld'
initialize(
  .Object,
  mu,
  Sigma,
  sigma2betaW,
  sigma2W,
  rho,
  smooth = c("RW1", "RW2"),
  ...
)
```

Arguments

.Object	the DualEndpointOld we want to initialize
mu	see DualEndpointOld
Sigma	see DualEndpointOld
sigma2betaW	see DualEndpointOld
sigma2W	see DualEndpointOld
rho	see DualEndpointOld
smooth	either "RW1" (default) or "RW2", for specifying the random walk prior on the biomarker level.
...	not used

LogisticIndepBeta *Intialization function for "LogisticIndepBeta" class*

Description

Intialization function for "LogisticIndepBeta" class

Usage

```
LogisticIndepBeta(binDLE, DLEdose, DLEweights, data)
```

Arguments

binDLE	the number of subjects observed with a DLE, the pseudo DLE responses
DLEdose	the corresponding dose levels for the pseudo DLE responses, pseudo dose levels
DLEweights	the total number of subjects treated at each of the dose levels, pseudo weights
data	the input data to update estimates of model parameters and follow the Data object class specification

Value

the LogisticIndepBeta

LogisticIndepBeta-class

No initialization function Standard logistic model with prior in form of pseudo data

---

Description

This is a class for the two-parameter logistic regression DLE model with prior expressed in form of pseudo data. This model describe the relationhship of the binary DLE (dose-limiting events) responses and the dose levels. More specifically, this DLE model reprsents the relationship of the probabilities of the occurrence of a DLE with their corresponding dose levels in log scale. This model is specified as

$$p(d_{(j)}) = \frac{exp(\phi_1 + \phi_2 log(d_{(j)}))}{1 + exp(\phi_1 + \phi_2 log(d_{(j)}))}$$

for any dose j where $p(d_{(j)})$ is the probability of the occurrence of a DLE at dose j. The two parameters of this model is the intercept ϕ_1 and the slope ϕ_2 It inherits all slots from ModelTox class.

Details

The pseudo data can be interpreted as as if we obtain some observations before the trial starts. These pseudo data can be used to express our prior, the initial beliefs for the model parameter(s). The pseudo data are expressed in the following way. First, fix at least two dose levels which are Then ask for experts' opinion how many subjects are to be treated at each of these dose levels and the number of subjects observed with DLE are observed. 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 these pseudo data are independent 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 perior density function of the two paramaters in the model is used. For details about the form of all these joint prior and posterior probability density function, please refers to Whitehead and Willamson (1998).

When expressing the pseudo data, binDLE,DLEdose and DLEweights are used. The binDLE represents the number of subjects observed with DLE. Note that, since the imaginary nature of the pseudo data, the number of subjects observed wtih DLE is not necesssary to be integer(s) but any scalar value. The DLEdose represents the dose levels at which the pseudo DLE responses (binDLE) are observed. The DLEweights represents the total number of subjects treated. Since at least two DLE pseudo responses are needed to obtain prior modal estimates (same as the maximum likelihood estimates) for the model parameters. binDLE, DLEdose and DLEweights must all be vectors of at least length 2. Since given one pseudo DLE responses, the number of subjects observed with a DLE relates to at which dose level they are treated and the total number of of subjects treated

at this dose level. Therefore, each of the elements in any of the vectors of binDLE, DLEdose and DLEweights must have a corresponding elements in the other two vectors. A set of three values with one of each in the vectors of binDLE, DLEdose and DLEweights. In this model, each of these three values must be specified in the same position as in each of the vector of binDLE, DLEdose and DLEweights. The order of the values or elements in one of the vector binDLE, DLEdose and DLEweights must corresponds to the values or elements specified in the other two vectors.

Slots

binDLE represents the vector of pseudo DLE responses. This must be at least f length 2 and the order of its elements must corresponds to values specified in DLEdose and DLEweights. (see details from above)

DLEdose represents the vector of the corresponding dose levels observed at each of the pseudo DLE responses (binDLE). This mus be at least of length 2 and the order of its elements must corresponds to values specified in binDLE and DLEweights. (see details from above)

DLEweights refers to the total number of subjects treated at each of the pseudo dose level (DLEdose). This must be of length of at least 2 and the oreder of its elements must corresponds to values specified in binDLE and DLEdose. (see details from above)

phi1 refers 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 refers to 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 refers to the 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.

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))

##specified our data set is the empty data
data<-emptydata
## Given the pseudo data such that
## Fix two dose level 25 and 300 mg and specified in (DLEdose slot).
## Total number of subjects treated in each of these levels is 3, specified in (DLEweights slot).
## The number of subjects observed with a DLE is 1.05 at dose 25 mg and 1.8 at dose 300 mg,
## and specified in (binDLE slot).
## the data set we used in the emptydata set, and specified in (data slot).
## Then to modal estimates of the model parameters.
model<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=data)

##using a data set (see data -class example specification) with observed DLE responses
```

```
##to obtain posterior modal estimates.
##for the model given the pseudo data

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

`LogisticKadane` *Initialization function for the "LogisticKadane" class*

Description

Initialization function for the "LogisticKadane" class

Usage

```
LogisticKadane(theta, xmin, xmax)
```

Arguments

`theta`	the target toxicity probability
`xmin`	the minimum of the dose range
`xmax`	the maximum of the dose range

Value

the `LogisticKadane`

`LogisticKadane-class` *Reparametrized logistic model*

Description

This is the logistic model in the parametrization of Kadane et al. (1980).

Details

Let $\rho_0 = p(x_{min})$ be the probability of a DLT and the minimum dose x_{min}, and let γ be the dose with target toxicity probability θ, i.e. $p(\gamma) = \theta$. Then it can easily be shown that the logistic regression model has intercept

$$\frac{\gamma logit(\rho_0) - x_{min} logit(\theta)}{\gamma - x_{min}}$$

and slope

$$\frac{logit(theta) - logit(\rho_0)}{\gamma - x_{min}}$$

The prior is a uniform distribution for γ between x_{min} and x_{max}, and for ρ_0 as well a uniform distribution between 0 and θ.

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.

Slots

`theta` the target toxicity probability θ

`xmin` the minimum of the dose range x_{min}

`xmax` the maximum of the dose range x_{max}

Examples

```
model <- LogisticKadane(theta = 0.33,
                        xmin = 1,
                        xmax = 200)
```

`LogisticLogNormal` *Initialization function for the "LogisticLogNormal" class*

Description

Initialization function for the "LogisticLogNormal" class

Usage

```
LogisticLogNormal(mean, cov, refDose)
```

Arguments

`mean`	the prior mean vector
`cov`	the prior covariance matrix
`refDose`	the reference dose

Value

the `LogisticLogNormal` object

`LogisticLogNormal-class`

Standard logistic model with bivariate (log) normal prior

Description

This is the usual logistic regression model with a bivariate normal prior on the intercept and log slope.

Details

The covariate is the natural logarithm of the dose x divided by the reference dose x^*:

$$logit[p(x)] = \alpha + \beta \cdot \log(x/x^*)$$

where $p(x)$ is the probability of observing a DLT for a given dose x.

The prior is

$$(\alpha, \log(\beta)) \sim Normal(\mu, \Sigma)$$

The slots of this class contain the mean vector and the covariance matrix of the bivariate normal distribution, as well as the reference dose.

Note that the parametrization inside the class uses alpha0 and alpha1. alpha0 is identical to the intercept α above and is the log-odds for a DLT at the reference dose x*. Therefore, the prior mean for alpha0 is the expected log-odds at the reference dose x* before observing any data. Note that the expected odds is not just the exp of the prior mean of alpha0, because the non-linearity of the exp transformation. The log-normal distribution on Wikipedia gives the formula for computing the prior mean of exp(alpha0). alpha0 is the log(alpha) in the Neuenschwander et al. (2008) paper. alpha1 is identical to β above and equals the beta in the Neuenschwander et al paper. exp(alpha1) gives the odds-ratio for DLT between two doses that differ by the factor exp(1) ~ 2.7. alpha1 has a log-normal distribution in the LogisticLogNormal model in order to ensure positivity of alpha1 and thus exp(alpha1) > 1.

Slots

`mean` the prior mean vector μ

`cov` the prior covariance matrix Σ

`refDose` the reference dose x^*

Examples

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

`LogisticLogNormalMixture`
Initialization function for the "LogisticLogNormalMixture" class

---

Description

Initialization function for the "LogisticLogNormalMixture" class

Usage

```
LogisticLogNormalMixture(mean, cov, refDose, shareWeight)
```

Arguments

`mean`	the prior mean vector
`cov`	the prior covariance matrix
`refDose`	the reference dose
`shareWeight`	the prior weight for the share component

Value

the `LogisticLogNormalMixture` object

`LogisticLogNormalMixture-class`
Standard logistic model with online mixture of two bivariate log normal priors

---

Description

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

Details

$$p(x) = \pi p_1(x) + (1 - \pi) p_2(x)$$

where π is the probability for the model $p(x)$ being the same as the model $p_1(x)$ - this 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 $p_2(x)$. 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

`shareWeight` the prior weight for sharing the same model $p_1(x)$

See Also

the `DataMixture` class for use with this model

Examples

```
## decide on the dose grid:
doseGrid <- 1:80

## and MCMC options:
options <- McmcOptions()

## the classic model would be:
model <- LogisticLogNormal(mean = c(-0.85, 1),
                           cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
                           refDose = 50)

nodata <- Data(doseGrid=doseGrid)

priorSamples <- mcmc(nodata, model, options)
plot(priorSamples, model, nodata)

## set up the mixture model and data share object:
modelShare <- LogisticLogNormalMixture(shareWeight=0.1,
                                       mean = c(-0.85, 1),
                                       cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
                                       refDose = 50)

nodataShare <- DataMixture(doseGrid=doseGrid,
                           xshare=
                             c(rep(10, 4),
                               rep(20, 4),
                               rep(40, 4)),
                           yshare=
                             c(rep(0L, 4),
                               rep(0L, 4),
```

```
                                rep(0L, 4)))

## now compare with the resulting prior model:
priorSamplesShare <- mcmc(nodataShare, modelShare, options)
plot(priorSamplesShare, modelShare, nodataShare)
```

`LogisticLogNormalSub` *Initialization function for the "LogisticLogNormalSub" class*

Description

Initialization function for the "LogisticLogNormalSub" class

Usage

```
LogisticLogNormalSub(mean, cov, refDose)
```

Arguments

`mean`	the prior mean vector
`cov`	the prior covariance matrix
`refDose`	the reference dose

Value

the `LogisticLogNormalSub` object

`LogisticLogNormalSub-class` *Standard logistic model with bivariate (log) normal prior with substractive dose standardization*

Description

This is the usual logistic regression model with a bivariate normal prior on the intercept and log slope.

Details

The covariate is the dose x minus the reference dose x^*:

$$logit[p(x)] = \alpha + \beta \cdot (x - x^*)$$

where $p(x)$ is the probability of observing a DLT for a given dose x.

The prior is

$$(\alpha, \log(\beta)) \sim Normal(\mu, \Sigma)$$

The slots of this class contain the mean vector and the covariance matrix of the bivariate normal distribution, as well as the reference dose.

Slots

`mean` the prior mean vector μ

`cov` the prior covariance matrix Σ

`refDose` the reference dose x^*

Examples

```
model <- LogisticLogNormalSub(mean = c(-0.85, 1),
                              cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
                              refDose = 50)
```

`LogisticNormal` *Initialization function for the "LogisticNormal" class*

Description

Initialization function for the "LogisticNormal" class

Usage

```
LogisticNormal(mean, cov, refDose)
```

Arguments

`mean`	the prior mean vector
`cov`	the prior covariance matrix
`refDose`	the reference dose

Value

the `LogisticNormal` object

LogisticNormal-class *Standard logistic model with bivariate normal prior*

Description

This is the usual logistic regression model with a bivariate normal prior on the intercept and slope.

Details

The covariate is the natural logarithm of the dose x divided by the reference dose x^*:

$$logit[p(x)] = \alpha + \beta \cdot \log(x/x^*)$$

where $p(x)$ is the probability of observing a DLT for a given dose x.

The prior is

$$(\alpha, \beta) \sim Normal(\mu, \Sigma)$$

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.

Slots

`mean` the prior mean vector μ

`cov` the prior covariance matrix Σ

`prec` the prior precision matrix Σ^{-1}

`refDose` the reference dose x^*

Examples

```
# Define the dose-grid
emptydata <- Data(doseGrid = c(1, 3, 5, 10, 15, 20, 25, 40, 50, 80, 100))


model <- LogisticNormal(mean = c(-0.85, 1),
                        cov = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
                        refDose = 50)

options <- McmcOptions(burnin=100,
                       step=2,
                       samples=1000)

options(error=recover)
mcmc(emptydata, model, options)
```

LogisticNormalFixedMixture

Initialization function for the "LogisticNormalFixedMixture" class

Description

Initialization function for the "LogisticNormalFixedMixture" class

Usage

```
LogisticNormalFixedMixture(components, weights, refDose, logNormal = FALSE)
```

Arguments

`components`	the specifications of the mixture components: a list with one list of `mean` and `cov` for each bivariate (log) normal prior
`weights`	the weights of the components, these must be positive and will be normalized to sum to 1
`refDose`	the reference dose
`logNormal`	should a log normal prior be specified, such that the mean vectors and covariance matrices are valid for the intercept and log slope? (not default)

Value

the `LogisticNormalFixedMixture` object

LogisticNormalFixedMixture-class

Standard logistic model with fixed mixture of multiple bivariate (log) normal priors

Description

This is standard logistic regression model with a 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.

Details

The covariate is the natural logarithm of the dose x divided by the reference dose x^*:

$$logit[p(x)] = \alpha + \beta \cdot \log(x/x^*)$$

where $p(x)$ is the probability of observing a DLT for a given dose x.

The prior is

$$(\alpha, \beta) \sim \sum_{j=1}^{K} w_j Normal(\mu_j, \Sigma_j)$$

if a normal prior is used and

$$(\alpha, \log(\beta)) \sim \sum_{j=1}^{K} w_j Normal(\mu_j, \Sigma_j)$$

if a log normal prior is used.

The weight w_j of the components are fixed and sum to 1.

The (additional) slots of this class comprise two lists, containing the mean vector, the covariance and precision matrices of the two bivariate normal distributions each, the parameters of the beta prior for the first component weight, as well as the reference dose. Moreover, a slot specifies whether a log normal prior is used.

Slots

`components` a list with one entry per component of the mixture. Each entry is a list with `mean`, `cov` and `prec` for the bivariate normal prior

`weights` the weights of the components, these must be positive and sum to 1

`refDose` the reference dose x^*

`logNormal` is a log normal prior specified for each of the components?

Examples

```
model <- LogisticNormalFixedMixture(components =
                                      list(comp1 = list(mean = c(-0.85, 1),
                                                        cov = matrix(c(1, -0.5, -0.5, 1),
                                                                     nrow = 2)),
                                           comp2 = list(mean = c(1, 1.5),
                                                        cov = matrix(c(1.2, -0.45, -0.45, 0.6),
                                                                     nrow = 2))),
                                    weights = c(0.3,0.7),
                                    refDose = 50)
```

LogisticNormalMixture *Initialization function for the "LogisticNormalMixture" class*

Description

Initialization function for the "LogisticNormalMixture" class

Usage

```
LogisticNormalMixture(comp1, comp2, weightpar, refDose)
```

Arguments

comp1	the specifications of the first component: a list with mean and cov for the first bivariate normal prior
comp2	the specifications of the second component
weightpar	the beta parameters for the weight of the first component
refDose	the reference dose

Value

the LogisticNormalMixture object

LogisticNormalMixture-class *Standard logistic model with flexible mixture of two bivariate normal priors*

Description

This is standard logistic regression model with a mixture of two bivariate normal priors on the intercept and slope parameters. 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.

Details

The covariate is the natural logarithm of the dose x divided by the reference dose x^*:

$$logit[p(x)] = \alpha + \beta \cdot \log(x/x^*)$$

where $p(x)$ is the probability of observing a DLT for a given dose x.

The prior is

$$(\alpha, \beta) \sim w * Normal(\mu_1, \Sigma_1) + (1 - w) * Normal(\mu_2, \Sigma_2)$$

The weight w for the first component is assigned a beta prior B(a, b).

The slots of this class comprise two lists, containing the mean vector, the covariance and precision matrices of the two bivariate normal distributions each, the parameters of the beta prior for the first component weight, as well as the reference dose.

Slots

`comp1` the specifications of the first component: a list with `mean`, `cov` and `prec` for the first bivariate normal prior

`comp2` the specifications of the second component

`weightpar` the beta parameters for the weight of the first component

`refDose` the reference dose x^*

Examples

```
model <- LogisticNormalMixture(comp1 = list(mean = c(-0.85, 1),
                                            cov = matrix(c(1, -0.5, -0.5, 1),
                                                         nrow = 2)),
                               comp2 = list(mean = c(1, 1.5),
                                            cov = matrix(c(1.2, -0.45, -0.45, 0.6),
                                                         nrow = 2)),
                               weightpar = c(a=1, b=1),
                               refDose = 50)
```

`logit` *Shorthand for logit function*

Description

Shorthand for logit function

Usage

```
logit(x)
```

Arguments

`x`	the function argument

Value

the logit(x)

matchTolerance — *Helper function for value matching with tolerance*

Description

This is a modified version of `match` that supports tolerance.

Usage

```
matchTolerance(x, table)

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`

Functions

- `x %~% table`: Helper function for checking inclusion in a table with tolerance

Examples

```
myDose <- c(rep(0.030, 6), rep(0.050, 3), rep(0.075, 4), rep(0.1, 9), rep(0.15, 7))
doseGrid <- seq(from = .025, to = .15, by = .005)

myDose %in% doseGrid
matchTolerance(myDose, doseGrid)
myDose %~% doseGrid

matchTolerance(c(myDose, 500), doseGrid)
c(myDose, 500) %~% doseGrid
```

maxDose *Determine the maximum possible next dose*

Description

Determine the upper limit of the next dose based on the increments rule.

Usage

```
maxDose(increments, data, ...)

## S4 method for signature 'IncrementsRelative,Data'
maxDose(increments, data, ...)

## S4 method for signature 'IncrementsNumDoseLevels,Data'
maxDose(increments, data, ...)

## S4 method for signature 'IncrementsRelativeParts,DataParts'
maxDose(increments, data, ...)

## S4 method for signature 'IncrementsRelativeDLT,Data'
maxDose(increments, data, ...)

## S4 method for signature 'IncrementMin,Data'
maxDose(increments, data, ...)
```

Arguments

`increments`	The rule, an object of class `Increments`
`data`	The data input, an object of class `Data`
`...`	further arguments

Details

This function outputs the maximum possible next dose, based on the corresponding rule `increments` and the `data`.

Value

the maximum possible next dose

Functions

- `maxDose(increments = IncrementsRelative, data = Data)`: Determine the maximum possible next dose based on relative increments
- `maxDose(increments = IncrementsNumDoseLevels, data = Data)`: Determine the maximum possible next dose based on maximum dose levels to increment for the next dose

- `maxDose(increments = IncrementsRelativeParts, data = DataParts)`: Determine the maximum possible next dose based on relative increments and part 1 and 2
- `maxDose(increments = IncrementsRelativeDLT, data = Data)`: Determine the maximum possible next dose based on relative increments determined by DLTs so far
- `maxDose(increments = IncrementMin, data = Data)`: Determine the maximum possible next dose based on multiple increment rules (taking the minimum across individual increments).

Examples

```
# Create the data
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),
             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)))


# In this example we define a rule for dose increments which would allow:
#   - doubling the dose if the last dose was below 20
#   - only increasing the dose by 1.33 if the last dose was equal or above 20
myIncrements <- IncrementsRelative(intervals=c(0, 20),
                                   increments=c(1, 0.33))

# Based on the rule above, we then calculate the maximum dose allowed
nextMaxDose <- maxDose(myIncrements,
                       data=data)


# Create the data
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),
             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)))


# In this example we define a rule for dose increments which would allow:
# maximum skip one dose level, that is 2 dose levels higher is maximum
# increment
myIncrements <- IncrementsNumDoseLevels(maxLevels=2)

# Based on the rule above, we then calculate the maximum dose allowed
nextMaxDose <- maxDose(myIncrements,
                       data=data)


# create an object of class 'DataParts'
myData <- 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))


myIncrements <- IncrementsRelativeParts(dltStart=0,
                                        cleanStart=1)

nextMaxDose <- maxDose(myIncrements,
                       data=myData)


# 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)))


# In this example we define a rule for dose increments which would allow:
#   - doubling the dose if no DLTs were yet observed
#   - only increasing the dose by 1.33 if 1 or 2 DLTs were already observed
#   - only increasing the dose by 1.2 if at least 3 DLTs were already observed
myIncrements <- IncrementsRelativeDLT(DLTintervals = c(0, 1, 3),
                                      increments = c(1, 0.33, 0.2))

# Based on the rule above, we then calculate the maximum dose allowed
nextMaxDose <- maxDose(myIncrements,
                       data=data)


# Create the data
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),
             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)))


# 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

# 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
```

```
myIncrements1 <- IncrementsRelativeDLT(DLTintervals = c(0, 1, 3),
                                       increments = c(1, 0.33, 0.2))

myIncrements2 <- IncrementsRelative(intervals=c(0, 20),
                                    increments=c(1, 0.33))

# Now we combine the 2 rules
combIncrement <- IncrementMin(IncrementsList=
                              list(myIncrements1,myIncrements2))

# Finally we then calculate the maximum dose allowed by taking the minimum of the two rules
nextMaxDose <- maxDose(combIncrement,
                       data)
```

`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),
                           cohortSize = 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(DLTintervals=c(0, 1),
                         cohortSize=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 *Obtain posterior samples for all model parameters*

Description

This is the function to actually run the 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,
  program = c("JAGS"),
  verbose = FALSE,
  fromPrior = data@nObs == 0L,
  ...
)

## S4 method for signature 'DataMixture,GeneralModel,McmcOptions'
mcmc(
  data,
  model,
  options,
```

```
  fromPrior = 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, ...)
```

Arguments

`data`	The data input, an object of class `GeneralData`
`model`	The model input, an object of class `GeneralModel`
`options`	MCMC options, an object of class `McmcOptions`
`...`	unused
`program`	the program which shall be used: currently only "JAGS" is supported
`verbose`	shall progress bar and messages be printed? (not default)
`fromPrior`	sample from the prior only? Defaults to checking if nObs is 0. For some models it might be necessary to specify it manually here though.

Details

Reproducible samples can be obtained by setting the seed via `set.seed` before in the user code as usual. However, note that because the RNG sampler used is external to R, running this MCMC function will not change the seed position – that is, the repeated call to this function will then result in exactly the same output.

Value

The posterior samples, an object of class `Samples`.

Functions

- `mcmc(data = GeneralData, model = GeneralModel, options = McmcOptions)`: Standard method which uses JAGS
- `mcmc(data = DataMixture, model = GeneralModel, options = McmcOptions)`: Method for DataMixture with different fromPrior 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 distribtuion. 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 $sigma^2_{beta_W}$, 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 efficay responses. In addition, both sigma2 and sigma2betaW acan 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.

Examples

```
# create some data from the class 'Data'
myData <- 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
model <- LogisticLogNormal(mean=c(-0.85, 1),
                           cov=
                             matrix(c(1, -0.5, -0.5, 1),
                                    nrow=2),
                           refDose=56)


# Sample from the posterior distribution
options <- McmcOptions(burnin=100,
                       step=2,
                       samples=1000)

samples <- mcmc(data = myData, model = model, options=options)


##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)
##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),Effdose=c(25,300),nu=c(a=1,b=0.025),data=data,c=0)

## options must be ''McmcOptions' class
options<-McmcOptions(burnin=100,step=2,samples=200)
set.seed(94)
samples<-mcmc(data=data,model=Effmodel,options=options)
##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),Effdose=c(25,300),
                  sigma2=c(a=0.1,b=0.1),sigma2betaW=c(a=20,b=50),smooth="RW2",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)
```

McmcOptions *Initialization function for the "McmcOptions" class*

Description

Initialization function for the "McmcOptions" class

Usage

```
McmcOptions(burnin = 10000L, step = 2L, samples = 10000L)
```

Arguments

burnin	number of burn-in iterations which are not saved (default: 10,000)
step	only every step-th iteration is saved after the burn-in (default: 2)
samples	number of resulting samples (by default 10,000 will result)

Value

the McmcOptions object

McmcOptions-class *Class for the three canonical MCMC options*

Description

Class for the three canonical MCMC options

Slots

iterations number of MCMC iterations

burnin number of burn-in iterations which are not saved

step only every step-th iteration is saved after the burn-in

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
options <- 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),
                           cohortSize = 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(DLTintervals=c(0, 1),
                         cohortSize=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)
```

`Model-class` *Class for the model input*

Description

This is the model class for single agent dose escalation, from which all other specific models inherit. It inherits all slots from `GeneralModel`.

Details

The `datamodel` must obey the convention that the data input is called exactly as in the `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.

The `dose` function has as first argument `prob`, a scalar toxicity probability which is targeted. Additional arguments are model parameters. Then it computes, using model parameter(s) (samples), the resulting dose. Note that the model parameters are called exactly as in the `model` and must be included in the `sample` vector. The vectors of all samples for these parameters will then be supplied to the function. So your function must be able to process vectors of the model parameters, i.e. it must vectorize over them.

The prob function has as first argument dose, which is a scalar dose. Additional arguments are model parameters. Then it computes, using model parameter(s) (samples), the resulting probability of toxicity at that dose. Again here, the function must vectorize over the model parameters.

If you work with multivariate parameters, then please assume that your the two functions receive either one parameter value as a row vector, or a samples matrix where the rows correspond to the sampling index, i.e. the layout is then nSamples x dimParameter.

Note that dose and prob are the inverse functions of each other.

Slots

dose a function computing the dose reaching a specific target probability, based on the model parameters and additional prior settings (see the details above)

prob a function computing the probability of toxicity for a specific dose, based on the model parameters and additional prior settings (see the details above)

See Also

LogisticNormal, LogisticLogNormal, LogisticLogNormalSub, LogisticKadane, DualEndpoint

---

ModelEff-class *No Initialization function class for Efficacy models using pseudo data prior*

Description

This is a class of which contains all efficacy models for which their prior are specified in form of pseudo data. It inherits all slots from ModelPseudo

Details

The dose function has a first argument ExpEff, a scalar expected efficacy value which is targeted. Additional arguements are model parameters. It computes using modal estimate(s) or samples model parameter(s), the resulting expected efficacy value at that dose level. If samples of the model parameters are used, the function must vectorize over the model parameters.

The ExpEff function has a first argument dose, a scalar dose level which is targeted. Additional arguments are model parameters. It computes using modal estimates or samples of the model parameter(s), the resulting dose level given that particular expected efficacy value. If samples of the model parameter(s) are used, the function must vectorize over the model parameters.

The data must obey the covention that the data input is called exactly in the DataDual class. This refers to any observed Efficacy/biomarker responses (w in DataDual class), the dose (levels) (x in DataDual or Data class) at which these responses are observed, all dose levels considered in the study (doseGrid in DataDual or Data) class and other specifications in DataDual class that can be used to generate prior or posterior modal estimates or samples estimates for model parmater(s). If no responses is observed, at least doseGrid in DataDual has to be specified in data slot for which prior modal estimates or samples can be obtained for model parameters based on the specified pseudo data.

Slots

- `dose` a function computing the dose reaching a specific target value of expected efficacy, based on the model parameter(s). The model parameter(s) (samples) are obtained based on prior specified in form of pseudo data and if any together with any observed responses (see details above)
- `ExpEff` a function computing the expected efficacy (value) for a specific dose, based on model parameter(s). The model parameter(s) (samples) are obtained based on pseudo data prior and (if any) with observed responses (see details above)
- `data` refers to the data input specification in `DataDual` class which are used to obtain model paramters estimates or samples (see details above)

See Also

`Effloglog`, `EffFlexi`

`ModelPseudo-class` *Class of models using expressing their prior in form of Pseudo data*

Description

This is the Pseudo model class, from which all models where their prior are expressed in form of pseudo data (as if some data are available before the trial starts) inherit. It also inherits all slots from `AllModels`.No slots for this class

See Also

`LogisticIndepBeta`, `Effloglog`, `EffFlexi`

`ModelTox-class` *No intialization function Class for DLE models using pseudo data prior. This is a class of DLE (dose-limiting events) models/ toxicity model which contains all DLE models for which their prior are specified in form of pseudo data (as if there is some data before the trial starts). It inherits all slots from* `ModelPseudo`

Description

The `dose` function has a first argument `prob`, a scalar a probability of the occurrence of a DLE which is targeted. Additional arguments are models parameters. It computes, using the model parameter(s)/ model parameter(s) samples, the resulting dose. Note that the model parameters are called exactly as in the `model`. The model estimates generated can be single values of the maximum likelihodd estimates (prior or posterior modal estimates) or samples of the model estimates generated. If samples of the model estimates are generated, the model parameters (samples) must be included in the `samples` vector. The vectors of all samples for these model paramters will be supplied to the function such that the function will be able to process vectors of model parameters.

Details

The prob function has a first argument dose, a scalar dose level which is targeted. Additional arguments are model paramters. It computes using model paramter(s) (samples), the resulting probabilities of a DLE occuring at the target dose level. If samples of model parameters are generated, the function must vectorize over the model parameters.

Note that dose and prob are the inverse functions of each other.

The data must obey the covention that the data input is called exactly in the Data class. This refers to any observed DLE responses (y in Data class), the dose (levels) (x in Data class) at which these responses are observed, all dose levels considered in the study (doseGrid in Data) class and other specifications in Data class that can be used to generate prior or posterior modal estimates or samples estimates for model parmater(s). If no responses is observed, at least doseGrid in Data has to be specified in data slot for which prior modal estimates or samples can be obtained for model parameters based on the specified pseudo data.

Slots

dose a function computing the dose level reaching a specific target probabilty of the occurrence of a DLE, based on the model parameters. The model paramters (samples)are obtained based on the prior specified in form of pseudo data and together with (if any) the observed DLE responses and their corresponding dose levels (see details above)

prob a function computing the probability of the occurrence of a DLEat a specidfied dose level, based on the model parameters. The model paramters (samples) are obtained the prior specified in form of pseudo data and together with (if any) the observed DLE responses and their corresponding dose levels (see dtails above)

data refers to the data input specification in Data class which are used to obtain model paramters estimates or samples (see details above)

See Also

LogisticIndepBeta, Effloglog, EffFlexi

---

multiplot *Multiple plot function*

Description

ggplot objects can be passed in ..., or to plotlist (as a list of ggplot objects). If the layout is something like matrix(c(1,2,3,3), nrow=2, byrow=TRUE), then plot 1 will go in the upper left, 2 will go in the upper right, and 3 will go all the way across the bottom.

Usage

```
multiplot(..., plotlist = NULL, rows = 1, layout = NULL)
```

Arguments

`...`	Objects to be passed
`plotlist`	a list of additional objects
`rows`	Number of rows in layout
`layout`	A matrix specifying the layout. If present, `rows` is ignored.

Value

Used for the side effect of plotting

---

nextBest *Find the next best dose*

Description

Compute the recommended next best dose.

Usage

```
nextBest(nextBest, doselimit, samples, model, data, ...)

## S4 method for signature 'NextBestMTD,numeric,Samples,Model,Data'
nextBest(nextBest, doselimit, samples, model, data, ...)

## S4 method for signature 'NextBestNCRM,numeric,Samples,Model,Data'
nextBest(nextBest, doselimit, samples, model, data, ...)

## S4 method for signature 'NextBestNCRM,numeric,Samples,Model,DataParts'
nextBest(nextBest, doselimit, 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, samples, model, data, ...)

## S4 method for signature
## 'NextBestTDsamples,numeric,Samples,LogisticIndepBeta,Data'
nextBest(nextBest, doselimit, samples, model, data, ...)

## S4 method for signature 'NextBestTD,numeric,missing,LogisticIndepBeta,Data'
nextBest(nextBest, doselimit, model, data, SIM = FALSE, ...)

## S4 method for signature 'NextBestMaxGain,numeric,missing,ModelTox,DataDual'
```

```
nextBest(nextBest, doselimit, model, data, Effmodel, SIM = FALSE, ...)

## S4 method for signature
## 'NextBestMaxGainSamples,numeric,Samples,ModelTox,DataDual'
nextBest(
  nextBest,
  doselimit,
  samples,
  model,
  data,
  Effmodel,
  Effsamples,
  SIM = FALSE,
  ...
)
```

Arguments

nextBest	The rule, an object of class NextBest
doselimit	The maximum allowed next dose. If this is an empty (length 0) vector, then no dose limit will be applied in the course of dose recommendation calculation, and a corresponding warning is given.
samples	the Samples object
model	The model input, an object of class Model
data	The data input, an object of class Data
...	possible additional arguments without method dispatch
SIM	internal command to notify if this method is used within simulations. Default as FALSE
Effmodel	the efficacy model of ModelEff class object
Effsamples	the efficacy samples of Samples class object

Details

This function outputs the next best dose recommendation based on the corresponding rule nextBest, the posterior samples from the model and the underlying data.

Value

a list with the next best dose (element value) on the grid defined in data, and a plot depicting this recommendation (element plot). In case of multiple plots also an element singlePlots is included which returns the list of single plots, which allows for further customization of these. Also additional list elements describing the outcome of the rule can be contained.

Functions

- nextBest(nextBest = NextBestMTD, doselimit = numeric, samples = Samples, model = Model, data = Data): Find the next best dose based on the MTD rule

- nextBest(nextBest = NextBestNCRM, doselimit = numeric, samples = Samples, model = Model, data = Data): Find the next best dose based on the NCRM method. 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 = NextBestNCRM, doselimit = numeric, samples = Samples, model = Model, data = DataParts): Find the next best dose based on the NCRM method when two parts trial is used - todo: need an example here for DataParts
- 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 = NextBestTDsamples, doselimit = numeric, samples = Samples, model = LogisticIndepBeta, data = Data): Find the next best dose based on the 'NextBestTD-samples' class rule. This a method based only on the DLE responses and for LogisticIndepBeta model class object involving DLE samples
- nextBest(nextBest = NextBestTD, doselimit = numeric, samples = missing, model = LogisticIndepBeta, data = Data): Find the next best dose based on the 'NextBestTD' class rule. This a method based only on the DLE responses and for LogisticIndepBeta model class object without DLE samples
- nextBest(nextBest = NextBestMaxGain, doselimit = numeric, samples = missing, model = ModelTox, data = DataDual): for slots nextBest,doselimit, data and SIM. This is a function to find the next best dose based on the 'NextBestMaxGain' class rule. This a method based on the DLE responses and efficacy responses without DLE and efficacy samples.
- nextBest(nextBest = NextBestMaxGainSamples, doselimit = numeric, samples = Samples, model = ModelTox, data = DataDual): for slots nextBest,doselimit, data and SIM. This is a function to find the next best dose based on the 'NextBestMaxGainSamples' class rule. This a method based on the DLE responses and efficacy responses with DLE and efficacy samples. Effmodel must be of class Effloglog or EffFlexi.

Examples

```
# 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),
                           refDose=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 'NextBestMTD'
mtdNextBest <- NextBestMTD(target=0.33,
                           derive=
                             function(mtdSamples){
                               quantile(mtdSamples, probs=0.25)
                             })

# Calculate the next best dose
doseRecommendation <- nextBest(mtdNextBest,
                               doselimit=nextMaxDose,
                               samples=samples, model=model, data=data)

# 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),
                           refDose=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),
                           maxOverdoseProb=0.25)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
                               doselimit=nextMaxDose,
                               samples=samples, model=model, data=data)

# Look at the probabilities
doseRecommendation$probs



# 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),
                           refDose=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(dltStart=0,
                                        cleanStart=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),
                           maxOverdoseProb=0.25)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
```

```
                                doselimit=nextMaxDose,
                                samples=samples,
                                model=model,
                                data=data)


# Create the data
data <- Data(x=c(5, 5, 5, 10, 10, 10),
             y=c(0, 0, 0, 0, 1, 0),
             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
myNextBest <- NextBestThreePlusThree()

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
                                data=data)


# 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(mu = c(0, 1),
                        Sigma = 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),
                        smooth = "RW1")

# 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),
                                   maxOverdoseProb=0.25)

# Calculate the next best dose
doseRecommendation <- nextBest(myNextBest,
                               doselimit=nextMaxDose,
                               samples=samples,
                               model=model,
                               data=data)

## joint plot
print(doseRecommendation$plot)

## show customization of single plot
variant1 <- doseRecommendation$singlePlots$plot1 + xlim(0, 20)
print(variant1)

## 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))
##The 'nextBest' method using NextBestTDsamples' rules class object
## That is dose-esclation procedure using the 'logisticIndepBeta' DLE model involving DLE samples
## model must be of 'LogisticIndepBeta' class
model<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=data)

##Define the options for MCMC
options <- McmcOptions(burnin=100,step=2,samples=1000)
##Then genreate the samples
samples <- mcmc(data, model, options)

##target probabilities of the occurrence of a DLE during trial and at the end of trial are
## defined as 0.35 and 0.3, respectively
##Specified in 'derive' such that the 30% posterior quantile of the TD35 and TD30 samples
## will be used as TD35 and TD30 estimates
tdNextBest<-NextBestTDsamples(targetDuringTrial=0.35,targetEndOfTrial=0.3,
                              derive=function(TDsamples){quantile(TDsamples,probs=0.3)})

##doselimit is the maximum allowable dose level to be given to subjects
RecommendDose<-nextBest(tdNextBest,doselimit=max(data@doseGrid),samples=samples,
                        model=model,data=data)
## 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))
##The 'nextBest' method using NextBestTD' rules class object
## That is dose-esclation procedure using the 'logisticIndepBeta' DLE model involving DLE samples
```

```
## model must be of 'LogisticIndepBeta' class
model<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=data)
##target probabilities of the occurrence of a DLE during trial and at the end of trial
## are defined as 0.35 and 0.3, respectively
tdNextBest<-NextBestTD(targetDuringTrial=0.35,targetEndOfTrial=0.3)

##doselimit is the maximum allowable dose level to be given to subjects
RecommendDose<- nextBest(tdNextBest,
              doselimit=max(data@doseGrid),
              model=model,
              data=data)
## 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)

##The 'nextBest' method using NextBestMaxGain' rules class object
## using the 'ModelTox' class DLE model
## DLEmodel e.g 'LogisticIndepBeta' class
DLEmodel<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=data)

## using the 'ModelEff' class efficacy model
## Effmodel e.g 'Effloglog' class
Effmodel<-Effloglog(Eff=c(1.223,2.513),Effdose=c(25,300),nu=c(a=1,b=0.025),data=data,c=0)

##target probabilities of the occurrence of a DLE during trial and at the
## end of trial are defined as
## 0.35 and 0.3, respectively
mynextbest<-NextBestMaxGain(DLEDuringTrialtarget=0.35,DLEEndOfTrialtarget=0.3)

##doselimit is the maximum allowable dose level to be given to subjects
RecommendDose<-nextBest(mynextbest,doselimit=300,model=DLEmodel,Effmodel=Effmodel,data=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),placebo=FALSE)
##The 'nextBest' method using NextBestMaxGainSamples' rules class object
## using the 'ModelTox' class DLE model
## DLEmodel e.g 'LogisticIndepBeta' class
DLEmodel<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=data)

## using the 'ModelEff' class efficacy model
## Effmodel e.g 'Effloglog' class
Effmodel<-Effloglog(c(1.223,2.513),c(25,300),nu=c(a=1,b=0.025),data=data,c=0)
##DLE and efficacy samples must be of 'Samples' Class
DLEsamples<-mcmc(data,DLEmodel,options)
Effsamples<-mcmc(data,Effmodel,options)

##target probabilities of the occurrence of a DLE during trial and at the end of trial
## are defined as 0.35 and 0.3, respectively
## Using 30% posterior quantile of the TD35 and TD30 samples as estimates of TD35 and TD30,
## function specified in TDderive slot
```

```
## Using the 50% posterior quantile of the Gstar (the dose which gives the maxim gain value)
## samples as Gstar estimate,function specified in Gstarderive slot
mynextbest<-NextBestMaxGainSamples(DLEDuringTrialtarget=0.35,
                                   DLEEndOfTrialtarget=0.3,
                                   TDderive=function(TDsamples){
                                     quantile(TDsamples,prob=0.3)},
                                   Gstarderive=function(Gstarsamples){
                                     quantile(Gstarsamples,prob=0.5)})

RecommendDose<-nextBest(mynextbest,doselimit=max(data@doseGrid),samples=DLEsamples,model=DLEmodel,
                        data=data,Effmodel=Effmodel,Effsamples=Effsamples)

## now using the 'EffFlexi' class efficacy model:

##The 'nextBest' method using NextBestMaxGainSamples' rules class object for 'EffFlexi' model class
## using the 'ModelTox' class DLE model
## DLEmodel e.g 'LogisticIndepBeta' class
Effmodel<- EffFlexi(Eff=c(1.223, 2.513),Effdose=c(25,300),
                    sigma2=c(a=0.1,b=0.1),
                    sigma2betaW=c(a=20,b=50),smooth="RW2",data=data)

##DLE and efficacy samples must be of 'Samples' Class
DLEsamples<-mcmc(data,DLEmodel,options)
Effsamples<-mcmc(data,Effmodel,options)

##target probabilities of the occurrence of a DLE during trial and at the
## end of trial are defined as 0.35 and 0.3, respectively
## Using 30% posterior quantile of the TD35 and TD30 samples as estimates of
## TD35 and TD30, function specified in TDderive slot
## Using the 50% posterio quantile of the Gstar (the dose which gives the maximum
## gain value) samples as Gstar estimate,function specified in Gstarderive slot
mynextbest<-NextBestMaxGainSamples(DLEDuringTrialtarget=0.35,
                                   DLEEndOfTrialtarget=0.3,
                                   TDderive=function(TDsamples){
                                     quantile(TDsamples,prob=0.3)},
                                   Gstarderive=function(Gstarsamples){
                                     quantile(Gstarsamples,prob=0.5)})

RecommendDose<-nextBest(mynextbest,doselimit=max(data@doseGrid),samples=DLEsamples,
                        model=DLEmodel,
                        data=data,Effmodel=Effmodel,Effsamples=Effsamples)
```

`NextBest-class` *The virtual class for finding next best dose*

Description

The virtual class for finding next best dose

See Also

NextBestMTD, NextBestNCRM, NextBestDualEndpoint, NextBestThreePlusThree

---

NextBestDualEndpoint *Initialization function for "NextBestDualEndpoint"*

Description

Initialization function for "NextBestDualEndpoint"

Usage

```
NextBestDualEndpoint(
  target,
  scale = c("relative", "absolute"),
  overdose,
  maxOverdoseProb,
  targetThresh = 0.01
)
```

Arguments

target	see NextBestDualEndpoint
scale	see NextBestDualEndpoint
overdose	see NextBestDualEndpoint
maxOverdoseProb	see NextBestDualEndpoint
targetThresh	see NextBestDualEndpoint

Value

the NextBestDualEndpoint object

NextBestDualEndpoint-class *The class with the input for finding the next dose based on the dual endpoint model*

Description

This rule first excludes all doses that exceed the probability maxOverdoseProb of having an overdose toxicity, as specified by the overdose interval overdose. Then, it picks under the remaining admissible doses the one that maximizes the probability to be in the target biomarker range, by default relative to the maximum biomarker level across the dose grid or relative to the Emax parameter in case a parametric model was selected (e.g. DualEndpointBeta, DualEndpointEmax)) However, is scale is set to "absolute" then the natural absolute biomarker scale can be used to set a target.

Slots

`target` the biomarker target range, that needs to be reached. For example, (0.8, 1.0) and `scale="relative"` means 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.

`scale` either `relative` (default, then the `target` is interpreted relative to the maximum, so must be a probability range) or `absolute` (then the `target` is interpreted as absolute biomarker range)

`overdose` the overdose toxicity interval (lower limit excluded, upper limit included)

`maxOverdoseProb` maximum overdose probability that is allowed

`targetThresh` which target probability threshold needs to be fulfilled before the target probability will be used for deriving the next best dose (default: 0.01)

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)

myNextBest <- NextBestDualEndpoint(target=c(0.9, 1),
                                   overdose=c(0.35, 1),
                                   maxOverdoseProb=0.25)

## now do it with an absolute target on the natural biomarker scale:

myNextBest <- NextBestDualEndpoint(target=c(200, 300),
                                   scale="absolute",
                                   overdose=c(0.35, 1),
                                   maxOverdoseProb=0.25)
```

`NextBestMaxGain` *Initialization function for the class 'NextBestMaxGain'*

Description

Initialization function for the class 'NextBestMaxGain'

Usage

```
NextBestMaxGain(DLEDuringTrialtarget, DLEEndOfTrialtarget)
```

Arguments

`DLEDuringTrialtarget`
: please refer to `NextBestMaxGain` class object

`DLEEndOfTrialtarget`
: please refer to `NextBestMaxGain` class object

Value

the NextBestMaxGain class object

NextBestMaxGain-class *Next best dose with maximum gain value based on a pseudo DLE and efficacy model without samples*

Description

This is a class for which to find the next dose which is safe and give the maximum gain value for allocation. This is a class where no DLE and efficacy samples are involved. This is only based on the probabilities of the occurrence of a DLE and the values of the mean efficacy responses obtained by using the modal estimates of the DLE and efficacy model parameters. There are two inputs which are the two target probabilities of the occurrence of a DLE used during trial and used at the end of trial, for finding the next best dose that is safe and gives the maximum gain value and the dose to recommend at the end of a trial. This is only suitable to use with DLE models specified in 'ModelTox' class and efficacy models specified in 'ModelEff' (except 'EffFlexi' model) class

Slots

DLEDuringTrialtarget the target probability of the occurrrence of a DLE to be used during the trial

DLEEndOfTrialtarget the target probability of the occurrence of a DLE to be used at the end of the trial. This target is particularly used to recommend the dose for which its posterior probability of the occurrence of a DLE is equal to this target

Examples

```
##define the NextBestMaxGain class (no samples and based a pseudo DLE model and a efficacy model)
##specified the target probability of the occurrence of a DLE during the trial be 0.35
##specified the target probability of the occurrence of a DLE at the end of trial be 0.3

myNextBest <-NextBestMaxGain(DLEDuringTrialtarget=0.35,
                             DLEEndOfTrialtarget=0.3)
```

NextBestMaxGainSamples *Initialization function for class "NextBestMaxGainSamples"*

Description

Initialization function for class "NextBestMaxGainSamples"

Usage

```
NextBestMaxGainSamples(
  DLEDuringTrialtarget,
  DLEEndOfTrialtarget,
  TDderive,
  Gstarderive
)
```

Arguments

DLEDuringTrialtarget	please refer to NextBestMaxGainSamples class object
DLEEndOfTrialtarget	please refer to NextBestMaxGainSamples class object
TDderive	please refer to NextBestMaxGainSamples class object
Gstarderive	please refer to NextBestMaxGainSamples class object

Value

the NextBestMaxGainSamples class object

NextBestMaxGainSamples-class

Next best dose with maximum gain value based on a pseudo DLE and efficacy model with samples

---

Description

This is a class for which to find the next dose which is safe and give the maximum gain value for allocation. This is a class where DLE and efficacy samples are involved. There are two inputs which are the two target probabilities of the occurrence of a DLE used during trial and used at the end of trial, for finding the next best dose that is safe and gives the maximum gain value and the dose to recommend at the end of a trial. This is only suitable to use with DLE models specified in 'ModelTox' class and efficacy models specified in 'ModelEff' class class

Slots

DLEDuringTrialtarget the target probability of the occurrrence of a DLE to be used during the trial

DLEEndOfTrialtarget the target probability of the occurrence of a DLE to be used at the end of the trial. This target is particularly used to recommend the dose for which its posterior probability of the occurrence of a DLE is equal to this target

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.

`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.

Examples

```
##define the NextBestMaxGainsamples class
##specified the target probability of the occurrence of a DLE during the trial be 0.35
##specified the target probability of the occurrence of a DLE at the end of trial be 0.3
## we want the use the 30% posterior quantile (the 30th percentaile) of the TD35
## (the dose level with probability of DLE equals 0.35) and TD30 samples
## For Gstar (the dose which gives tha maximum
##gain) samples, we will use the 50% posterio quantile (the median or th 50th percentile)
## of the Gstar sample
##A function is then defined in the 'TDderive' slot for the TD30 and TD35 samples
## and another function is defined in the 'Gstarderive' slot for Gstar samples
mynextbest<-NextBestMaxGainSamples(DLEDuringTrialtarget=0.35,
                                   DLEEndOfTrialtarget=0.3,
                                   TDderive=function(TDsamples){
                                     quantile(TDsamples,prob=0.3)},
                                   Gstarderive=function(Gstarsamples){
                                     quantile(Gstarsamples,prob=0.5)})
```

`NextBestMTD` *Initialization function for class "NextBestMTD"*

Description

Initialization function for class "NextBestMTD"

Usage

```
NextBestMTD(target, derive)
```

Arguments

`target`	see `NextBestMTD`
`derive`	see `NextBestMTD`

Value

the `NextBestMTD` object

NextBestMTD-class *The class with the input for finding the next best MTD estimate*

Description

The class with the input for finding the next best MTD estimate

Slots

`target` the target toxicity probability

`derive` the function which derives from the input, a vector of posterior MTD samples called `mtdSamples`, the final next best MTD estimate.

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.
mtdNextBest <- NextBestMTD(target=0.33,
                           derive=
                             function(mtdSamples){
                               quantile(mtdSamples, probs=0.25)
                             })
```

NextBestNCRM *Initialization function for "NextBestNCRM"*

Description

Initialization function for "NextBestNCRM"

Usage

```
NextBestNCRM(target, overdose, maxOverdoseProb)
```

Arguments

`target`	see `NextBestNCRM`
`overdose`	see `NextBestNCRM`
`maxOverdoseProb`	see `NextBestNCRM`

Value

the `NextBestNCRM` object

NextBestNCRM-class *The class with the input for finding the next dose in target interval*

Description

Note that 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 `maxOverdoseProb`. 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` the target toxicity interval (limits included)

`overdose` the overdose toxicity interval (lower limit excluded, upper limit included)

`maxOverdoseProb` maximum overdose probability that is allowed

Examples

```
# In the example below, the target toxiciy 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%.
myNextBest <- NextBestNCRM(target=c(0.2, 0.35),
                           overdose=c(0.35, 1),
                           maxOverdoseProb=0.25)
```

NextBestTD *Initialization function for the class "NextBestTD"*

Description

Initialization function for the class "NextBestTD"

Usage

```
NextBestTD(targetDuringTrial, targetEndOfTrial)
```

Arguments

`targetDuringTrial`
: please refer to `NextBestTD` class object

`targetEndOfTrial`
: please refer to `NextBestTD` class object

Value

the NextBestTD class object

NextBestTD-class *Next best dose based on Pseudo DLE model without sample*

Description

The class is to find the next best dose for allocation and the dose for final recommendation at the end of a trial without involving any samples. This is a class for which only DLE response will be incorporated for the dose-allocation. This is only based on the probabilities of the occurrence of a DLE obtained by using the modal estimates of the model paramters. There are two inputs inputs which are the two target probabilities of the occurrence of a DLE used during trial and used at the end of trial, for finding the next best dose for allocation and the dose for recommendation at the end of the trial. It is only suitable to use with the model specified in ModelTox class.

Slots

`targetDuringTrial` the target probability of the occurrrence of a DLE to be used during the trial

`targetEndOfTrial` the target probability of the occurrence of a DLE to be used at the end of the trial. This target is particularly used to recommend the dose for which its posterior probability of the occurrence of a DLE is equal to this target

Examples

```
##define the NextBestTD class (no samples and based a pseudo DLE model)
##specified the target probability of the occurrence of a DLE during the trial be 0.35
##specified the target probability of the occurrence of a DLE at the end of trial be 0.3

myNextBest <-NextBestTD(targetDuringTrial=0.35,
                        targetEndOfTrial=0.3)
```

NextBestTDsamples *Initialization function for class "NextBestTDsamples"*

Description

Initialization function for class "NextBestTDsamples"

Usage

```
NextBestTDsamples(targetDuringTrial, targetEndOfTrial, derive)
```

Arguments

`targetDuringTrial`
: please refer to `NextBestTDsamples` class object

`targetEndOfTrial`
: please refer to `NextBestTDsamples` class object

`derive`
: please refer to `NextBestTDsamples` class object

Value

the `NextBestTDsamples` class object

NextBestTDsamples-class

Next best dose based on Pseudo DLE Model with samples

Description

The class is to find the next best dose for allocation and the dose for final recommendation at the end of a trial. There are two input target probabilities of the occurrence of a DLE used during trial and used at the end of trial to find the two doses. For this class, only DLE response will be incorporated for the dose allocation and DLEsamples must be used to obtain the next dose for allocation.

Slots

`targetDuringTrial` the target probability of the occurrrence of a DLE to be used during the trial

`targetEndOfTrial` the target probability of the occurrence of a DLE to be used at the end of the trial. This target is particularly used to recommend the dose at the end of a trial for which its posterior probability of the occurrence of a DLE is equal to this target

`derive` 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.

Examples

```
##define the NextBestTDsamples class
##specified the target probability of the occurrence of a DLE during the trial be 0.35
##specified the target probability of the occurrence of a DLE at the end of trial be 0.3
## we want the use the 30% posterior quantile (the 30th percentaile) of the TD35 (the dose
## level with probability of DLE equals 0.35) and TD30 samples. A function is then defined
## in the 'derive' slot
myNextBest <-NextBestTDsamples(targetDuringTrial=0.35,
                               targetEndOfTrial=0.3,
                              derive=function(TDsamples){quantile(TDsamples,probs=0.3)})
```

NextBestThreePlusThree

Initialization function for "NextBestThreePlusThree"

Description

Initialization function for "NextBestThreePlusThree"

Usage

```
NextBestThreePlusThree()
```

Value

the NextBestThreePlusThree object

NextBestThreePlusThree-class

The class with the input for finding the next dose in target interval

Description

Implements the classical 3+3 dose recommendation. No input is required, hence this class has no slots.

Examples

```
## Choose the next best dose using the  classical 3+3 design.
## No input is required, hence this function has no parameters.
myNextBest <- NextBestThreePlusThree()
```

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,missing-method

Plot method for the "Data" class

Description

Plot method for the "Data" class

Usage

```
## S4 method for signature 'Data,missing'
plot(x, y, blind = FALSE, ...)
```

Arguments

x	the Data object we want to plot
y	missing
blind	Logical (default FALSE) if to blind the data. If TRUE, then placebo subjects are reported by the active dose level of the corresponding cohort and DLEs are always assigned to the firsts subjects.
...	not used

Value

the ggplot object

Examples

```
# Create some data of class 'Data'
myData <- 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)))

# Plot the data
plot(myData)
```

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),
           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,DataDual,missing-method

Plot method for the "DataDual" class

Description

Plot method for the "DataDual" class

Usage

```
## S4 method for signature 'DataDual,missing'
plot(x, y, blind = FALSE, ...)
```

Arguments

x	the DataDual object we want to plot
y	missing
blind	Logical (default FALSE) if to blind the data
...	not used

Value

the ggplot object

Examples

```
# Create some data of class 'DataDual'
myData <- 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
#grid.arrange(plot(myData))

plot(myData)
```

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

```
##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),Effdose=c(25,300),nu=c(a=1,b=0.025),data=data,c=0)
## plot the dose-efficacy curve
## 'x' is the data and 'y' is the model in plot
plot(x=data,y=Effmodel)
```

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 gtable 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
model <- DualEndpointRW(mu = c(0, 1),
                        Sigma = 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),
                        smooth="RW1")

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

# Choose the rule for the cohort-size
mySize1 <- CohortSizeRange(intervals=c(0, 30),
```

```
                            cohortSize=c(1, 3))
mySize2 <- CohortSizeDLT(DLTintervals=c(0, 1),
                         cohortSize=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)

# 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,
                     cohortSize = CohortSizeConst(3),
                     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
##Also for illustration purpose, we will use 5 burn-ins to generate 20 samples
# this should be increased of course
mySims <- simulate(design,
                   trueTox=trueTox,
                   trueBiomarker=trueBiomarker,
```

```
                    sigma2W=0.01,
                    rho=0,
                    nsim=1,
                    parallel=FALSE,
                    seed=3,
                    startingDose=6,
                    mcmcOptions =
                      McmcOptions(burnin=5,
                                  step=1,
                                  samples=20))

# Plot the results of the simulation
print(plot(mySims))
```

---

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 gtable 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
model <- DualEndpointRW(mu = c(0, 1),
                        Sigma = 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),
                        smooth="RW1")

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

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

# Choose the rule for stopping
myStopping4 <- StoppingTargetBiomarker(target=c(0.9, 1),
                                       prob=0.5)
# only 10 patients here for illustration!
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,
                     cohortSize = CohortSizeConst(3),
                     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
##For illustration purpose we will use 5 burn-ins to generate 20 samples
# this should be increased of course
mySims <- simulate(design,
                   trueTox=trueTox,
                   trueBiomarker=trueBiomarker,
                   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(mySims,
             trueTox = trueTox,
             trueBiomarker = trueBiomarker))
```

---

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 gtable object.

Examples

```
##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(targetDuringTrial=0.35,
                         targetEndOfTrial=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,
                   cohortSize=mySize,
                   data=data,startingDose=25)

##Specify the truth of the DLE responses
myTruth <- function(dose)
{ model@prob(dose, 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(targetDuringTrial=0.35,
                              targetEndOfTrial=0.3,
                              derive=function(TDsamples){quantile(TDsamples,probs=0.3)})
##specify the design
design <- TDsamplesDesign(model=model,
                          nextBest=tdNextBest,
                          stopping=myStopping,
                          increments=myIncrements,
                          cohortSize=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))
#
```

---

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 gtable object.

plot,PseudoDualFlexiSimulations,missing-method

Plot for PseudoDualFlexiSimulations

Description

This plot method can be applied to PseudoDualFlexiSimulations 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

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 gtable object.

Examples

```
##obtain the plot for the simulation results
##If DLE and efficacy responses are considered in the simulations


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)

##The efficacy model must be of 'EffFlexi' class
```

```
Effmodel<- EffFlexi(Eff=c(1.223, 2.513),Effdose=c(25,300),
                  sigma2=c(a=0.1,b=0.1),sigma2betaW=c(a=20,b=50),smooth="RW2",data=data)

##The escalation rule using the 'NextBestMaxGainSamples' class
mynextbest<-NextBestMaxGainSamples(DLEDuringTrialtarget=0.35,
                                   DLEEndOfTrialtarget=0.3,
                                   TDderive=function(TDsamples){
                                     quantile(TDsamples,prob=0.3)},
                                   Gstarderive=function(Gstarsamples){
                                     quantile(Gstarsamples,prob=0.5)})

## 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
design <- DualResponsesSamplesDesign(nextBest=mynextbest,
                                     cohortSize=mySize,
                                     startingDose=25,
                                     model=DLEmodel,
                                     Effmodel=Effmodel,
                                     data=data,
                                     stopping=myStopping,
                                     increments=myIncrements)
##specified the true DLE curve and the true expected efficacy values at all dose levels
myTruthDLE<- function(dose)
{ DLEmodel@prob(dose, 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
myTruthGain <- function(dose)
{return((myTruthEff(dose))/(1+(myTruthDLE(dose)/(1-myTruthDLE(dose)))))}


##options for MCMC
options<-McmcOptions(burnin=10,step=1,samples=20)
##The simulations
##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,
                 mcmcOptions=options,
                 nsim=1,
                 seed=819,
                 parallel=FALSE)
##plot this simulated results
print(plot(mySim))
```

`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 `gtable` object.

Examples

```
##obtain the plot for the simulation results
##If DLE and efficacy responses are considered in the simulations
##Specified your simulations when no 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),Effdose=c(25,300),
                    nu=c(a=1,b=0.025),data=data,c=0)

##The escalation rule using the 'NextBestMaxGain' class
mynextbest<-NextBestMaxGain(DLEDuringTrialtarget=0.35,
                            DLEEndOfTrialtarget=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,
                              Effmodel=Effmodel,
                              stopping=myStopping,
                              increments=myIncrements,
                              cohortSize=mySize,
                              data=data,startingDose=25)
##Specify the true DLE and efficacy curves
myTruthDLE<- function(dose)
{ DLEmodel@prob(dose, phi1=-53.66584, phi2=10.50499)
}

myTruthEff<- function(dose)
{Effmodel@ExpEff(dose,theta1=-4.818429,theta2=3.653058)
}

## Then specified the simulations and generate the trial
##For illustration purpose only 1 simulation is produced (nsim=1).
mySim <-simulate(object=design,
```

```
                  args=NULL,
                  trueDLE=myTruthDLE,
                  trueEff=myTruthEff,
                  trueNu=1/0.025,
                  nsim=1,
                  seed=819,
                  parallel=FALSE)

##plot the simulation results
print(plot(mySim))

##If DLE and efficacy samples are involved
##The escalation rule using the 'NextBestMaxGainSamples' class
mynextbest<-NextBestMaxGainSamples(DLEDuringTrialtarget=0.35,
                                   DLEEndOfTrialtarget=0.3,
                                   TDderive=function(TDsamples){
                                     quantile(TDsamples,prob=0.3)},
                                   Gstarderive=function(Gstarsamples){
                                     quantile(Gstarsamples,prob=0.5)})
##The design of 'DualResponsesSamplesDesign' class
design <- DualResponsesSamplesDesign(nextBest=mynextbest,
                                     cohortSize=mySize,
                                     startingDose=25,
                                     model=DLEmodel,
                                     Effmodel=Effmodel,
                                     data=data,
                                     stopping=myStopping,
                                     increments=myIncrements)
##options for MCMC
options<-McmcOptions(burnin=10,step=1,samples=20)
##The simulations
##For illustration purpose only 1 simulation is produced (nsim=1).
mySim<-simulate(design,
                args=NULL,
                trueDLE=myTruthDLE,
                trueEff=myTruthEff,
                trueNu=1/0.025,
                nsim=1,
                mcmcOptions=options,
                seed=819,
                parallel=FALSE)
##plot the simulation results
print(plot(mySim))
```

---

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 ploted. 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 gtable object.

Examples

```
##obtain the plot of the summary for the simulation results
##If DLE and efficacy responses are considered in the simulations
##Specified your simulations when no 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),Effdose=c(25,300),
                    nu=c(a=1,b=0.025),data=data,c=0)

##The escalation rule using the 'NextBestMaxGain' class
mynextbest<-NextBestMaxGain(DLEDuringTrialtarget=0.35,
                            DLEEndOfTrialtarget=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 10 subjects are treated (for illustration)
myStopping <- StoppingMinPatients(nPatients=10)
##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,
                              Effmodel=Effmodel,
                              stopping=myStopping,
                              increments=myIncrements,
                              cohortSize=mySize,
                              data=data,startingDose=25)
##Specify the true DLE and efficacy curves
myTruthDLE<- function(dose)
{ DLEmodel@prob(dose, phi1=-53.66584, phi2=10.50499)
}

myTruthEff<- function(dose)
{Effmodel@ExpEff(dose,theta1=-4.818429,theta2=3.653058)
}

## Then specified the simulations and generate the trial
##For illustration purpose only 1 simulation is produced (nsim=1).
mySim <-simulate(object=design,
                 args=NULL,
                 trueDLE=myTruthDLE,
                 trueEff=myTruthEff,
                 trueNu=1/0.025,
                 nsim=1,
                 ## this would need to be increased in the real
                 ## application:
                 mcmcOptions=McmcOptions(burnin=10, step=1, samples=50),
                 seed=819,
                 parallel=FALSE)

##Then produce a summary of your simulations
MYSUM <- summary(mySim,
                 trueDLE=myTruthDLE,
                 trueEff=myTruthEff)
```

```
##Then plot the summary of the simulations
print(plot(MYSUM))



##If DLE and efficacy samples are involved
##Please refer to design-method 'simulate DualResponsesSamplesDesign' examples for details
##specified the next best
mynextbest<-NextBestMaxGainSamples(DLEDuringTrialtarget=0.35,
                                   DLEEndOfTrialtarget=0.3,
                                   TDderive=function(TDsamples){
                                     quantile(TDsamples,prob=0.3)},
                                   Gstarderive=function(Gstarsamples){
                                     quantile(Gstarsamples,prob=0.5)})
##specified the design
design <- DualResponsesSamplesDesign(nextBest=mynextbest,
                                     cohortSize=mySize,
                                     startingDose=25,
                                     model=DLEmodel,
                                     Effmodel=Effmodel,
                                     data=data,
                                     stopping=myStopping,
                                     increments=myIncrements)
##options for MCMC
##for illustration purpose we use 10 burn-in and generate 50 samples
options<-McmcOptions(burnin=10,step=2,samples=50)
##The simulations
##For illustration purpose only 1 simulation is produced (nsim=1).
# mySim<-simulate(design,
#                 args=NULL,
#                 trueDLE=myTruthDLE,
#                 trueEff=myTruthEff,
#                 trueNu=1/0.025,
#                 nsim=1,
#                 mcmcOptions=options,
#                 seed=819,
#                 parallel=FALSE)
#
# ##Then produce a summary of your simulations
# MYSUM <- summary(mySim,
#                  trueDLE=myTruthDLE,
#                  trueEff=myTruthEff)
#
# ##Then plot the summary of the simulations
# print(plot(MYSUM))



##OR if the 'EffFlexi' class is used
## for the efficacy model

Effmodel<- EffFlexi(Eff=c(1.223, 2.513),Effdose=c(25,300),
```

```
                  sigma2=c(a=0.1,b=0.1),sigma2betaW=c(a=20,b=50),smooth="RW2",data=data)

##Specified the design
design <- DualResponsesSamplesDesign(nextBest=mynextbest,
                                     cohortSize=mySize,
                                     startingDose=25,
                                     model=DLEmodel,
                                     Effmodel=Effmodel,
                                     data=data,
                                     stopping=myStopping,
                                     increments=myIncrements)
##specified the true DLE curve and the true expected efficacy values at all dose levels
myTruthDLE<- function(dose)
{ DLEmodel@prob(dose, 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
myTruthGain <- function(dose)
{return((myTruthEff(dose))/(1+(myTruthDLE(dose)/(1-myTruthDLE(dose)))))}

##The simulations
# ##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,
#                 mcmcOptions=options,
#                 seed=819,
#                 parallel=FALSE)
# ##Then produce a summary of your simulations
# MYSUM <- summary(mySim,
#                  trueDLE=myTruthDLE,
#                  trueEff=myTruthEff)
#
# ##Then plot the summary of the simulations
# print(plot(MYSUM))
```

`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 gtable object.

Examples

```
##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(targetDuringTrial=0.35,
                         targetEndOfTrial=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,
                   cohortSize=mySize,
                  data=data,startingDose=25)

##Specify the truth of the DLE responses
myTruth <- function(dose)
{ model@prob(dose, 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(targetDuringTrial=0.35,
                              targetEndOfTrial=0.3,
                              derive=function(TDsamples){quantile(TDsamples,probs=0.3)})
##specify the design
design <- TDsamplesDesign(model=model,
                          nextBest=tdNextBest,
                          stopping=myStopping,
                          increments=myIncrements,
                          cohortSize=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))
```

---

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,Model-method

Value

This returns the ggplot object with the dose-toxicity and dose-biomarker model fits

Examples

```
# 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(mu = c(0, 1),
                        Sigma = 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),
                        smooth = "RW1")

# 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)
```

`plot,Samples,Model-method`

Plotting dose-toxicity model fits

Description

Plotting dose-toxicity model fits

Usage

```
## S4 method for signature 'Samples,Model'
plot(
  x,
  y,
  data,
  ...,
  xlab = "Dose level",
  ylab = "Probability of DLT [%]",
  showLegend = TRUE
)
```

Arguments

x	the `Samples` object
y	the `Model` 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

```
# 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),
                           refDose = 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)
```

---

plot,Samples,ModelEff-method

Plot the fitted dose-effcacy curve using a model from ModelEff *class with samples*

Description

Plot the fitted dose-effcacy 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

```
## 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),Effdose=c(25,300),nu=c(a=1,b=0.025),data=data,c=0)
## 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)
```

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 gtable object.

Examples

```
# 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),
                           refDose=56)

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

# Choose the rule for the cohort-size
mySize1 <- CohortSizeRange(intervals=c(0, 30),
                           cohortSize=c(1, 3))
mySize2 <- CohortSizeDLT(DLTintervals=c(0, 1),
                         cohortSize=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,
                 cohortSize=mySize,
                 data=emptydata,
                 startingDose=3)

## define the true function
myTruth <- function(dose)
{
  model@prob(dose, 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))
```

`plot.gtable` *Plots gtable objects*

Description

Plots gtable objects

Usage

```
## S3 method for class 'gtable'
plot(x, ...)
```

Arguments

`x`	the gtable object
`...`	additional parameters for `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,Model-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 todo
- 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

```
## 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),Effdose=c(25,300),nu=c(a=1,b=0.025),data=data,c=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)
##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)
## 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),Effdose=c(25,300),nu=c(a=1,b=0.025),data=data,c=0)
##plot the dose-DLE and dose-efficacy curves with two plot side by side.
plotDualResponses(DLEmodel=DLEmodel,
                  Effmodel=Effmodel,
                  data=data)
```

`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, DLEsamples, Effmodel, Effsamples, data, ...)
```

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

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

```
## 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),Effdose=c(25,300),nu=c(a=1,b=0.025),data=data,c=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)
## 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),Effdose=c(25,300),nu=c(a=1,b=0.025),data=data,c=0)
##plot the three curves of using modal estimates of model parameters at all dose levels
plotGain(DLEmodel=DLEmodel,
         Effmodel=Effmodel,
         data=data)
```

---

prob *Compute the probability for a given dose, given model and samples*

Description

Compute the probability for a given dose, given model and samples

Usage

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

## S4 method for signature 'numeric,Model,Samples'
prob(dose, model, samples, ...)

## S4 method for signature 'numeric,ModelTox,Samples'
prob(dose, model, samples, ...)
```

```
## S4 method for signature 'numeric,ModelTox,missing'
prob(dose, model, samples, ...)
```

Arguments

dose	the dose
model	the Model object
samples	the Samples
...	unused

Value

the vector (for Model objects) of probability samples.

Functions

- prob(dose = numeric, model = ModelTox, samples = Samples): Compute the probability for a given dose, given Pseudo DLE model and samples
- prob(dose = numeric, model = ModelTox, samples = missing): Compute the probability for a given dose, given Pseudo DLE model without samples

Examples

```
# 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),
                           refDose=56)

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

# posterior for Prob(DLT | dose=50)
tox.prob <- prob(dose=50, model=model, samples=samples)




# create data from the '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))

## Initialize a model 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)

options <- McmcOptions(burnin=100, step=2, samples=200)
DLEsamples <- mcmc(data=data,model=DLEmodel,options=options)

tox.prob <- prob(dose=100, model = DLEmodel, samples = DLEsamples)



# create data from the '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))

## Initialize a model 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)

tox.prob <- prob(dose=100, model = DLEmodel)
```

`probit` *Shorthand for probit function*

Description

Shorthand for probit function

Usage

```
probit(x)
```

Arguments

x	the function argument

Value

the probit(x)

ProbitLogNormal *Initialization function for the "ProbitLogNormal" class*

Description

Initialization function for the "ProbitLogNormal" class

Usage

```
ProbitLogNormal(mu, Sigma, refDose = 1, useLogDose = FALSE)
```

Arguments

mu	the prior mean vector
Sigma	the prior covariance matrix
refDose	the reference dose x^*, default 1 (no standardization)
useLogDose	should the log of (standardized) dose be used? (not default)

Value

the ProbitLogNormal object

ProbitLogNormal-class *Probit model with bivariate log normal prior*

Description

This is probit regression model with a bivariate normal prior on the intercept and log slope. The covariate is the dose x itself, potentially divided by a reference dose x^*, or the logarithm of it:

Details

$$probit[p(x)] = \alpha + \beta \cdot x/x^*$$

or

$$probit[p(x)] = \alpha + \beta \cdot \log(x/x^*)$$

in case that the option useLogDose is TRUE. Here $p(x)$ is the probability of observing a DLT for a given dose x.

The prior is

$$(\alpha, \log(\beta)) \sim Normal(\mu, \Sigma)$$

The slots of this class contain the mean vector and the covariance matrix of the bivariate normal distribution, as well as the reference dose. Note that the parametrization inside the class uses alpha0 and alpha1.

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.

Slots

`mu` the prior mean vector μ

`Sigma` the prior covariance matrix Σ

`refDose` the reference dose x^*

`useLogDose` should the log of (standardized) dose be used?

Examples

```
model <- ProbitLogNormal(mu = c(-0.85, 1),
                          Sigma = matrix(c(1, -0.5, -0.5, 1), nrow = 2))

## we can also specify a reference dose, and use a log transformation of
## standardized dose in the model:
model <- ProbitLogNormal(mu = c(-0.85, 1),
                         Sigma = matrix(c(1, -0.5, -0.5, 1), nrow = 2),
                         refDose = 7.2,
                         useLogDose=TRUE)
```

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.

Slots

`sigma2betaWest` the vector of the final posterior mean sigma2betaW estimates

PseudoDualSimulations *Initialization function for 'DualPseudoSimulations' class*

Description

Initialization function for 'DualPseudoSimulations' class

Usage

```
PseudoDualSimulations(
  fitEff,
  FinalGstarEstimates,
  FinalGstarAtDoseGrid,
  FinalGstarCIs,
  FinalGstarRatios,
  FinalOptimalDose,
  FinalOptimalDoseAtDoseGrid,
  sigma2est,
  ...
)
```

Arguments

- `fitEff` please refer to `PseudoDualSimulations` class object
- `FinalGstarEstimates` please refer to `PseudoDualSimulations` class object
- `FinalGstarAtDoseGrid` please refer to `PseudoDualSimulations` class object
- `FinalGstarCIs` please refer to `PseudoDualSimulations` class object
- `FinalGstarRatios` please refer to `PseudoDualSimulations` class object
- `FinalOptimalDose` please refer to `PseudoDualSimulations` class object
- `FinalOptimalDoseAtDoseGrid` please refer to `PseudoDualSimulations` class object
- `sigma2est` please refer to `PseudoDualSimulations` class object
- `...` additional parameters from `PseudoSimulations`

Value

the `PseudoDualSimulations` object

PseudoDualSimulations-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* `ModelEff` *class (except* `EffFlexi` *class). It contains all slots from* `GeneralSimulations` *and* `PseudoSimulations` *object. In comparison to the parent class* `PseudoSimulations`*, it contains additional slots to capture the dose-efficacy curve and the sigma2 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 `ModelEff` class (except `EffFlexi` class). It contains all slots from `GeneralSimulations` and `PseudoSimulations` object. In comparison to the parent class `PseudoSimulations`, it contains additional slots to capture the dose-efficacy curve and the sigma2 estimates.

Slots

`fitEff` list of the final values. If DLE and efficacy samples are gerneated, it contains the final fitted values. If no DLE and efficacy samples are used, it contains the modal estimates of the parameters in the two models and the posterior estimates of the probabilities of the occurrence of a DLE and the expected efficacy responses.

`FinalGstarEstimates` a vector of the final estimates of Gstar at the end of each simulations.

`FinalGstarAtDoseGrid` is a vectorof the final estimates of Gstar at dose Grid at the end of each simulations

`FinalGstarCIs` is the list of all 95% credibility interval of the final estimates of Gstar

`FinalGstarRatios` is the vector of the ratios of the CI, the ratio of the upper to the lower 95% credibility interval of the final estimates of Gstar

`FinalOptimalDose` is the vector of the final optimal dose, the minimum of the final TDtargetEndOfTrial estimates and Gstar estimates

`FinalOptimalDoseAtDoseGrid` is the vector of the final optimal dose, the minimum of the final TDtargetEndOfTrial estimates and Gstar estimates at dose Grid

`sigma2est` the vector of the final posterior mean sigma2 estimates

`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.

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 efficacy value at each dose level.

PseudoSimulations *Initialization function of the 'PseudoSimulations' class*

Description

Initialization function of the 'PseudoSimulations' class

Usage

```
PseudoSimulations(
  fit,
  FinalTDtargetDuringTrialEstimates,
  FinalTDtargetEndOfTrialEstimates,
  FinalTDtargetDuringTrialAtDoseGrid,
  FinalTDtargetEndOfTrialAtDoseGrid,
  FinalTDEOTCIs,
  FinalTDEOTRatios,
  FinalCIs,
  FinalRatios,
  stopReasons,
  ...
)
```

Arguments

fit	please refer to PseudoSimulations class object
FinalTDtargetDuringTrialEstimates	please refer to PseudoSimulations class object
FinalTDtargetEndOfTrialEstimates	please refer to PseudoSimulations class object
FinalTDtargetDuringTrialAtDoseGrid	please refer to PseudoSimulations class object
FinalTDtargetEndOfTrialAtDoseGrid	please refer to PseudoSimulations class object
FinalTDEOTCIs	please refer to PseudoSimulations class object
FinalTDEOTRatios	please refer to PseudoSimulations class object
FinalCIs	please refer to PseudoSimulations class object
FinalRatios	please refer to PseudoSimulations class object
stopReasons	please refer to PseudoSimulations class object
...	additional parameters from GeneralSimulations

Value

the PseudoSimulations object

PseudoSimulations-class

This is a class which captures the trial simulations from designs using pseudo model. The design for DLE only responses and model from `ModelTox` *class object. It contains all slots from* `GeneralSimulations` *object. Additional slots fit and stopReasons compared to the general class* `GeneralSimulations`.

Description

This is a class which captures the trial simulations from designs using pseudo model. The design for DLE only responses and model from `ModelTox` class object. It contains all slots from `GeneralSimulations` object. Additional slots fit and stopReasons compared to the general class `GeneralSimulations`.

Slots

`fit` list of the final values. If samples are involved, these are the final fitted values. If no samples are involved, these are included the final modal estimates of the model parameters and the posterior estimates of the probabilities of the occurrence of a DLE.

`FinalTDtargetDuringTrialEstimates` the vector of all final estimates (the last estimate of) the TDtargetDuringTrial at the end of each simultaions/when each trial stops

`FinalTDtargetEndOfTrialEstimates` vector of all final estimates or the last estimate of the TDtargetEndOfTrial when each trial stops

`FinalTDtargetDuringTrialAtDoseGrid` vector of the dose levels at dose grid closest below the final TDtargetDuringTrial estimates

`FinalTDtargetEndOfTrialAtDoseGrid` vector of the dose levels at dose grid closest below the final TDtargetEndOfTrial estimates

`FinalTDEOTCIs` is the list of all 95% credibility interval of the final estimates of the TDtargetEndOfTrial

`FinalTDEOTRatios` is the vector of the ratios of the CI, the raatio of the upper to the lower 95% credibility intervals of the final estimates of the TDtargetEndOfTrial

`FinalCIs` list of all the final 95% credibility intervals of the TDtargetEndofTrial estimates or of the final optimal dose estimates when DLE and efficacy responses are incorporated after each simulations

`FinalRatios` vector of all the final ratios, the ratios of the upper to the lower 95% credibility interval of the final estimates of the TDtargetEndOfTrial or of the final optimal dose estiamtes (when DLE and efficacy responses are incorporated) after each simulations

`stopReasons` todo: add slot description

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 quatile), 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 quatile), 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 quatile), 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 targetDoseDuringTrial

`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 quantiles of the mean fitted toxicity at each dose level

---

`Quantiles2LogisticNormal` *Convert prior quantiles (lower, median, upper) to logistic (log) normal model*

Description

This function uses generalised simulated annealing to optimise 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 *Initialization function for "RuleDesign"*

Description

Initialization function for "RuleDesign"

Usage

```
RuleDesign(nextBest, cohortSize, data, startingDose)
```

Arguments

nextBest	see RuleDesign
cohortSize	see RuleDesign
data	see RuleDesign
startingDose	see RuleDesign

Value

the RuleDesign object

RuleDesign-class *Class for rule-based designs*

Description

The difference to Design class is that model, stopping and increments slots are missing.

Slots

nextBest how to find the next best dose, an object of class NextBest

cohortSize rules for the cohort sizes, an object of class CohortSize

data what is the dose grid, any previous data, etc., contained in an object of class Data

startingDose what is the starting dose? Must lie on the grid in data

Examples

```
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(),
                       cohortSize = CohortSizeConst(size=3L),
                       data = emptydata,
                       startingDose = 5)
```

`Samples` *Initialization function for "Samples"*

Description

Initialization function for "Samples"

Usage

```
Samples(data, options)
```

Arguments

`data`	see `Samples`
`options`	see `Samples`

Value

the `Samples` object

`Samples-class` *Class for the MCMC output*

Description

Class for the MCMC output

Slots

`data` a list where each entry contains the samples of a (vector-valued) parameter in a vector/matrix in the format (number of samples) x (dimension of the parameter).

`options` the `McmcOptions` which have been used

Examples

```
# Saving the mcmc options which were used to generate the samples
options <- McmcOptions(burnin=10000,
                       step=2,
                       samples=10000)

# Create an object of class 'Samples'
# Here the parameters 'alpha' and 'beta' are randomly generated. Of coure in
# a real example these would come as an output from mcmc procedures
samples <- Samples(data=list(alpha=rnorm(10000),beta=rnorm(10000)),
                   options=options)
```

---

sampleSize *Compute the number of samples for a given MCMC options triple*

Description

Compute the number of samples for a given MCMC options triple

Usage

```
sampleSize(mcmcOptions)
```

Arguments

mcmcOptions	the McmcOptions object

Value

the resulting sample size

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
options <- McmcOptions(burnin=10000,
                       step=2,
                       samples=10000)

sampleSize(options)
```

setSeed *Helper function to set and save the RNG seed*

Description

This is basically copied from simulate.lm

Usage

```
setSeed(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 RNGstate will be returned, in order to call this function with this input to reproduce the obtained simulation results

Author(s)

Daniel Sabanes Bove <sabanesd@roche.com>

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
model <- DualEndpointRW(mu = c(0, 1),
                        Sigma = 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),
                        smooth="RW1")

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

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

# Choose the rule for stopping
myStopping4 <- StoppingTargetBiomarker(target=c(0.9, 1),
                                       prob=0.5)
# small number of patients just for illustration here
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,
                     cohortSize = CohortSizeConst(3),
                     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
# Also for illustration purpose, we will use 5 burn-ins to generate 20 samples
# this should be increased of course
mySims <- simulate(design,
                   trueTox=trueTox,
                   trueBiomarker=trueBiomarker,
                   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(mySims,
             trueTox = trueTox,
             trueBiomarker = trueBiomarker))
```

---

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

```
##If DLE and efficacy responses are considered in the simulations
##Specified your simulations when no 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),Effdose=c(25,300),
                    nu=c(a=1,b=0.025),data=data,c=0)

##The escalation rule using the 'NextBestMaxGain' class
mynextbest<-NextBestMaxGain(DLEDuringTrialtarget=0.35,
                            DLEEndOfTrialtarget=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,
                              Effmodel=Effmodel,
                              stopping=myStopping,
                              increments=myIncrements,
                              cohortSize=mySize,
                              data=data,startingDose=25)
##Specify the true DLE and efficacy curves
myTruthDLE<- function(dose)
{ DLEmodel@prob(dose, phi1=-53.66584, phi2=10.50499)
}

myTruthEff<- function(dose)
{Effmodel@ExpEff(dose,theta1=-4.818429,theta2=3.653058)
}

## Then specified the simulations and generate the trial
##For illustration purpose only 2 simulation is produced (nsim=2).
mySim <-simulate(object=design,
                 args=NULL,
                 trueDLE=myTruthDLE,
                 trueEff=myTruthEff,
                 trueNu=1/0.025,
                 nsim=2,
                 seed=819,
                 parallel=FALSE)
##Then produce a summary of your simulations
MYSUM <- summary(mySim,
                 trueDLE=myTruthDLE,
                 trueEff=myTruthEff)
##Then show the summary in data frame for your simulations
```

```
show(MYSUM)




##If DLE and efficacy samples are involved
##The escalation rule using the 'NextBestMaxGainSamples' class
mynextbest<-NextBestMaxGainSamples(DLEDuringTrialtarget=0.35,
                                   DLEEndOfTrialtarget=0.3,
                                   TDderive=function(TDsamples){
                                     quantile(TDsamples,prob=0.3)},
                                   Gstarderive=function(Gstarsamples){
                                     quantile(Gstarsamples,prob=0.5)})
##The design of 'DualResponsesSamplesDesign' class
design <- DualResponsesSamplesDesign(nextBest=mynextbest,
                                     cohortSize=mySize,
                                     startingDose=25,
                                     model=DLEmodel,
                                     Effmodel=Effmodel,
                                     data=data,
                                     stopping=myStopping,
                                     increments=myIncrements)
##options for MCMC
##for illustration purpose, we will use 50 burn-ins to generate 200 samples
options<-McmcOptions(burnin=50,step=2,samples=200)
##The simulations for illustration purpose we only simulate 2 trials (nsim=2)
mySim<-simulate(design,
                args=NULL,
                trueDLE=myTruthDLE,
                trueEff=myTruthEff,
                trueNu=1/0.025,
                nsim=2,
                mcmcOptions=options,
                seed=819,
                parallel=FALSE)


##Then produce a summary of your simulations
MYSUM <- summary(mySim,
                 trueDLE=myTruthDLE,
                trueEff=myTruthEff)
##Then show the summary in data frame for your simulations
show(MYSUM)
```

---

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 your simulations when no DLE samples are used
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(targetDuringTrial=0.35,
                         targetEndOfTrial=0.3)

##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,
                   cohortSize=mySize,
                   data=data,startingDose=25)

##Specify the truth of the DLE responses
myTruth <- function(dose)
{ model@prob(dose, phi1=-53.66584, phi2=10.50499)
}
```

```
##The simulations
##For illustration purpose only 1 simulation is produced (nsim=1).
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)
##show the summary of the simulated results in a data frame
show(MYSUM)



##If DLE samples are involved
##The escalation rule
tdNextBest<-NextBestTDsamples(targetDuringTrial=0.35,
                              targetEndOfTrial=0.3,
                              derive=function(TDsamples){quantile(TDsamples,probs=0.3)})
##The design
design <- TDsamplesDesign(model=model,
                          nextBest=tdNextBest,
                          stopping=myStopping,
                          increments=myIncrements,
                          cohortSize=mySize,
                          data=data,startingDose=25)
##Options for MCMC
##For illustration purpose, we will use 50 burn-ins to generate 200 samples and
##only simulate for 2 trials (nsim=2)
options<-McmcOptions(burnin=50,step=2,samples=200)
##The simualtions
mySim <- simulate(design,
                  args=NULL,
                  truth=myTruth,
                  nsim=2,
                  seed=819,
                  mcmcOptions=options,
                  parallel=FALSE)
##Then produce a summary of your simulations
MYSUM <- summary(mySim,
         truth=myTruth)
##show the summary of the simulated results in a data frame
show(MYSUM)
```

---

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

```
# 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),
                           refDose=56)

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

# Choose the rule for the cohort-size
mySize1 <- CohortSizeRange(intervals=c(0, 30),
                           cohortSize=c(1, 3))
mySize2 <- CohortSizeDLT(DLTintervals=c(0, 1),
                         cohortSize=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,
                 cohortSize=mySize,
                 data=emptydata,
                 startingDose=3)

## define the true function
myTruth <- function(dose)
{
  model@prob(dose, 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))
```

---

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),
  ...
)
```

Arguments

object	the Design object we want to simulate data from
nsim	the number of simulations (default: 1)
seed	see setSeed
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.
...	not used

Value

an object of class Simulations

Examples

```
# 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),
                           refDose=56)
```

```
# Choose the rule for selecting the next dose
myNextBest <- NextBestNCRM(target=c(0.2, 0.35),
                           overdose=c(0.35, 1),
                           maxOverdoseProb=0.25)

# Choose the rule for the cohort-size
mySize1 <- CohortSizeRange(intervals=c(0, 30),
                           cohortSize=c(1, 3))
mySize2 <- CohortSizeDLT(DLTintervals=c(0, 1),
                         cohortSize=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,
                 cohortSize=mySize,
                 data=emptydata,
                 startingDose=3)

## define the true function
myTruth <- function(dose)
{
  model@prob(dose, 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]
```

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),
  ...
)
```

Arguments

object	the DualDesign object we want to simulate data from
nsim	the number of simulations (default: 1)
seed	see setSeed
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.
...	not used

Value

an object of class DualSimulations

Examples

```
# 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(mu = c(0, 1),
                        Sigma = matrix(c(1, 0, 0, 1), nrow=2),
                        sigma2betaW = 0.01,
                        sigma2W = c(a=0.1, b=0.1),
                        useLogDose=TRUE,
                        refDose=2,
                        rho = c(a=1, b=1),
                        smooth="RW1")

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

# Choose the rule for the cohort-size
mySize1 <- CohortSizeRange(intervals=c(0, 30),
                           cohortSize=c(1, 3))
mySize2 <- CohortSizeDLT(DLTintervals=c(0, 1),
                         cohortSize=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,
                      cohortSize = 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))
```

`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(), 5),
  ...
)
```

Arguments

`object`	the `DualResponsesDesign` object we want to simulate the data from
`nsim`	the number of simulations (default :1)
`seed`	see `setSeed`
`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

```
##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),Effdose=c(25,300),
                    nu=c(a=1,b=0.025),data=data,c=0)

##The escalation rule using the 'NextBestMaxGain' class
mynextbest<-NextBestMaxGain(DLEDuringTrialtarget=0.35,
                            DLEEndOfTrialtarget=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,
                              Effmodel=Effmodel,
                              stopping=myStopping,
                              increments=myIncrements,
                              cohortSize=mySize,
```

```
                                data=data,startingDose=25)
##Specify the true DLE and efficacy curves
myTruthDLE<- function(dose)
{ DLEmodel@prob(dose, phi1=-53.66584, phi2=10.50499)
}

myTruthEff<- function(dose)
{Effmodel@ExpEff(dose,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)
```

`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(), 5),
  ...
)
```

Arguments

object	the DualResponsesSamplesDesign object we want to simulate the data from
nsim	the number of simulations (default :1)
seed	see setSeed
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 EffFlexi) the precision, the inverse of the variance of the efficacy responses
trueSigma2	(only with EffFlexi) the true variance of the efficacy responses which must be a single positive scalar.
trueSigma2betaW	(only with EffFlexi) the true variance for the random walk model used for smoothing. This must be a single postive 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

```
##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),Effdose=c(25,300),
                    nu=c(a=1,b=0.025),data=data,c=0)


##The escalation rule using the 'NextBestMaxGainSamples' class
mynextbest<-NextBestMaxGainSamples(DLEDuringTrialtarget=0.35,
                                   DLEEndOfTrialtarget=0.3,
                                   TDderive=function(TDsamples){
                                     quantile(TDsamples,prob=0.3)},
                                   Gstarderive=function(Gstarsamples){
                                     quantile(Gstarsamples,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,
                                     cohortSize=mySize,
                                     startingDose=25,
                                     model=DLEmodel,
                                     Effmodel=Effmodel,
                                     data=data,
                                     stopping=myStopping,
                                     increments=myIncrements)
```

```
##specified the true DLE and efficacy curve
myTruthDLE<- function(dose)
{ DLEmodel@prob(dose, phi1=-53.66584, phi2=10.50499)
}


myTruthEff<- function(dose)
{Effmodel@ExpEff(dose,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),Effdose=c(25,300),
                  sigma2=c(a=0.1,b=0.1),sigma2betaW=c(a=20,b=50),smooth="RW2",data=data)


##Specified the design
design <- DualResponsesSamplesDesign(nextBest=mynextbest,
                                     cohortSize=mySize,
                                     startingDose=25,
                                     model=DLEmodel,
                                     Effmodel=Effmodel,
                                     data=data,
                                     stopping=myStopping,
                                     increments=myIncrements)
##specified the true DLE curve and the true expected efficacy values at all dose levels
myTruthDLE<- function(dose)
{ DLEmodel@prob(dose, 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)
```

---

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(), 5),
  ...
)
```

Arguments

object	the RuleDesign object we want to simulate data from
nsim	the number of simulations (default: 1)
seed	see setSeed
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

```
# 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(),
                       cohortSize = 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),
                           refDose = 50)

## define the true function
myTruth <- function(dose)
{
  model@prob(dose, 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)
```

`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(), 5),
  ...
)
```

Arguments

object	the TDDesign object we want to simulate the data from
nsim	the number of simulations (default :1)
seed	see setSeed
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

```
##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(targetDuringTrial=0.35,
                         targetEndOfTrial=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,
                   cohortSize=mySize,
                   data=data,startingDose=25)

##Specify the truth of the DLE responses
myTruth <- function(dose)
{ model@prob(dose, 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)
```

`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(), 5),
  ...
)
```

Arguments

`object`	the `TDsamplesDesign` object we want to simulate the data from
`nsim`	the number of simulations (default :1)
`seed`	see `setSeed`
`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

```
##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(targetDuringTrial=0.35,
                              targetEndOfTrial=0.3,
                              derive=function(TDsamples){quantile(TDsamples,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,
```

```
                            cohortSize=mySize,
                            data=data,startingDose=25)

##Specify the truth of the DLE responses
myTruth <- function(dose)
{ model@prob(dose, 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)
```

`Simulations` *Initialization function for the "Simulations" class*

Description

Initialization function for the "Simulations" class

Usage

```
Simulations(fit, stopReasons, ...)
```

Arguments

`fit`	see `Simulations`
`stopReasons`	see `Simulations`
`...`	additional parameters from `GeneralSimulations`

Value

the `Simulations` object

Simulations-class *Class for the simulations output from model based designs*

Description

This class captures the trial simulations from model based designs. Additional slots fit and stopReasons compared to the general class `GeneralSimulations`.

Slots

`fit` list with the final fits

`stopReasons` list of stopping reasons for each simulation run

SimulationsSummary-class *Class for the summary of model-based simulations output*

Description

In addition to the slots in the parent class `GeneralSimulationsSummary`, it contains two slots with model fit information.

Details

Note that objects should not be created by users, therefore no initialization function is provided for this class.

Slots

`fitAtDoseMostSelected` fitted toxicity rate at dose most often selected

`meanFit` list with the average, lower (2.5 quantiles of the mean fitted toxicity at each dose level

size *Determine the size of the next cohort*

Description

This function determines the size of the next cohort.

Usage

```
size(cohortSize, dose, data, ...)

## S4 method for signature 'CohortSizeRange,ANY,Data'
size(cohortSize, dose, data, ...)

## S4 method for signature 'CohortSizeDLT,ANY,Data'
size(cohortSize, dose, data, ...)

## S4 method for signature 'CohortSizeMax,ANY,Data'
size(cohortSize, dose, data, ...)

## S4 method for signature 'CohortSizeMin,ANY,Data'
size(cohortSize, dose, data, ...)

## S4 method for signature 'CohortSizeConst,ANY,Data'
size(cohortSize, dose, data, ...)

## S4 method for signature 'CohortSizeParts,ANY,DataParts'
size(cohortSize, dose, data, ...)
```

Arguments

`cohortSize`	The rule, an object of class `CohortSize`
`dose`	the next dose
`data`	The data input, an object of class `Data`
`...`	additional arguments

Value

the size as integer value

Functions

- `size(cohortSize = CohortSizeRange, dose = ANY, data = Data)`: Determine the cohort size based on the range into which the next dose falls into
- `size(cohortSize = CohortSizeDLT, dose = ANY, data = Data)`: Determine the cohort size based on the number of DLTs so far

- size(cohortSize = CohortSizeMax, dose = ANY, data = Data): Size based on maximum of multiple cohort size rules
- size(cohortSize = CohortSizeMin, dose = ANY, data = Data): Size based on minimum of multiple cohort size rules
- size(cohortSize = CohortSizeConst, dose = ANY, data = Data): Constant cohort size
- size(cohortSize = CohortSizeParts, dose = ANY, data = DataParts): Cohort size based on the parts

Examples

```
# 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),
                           refDose=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),
                           maxOverdoseProb=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),
                          cohortSize=c(1, 3))

# Determine the cohort size for the next cohort
size(mySize, dose=doseRecommendation$value, data = data)


# 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),
                           refDose=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),
                           maxOverdoseProb=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(DLTintervals = c(0, 1),
                        cohortSize = c(1, 3))
```

```
# Determine the cohort size for the next cohort
size(mySize, dose=doseRecommendation$value, data = data)


# 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),
                           refDose=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),
                           maxOverdoseProb=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),
                           cohortSize = 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(DLTintervals=c(0, 1),
                         cohortSize=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)




# 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),
                           refDose=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),
                           maxOverdoseProb=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),
                           cohortSize = 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(DLTintervals=c(0, 1),
                         cohortSize=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)




# 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),
                           refDose=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),
                           maxOverdoseProb=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, data = data)




# 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),
                           refDose=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(dltStart=0,
                                        cleanStart=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),
                           maxOverdoseProb=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(sizes=c(1,3))

# Determine the cohort size for the next cohort
size(mySize, dose=doseRecommendation$value, data = data)
```

Stopping-class *The virtual class for stopping rules*

Description

The virtual class for stopping rules

See Also

StoppingList, StoppingCohortsNearDose, StoppingPatientsNearDose, StoppingMinCohorts, StoppingMinPatients, StoppingTargetProb StoppingMTDdistribution, StoppingTargetBiomarker, StoppingHighestDose

---

StoppingAll *Initialization function for "StoppingAll"*

Description

Initialization function for "StoppingAll"

Usage

```
StoppingAll(stopList)
```

Arguments

stopList	see StoppingAll

Value

the StoppingAll object

StoppingAll-class *Stop based on fullfillment of all multiple stopping rules*

Description

This class can be used to combine multiple stopping rules with an AND operator.

Details

`stopList` contains all stopping rules, which are again objects of class `Stopping`. All stopping rules must be fulfilled in order that the result of this rule is to stop.

Slots

`stopList` list of stopping rules

Examples

```
# Define some 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 'StoppingAll') which would then be
# summarized by the 'all' function, meaning that the study would be stopped only if
# 'all' the single stopping rules are TRUE
mystopping <- StoppingAll(stopList=c(myStopping1,myStopping2,myStopping3))
```

StoppingAny *Initialization function for "StoppingAny"*

Description

Initialization function for "StoppingAny"

Usage

```
StoppingAny(stopList)
```

Arguments

`stopList`	see `StoppingAny`

Value

the `StoppingAny` object

StoppingAny-class *Stop based on fullfillment of any stopping rule*

Description

This class can be used to combine multiple stopping rules with an OR operator.

Details

`stopList` contains all stopping rules, which are again objects of class `Stopping`. Any of these rules must be fulfilled in order that the result of this rule is to stop.

Slots

`stopList` list of stopping rules

Examples

```
# Define some 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 'StoppingAny') which will then be
# summarized with the 'any' function, meaning that the study would be stopped if
# 'any' of the single stopping rules is TRUE.
mystopping <- StoppingAny(stopList=c(myStopping1,myStopping2,myStopping3))
```

StoppingCohortsNearDose *Initialization function for "StoppingCohortsNearDose"*

Description

Initialization function for "StoppingCohortsNearDose"

Usage

```
StoppingCohortsNearDose(nCohorts, percentage)
```

Arguments

`nCohorts`	see `StoppingCohortsNearDose`
`percentage`	see `StoppingCohortsNearDose`

Value

the StoppingCohortsNearDose object

StoppingCohortsNearDose-class *Stop based on number of cohorts near to next best dose*

Description

Stop based on number of cohorts near to next best dose

Slots

`nCohorts` number of required cohorts

`percentage` percentage (between 0 and 100) within the next best dose the cohorts must lie

Examples

```
# As example, 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

myStopping <- StoppingCohortsNearDose(nCohorts = 3,
                                      percentage = 0.2)
```

StoppingGstarCIRatio *Initialization function for "StoppingGstarCIRatio"*

Description

Initialization function for "StoppingGstarCIRatio"

Usage

```
StoppingGstarCIRatio(targetRatio, targetEndOfTrial)
```

Arguments

`targetRatio` please refer to StoppingGstarCIRatio class object

`targetEndOfTrial` please refer to StoppingGstarCIRatio class object

Value

the StoppingGstarCIRatio class object

StoppingGstarCIRatio-class

Stop based on a target ratio, the ratio of the upper to the lower 95% credibility interval of the estimate of the minimum of the dose which gives the maximum gain (Gstar) and the TD end of trial, the dose with probability of DLE equals to the target probability of DLE used at the end of a trial.

Description

Stop based on a target ratio, the ratio of the upper to the lower 95% credibility interval of the estimate of the minimum of the dose which gives the maximum gain (Gstar) and the TD end of trial, the dose with probability of DLE equals to the target probability of DLE used at the end of a trial.

Slots

`targetRatio` the target ratio of the upper to the lower of the 95% credibility interval of the estimate that required to stop a trial

`targetEndOfTrial` the target probability of DLE to be used at the end of a trial

Examples

```
##Define the target stopping ratio of 5 and
##the target probability of DLE to be used at the end of a trial
##This is a ratio of the upper to the lower 95% credibility interval of the estimates
myStopping <- StoppingGstarCIRatio(targetRatio=5,
                                   targetEndOfTrial=0.3)
```

StoppingHighestDose *Initialization function for "StoppingHighestDose"*

Description

Initialization function for "StoppingHighestDose"

Usage

```
StoppingHighestDose()
```

Value

the `StoppingHighestDose` object

StoppingHighestDose-class

Stop when the highest dose is reached

Description

Stop when the highest dose is reached

Examples

```
## for example this can be used in the following way:
## we would like to stop if:
## - next proposed dose is highest dose
## - there are already at least 3 patients on that dose
## - we are sure that this dose is safe, e.g. the
## probability to be in the interval (0%, 20%) of DLT
## rate is above 50%.
## This would be implemented as the following combination:

stopHigh <-
  StoppingHighestDose() &
  StoppingPatientsNearDose(nPatients=3, percentage=0) &
  StoppingTargetProb(target=c(0, 0.2),
                     prob=0.5)

## of course this rule would then need to be combined
## with the other standard rules for when to stop
## 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:
## stopRule <- stopHigh | stopLow | stopSamplesize
```

StoppingList

Initialization function for "StoppingList"

Description

Initialization function for "StoppingList"

Usage

```
StoppingList(stopList, summary)
```

Arguments

stopList	see StoppingList
summary	see StoppingList

Value

the StoppingList object

StoppingList-class *Stop based on multiple stopping rules*

Description

This class can be used to combine multiple stopping rules.

Details

stopList contains all stopping rules, which are again objects of class Stopping, and the summary is a function taking a logical vector of the size of stopList and returning a single logical value. For example, if the function all is given as summary function, then this means that all stopping rules must be fulfilled in order that the result of this rule is to stop.

Slots

stopList list of stopping rules

summary the summary function to combine the results of the stopping rules into a single result

Examples

```
# Define some 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(stopList=c(myStopping1,myStopping2,myStopping3),
                           summary=any)
```

StoppingMinCohorts *Initialization function for "StoppingMinCohorts"*

Description

Initialization function for "StoppingMinCohorts"

Usage

```
StoppingMinCohorts(nCohorts)
```

Arguments

nCohorts	see StoppingMinCohorts

Value

the StoppingMinCohorts object

StoppingMinCohorts-class *Stop based on minimum number of cohorts*

Description

Stop based on minimum number of cohorts

Slots

nCohorts minimum required number of cohorts

Examples

```
# As example, here is the rule for:
#   stopping the study if at least 6 cohorts were already dosed

myStopping <- StoppingMinCohorts(nCohorts = 6)
```

StoppingMinPatients *Initialization function for "StoppingMinPatients"*

Description

Initialization function for "StoppingMinPatients"

Usage

```
StoppingMinPatients(nPatients)
```

Arguments

nPatients	see StoppingMinPatients

Value

the StoppingMinPatients object

StoppingMinPatients-class *Stop based on minimum number of patients*

Description

Stop based on minimum number of patients

Slots

nPatients minimum allowed number of patients

Examples

```
# As example, here is the rule for:
#   stopping the study if at least 20 patients were already dosed

myStopping <- StoppingMinPatients(nPatients = 20)
```

StoppingMTDdistribution

Initialization function for "StoppingMTDdistribution"

Description

Initialization function for "StoppingMTDdistribution"

Usage

```
StoppingMTDdistribution(target, thresh, prob)
```

Arguments

target	see StoppingMTDdistribution
thresh	see StoppingMTDdistribution
prob	see StoppingMTDdistribution

Value

the StoppingMTDdistribution object

StoppingMTDdistribution-class

Stop based on MTD distribution

Description

Has 90% probability above a threshold of 50% of the current MTD been reached? This class is used for this question.

Slots

target the target toxicity probability (e.g. 0.33) defining the MTD

thresh the threshold relative to the MTD (e.g. 0.5)

prob required probability (e.g. 0.9)

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(DLE)=0.33

myStopping <- StoppingMTDdistribution(target = 0.33,
                                      thresh = 0.5,
                                      prob = 0.9)
```

StoppingPatientsNearDose

Initialization function for "StoppingPatientsNearDose"

Description

Initialization function for "StoppingPatientsNearDose"

Usage

```
StoppingPatientsNearDose(nPatients, percentage)
```

Arguments

nPatients	see StoppingPatientsNearDose
percentage	see StoppingPatientsNearDose

Value

the StoppingPatientsNearDose object

StoppingPatientsNearDose-class

Stop based on number of patients near to next best dose

Description

Stop based on number of patients near to next best dose

Slots

nPatients number of required patients

percentage percentage (between 0 and 100) within the next best dose the patients must lie

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

myStopping <- StoppingPatientsNearDose(nPatients = 9,
                                       percentage = 0.2)
```

StoppingTargetBiomarker

Initialization function for "StoppingTargetBiomarker"

Description

Initialization function for "StoppingTargetBiomarker"

Usage

```
StoppingTargetBiomarker(target, scale = c("relative", "absolute"), prob)
```

Arguments

target	see StoppingTargetBiomarker
scale	see StoppingTargetBiomarker
prob	see StoppingTargetBiomarker

Value

the StoppingTargetBiomarker object

StoppingTargetBiomarker-class

Stop based on probability of target biomarker

Description

Stop based on probability of target biomarker

Slots

target the biomarker target range, that needs to be reached. For example, (0.8, 1.0) and scale="relative" means we target a dose with at least 80% of maximum biomarker level.

scale either relative (default, then the target is interpreted relative to the maximum, so must be a probability range) or absolute (then the target is interpreted as absolute biomarker range)

prob required target probability for reaching sufficient precision

Examples

```
# As example, here is the rule for:
#   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).

myStopping <- StoppingTargetBiomarker(target = c(0.9, 1),
                                      prob = 0.5)
```

StoppingTargetProb *Initialization function for "StoppingTargetProb"*

Description

Initialization function for "StoppingTargetProb"

Usage

```
StoppingTargetProb(target, prob)
```

Arguments

target	see StoppingTargetProb
prob	see StoppingTargetProb

Value

the StoppingTargetProb object

StoppingTargetProb-class *Stop based on probability of target tox interval*

Description

Stop based on probability of target tox interval

Slots

target the target toxicity interval, e.g. c(0.2, 0.35)

prob required target toxicity probability (e.g. 0.4) for reaching sufficient precision

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

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

`StoppingTDCIRatio` *Initialization function for "StoppingTDCIRatio"*

Description

Initialization function for "StoppingTDCIRatio"

Usage

```
StoppingTDCIRatio(targetRatio, targetEndOfTrial)
```

Arguments

`targetRatio` please refer to `StoppingTDCIRatio` class object

`targetEndOfTrial` please refer to `StoppingTDCIRatio` class object

Value

the `StoppingTDCIRatio` class object

`StoppingTDCIRatio-class` *Stop based on a target ratio, the ratio of the upper to the lower 95% credibility interval of the estimate of TD end of trial, the dose with probability of DLE equals to the target probability of DLE used at the end of a trial*

Description

Stop based on a target ratio, the ratio of the upper to the lower 95% credibility interval of the estimate of TD end of trial, the dose with probability of DLE equals to the target probability of DLE used at the end of a trial

Slots

`targetRatio` the target ratio of the upper to the lower of the 95% credibility interval of the estimate that required to stop a trial

`targetEndOfTrial` the target probability of DLE to be used at the end of a trial

Examples

```
##Define the target stopping ratio of 5 and
##the target probability of DLE to be used at the end of a trial
##This is a ratio of the upper to the lower 95% credibility interval of the estimates
myStopping <- StoppingTDCIRatio(targetRatio=5,
                                targetEndOfTrial=0.3)
```

---

stopTrial *Stop the trial?*

Description

This function returns whether to stop the trial.

Usage

```
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,Model,ANY'
stopTrial(stopping, dose, samples, model, data, ...)

## S4 method for signature 'StoppingMTDdistribution,numeric,Samples,Model,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 'StoppingHighestDose,numeric,ANY,ANY,Data'
stopTrial(stopping, dose, samples, model, data, ...)

## 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 'StoppingGstarCIRatio,ANY,Samples,ModelTox,DataDual'
stopTrial(
  stopping,
  dose,
  samples,
  model,
  data,
  TDderive,
  Effmodel,
  Effsamples,
  Gstarderive,
  ...
)

## S4 method for signature 'StoppingGstarCIRatio,ANY,missing,ModelTox,DataDual'
stopTrial(stopping, dose, model, data, Effmodel, ...)
```

Arguments

stopping	The rule, an object of class Stopping
dose	the recommended next best dose
samples	the Samples object
model	The model input, an object of class Model
data	The data input, an object of class Data
...	additional arguments
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 = 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 = Model, data = ANY): Stop based on probability of target tox interval
- stopTrial(stopping = StoppingMTDdistribution, dose = numeric, samples = Samples, model = Model, 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 = StoppingHighestDose, dose = numeric, samples = ANY, model = ANY, data = Data): Stop when the highest dose is reached
- 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 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 interval of the estimate (TDtargetEndOfTrial). This is a stopping rule which incorporate only DLE responses and no DLE samples are involved
- stopTrial(stopping = StoppingGstarCIRatio, dose = ANY, samples = Samples, model = ModelTox, data = DataDual): Stop based on reaching the target ratio of the upper to the lower 95 interval of the estimate (the minimum of Gstar and TDtargetEndOfTrial). This is a stopping rule which incorporate DLE and efficacy responses and DLE and efficacy samples are also used.
- stopTrial(stopping = StoppingGstarCIRatio, dose = ANY, samples = missing, model = ModelTox, data = DataDual): Stop based on reaching the target ratio of the upper to the lower 95 interval of the estimate (the minimum of Gstar and TDtargetEndOfTrial). 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





# 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),
                           refDose=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),
                           maxOverdoseProb=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(stopList=c(myStopping1,myStopping2,myStopping3),
                           summary=any)

# Evaluate if to stop the Trial
stopTrial(stopping=myStopping, dose=doseRecommendation$value,
          samples=samples, model=model, data=data)




# 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),
                           refDose=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),
                           maxOverdoseProb=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)




# 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),
                           refDose=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),
                           maxOverdoseProb=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)




# 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),
                           refDose=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),
                           maxOverdoseProb=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)


# 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),
                           refDose=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),
                           maxOverdoseProb=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)


# 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),
                           refDose=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),
                           maxOverdoseProb=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)


# 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),
                           refDose=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),
                           maxOverdoseProb=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)


# 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),
                           refDose=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),
                           maxOverdoseProb=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)


# 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),
                           refDose=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),
                           maxOverdoseProb=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)


# 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(mu = c(0, 1),
                        Sigma = 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),
                        smooth = "RW1")

# 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),
                                   maxOverdoseProb=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)

# 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, 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),
                           refDose=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),
                           maxOverdoseProb=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)



##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(targetRatio=5,
                                        targetEndOfTrial=0.3)
##Find the next Recommend dose using the nextBest method (plesae refer to nextbest examples)
tdNextBest<-NextBestTDsamples(targetDuringTrial=0.35,targetEndOfTrial=0.3,
                              derive=function(TDsamples){quantile(TDsamples,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$nextdose,
          samples=samples,model=model,data=data)
## RecommendDose$nextdose refers to the next dose obtained in RecommendDose
##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(targetRatio=5,
                                targetEndOfTrial=0.3)
##Find the next Recommend dose using the nextBest method (plesae refer to nextbest examples)
tdNextBest<-NextBestTD(targetDuringTrial=0.35,targetEndOfTrial=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$nextdose,
          model=model,data=data)
## RecommendDose$nextdose refers to the next dose obtained in RecommendDose
##define the stopping rules based on the 'StoppingGstarCIRatio' 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),Effdose=c(25,300),nu=c(a=1,b=0.025),data=data,c=0)
##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 'StoppingGstarCIRatio' class
myStopping <- StoppingGstarCIRatio(targetRatio=5,
                                          targetEndOfTrial=0.3)
##Find the next Recommend dose using the nextBest method (plesae refer to nextbest examples)
mynextbest<-NextBestMaxGainSamples(DLEDuringTrialtarget=0.35,
                                   DLEEndOfTrialtarget=0.3,
                                   TDderive=function(TDsamples){
                                     quantile(TDsamples,prob=0.3)},
                                   Gstarderive=function(Gstarsamples){
                                     quantile(Gstarsamples,prob=0.5)})

RecommendDose<-nextBest(mynextbest,doselimit=max(data@doseGrid),samples=DLEsamples,model=DLEmodel,
                        data=data,Effmodel=Effmodel,Effsamples=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$nextdose,
          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)})

## RecommendDose$nextdose refers to the next dose obtained in RecommendDose
##define the stopping rules based on the 'StoppingGstarCIRatio' 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),Effdose=c(25,300),nu=c(a=1,b=0.025),data=data,c=0)


##define the 'StoppingGstarCIRatio' class
myStopping <- StoppingGstarCIRatio(targetRatio=5,
                                   targetEndOfTrial=0.3)
##Find the next Recommend dose using the nextBest method (plesae refer to nextbest examples)
mynextbest<-NextBestMaxGain(DLEDuringTrialtarget=0.35,DLEEndOfTrialtarget=0.3)
```

```
RecommendDose<-nextBest(mynextbest,doselimit=max(data@doseGrid),model=DLEmodel,
                        Effmodel=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$nextdose,model=DLEmodel,
          data=data, Effmodel=Effmodel)


## RecommendDose$nextdose refers to the next dose obtained in RecommendDose
```

---

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
model <- DualEndpointRW(mu = c(0, 1),
                        Sigma = 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),
                        smooth="RW1")

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

# Choose the rule for stopping
myStopping4 <- StoppingTargetBiomarker(target=c(0.9, 1),
                                       prob=0.5)
# StoppingMinPatients will usually take a higher sample size,
# just for illustration here
myStopping <- myStopping4 | StoppingMinPatients(6)

# 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,
                     cohortSize = CohortSizeConst(3),
                     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
# Also for illustration purpose, we will use 5 burn-ins to generate 20 samples
# this should be increased of course!
mySims <- simulate(design,
                   trueTox=trueTox,
                   trueBiomarker=trueBiomarker,
                   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(mySims,
        trueTox = trueTox,
        trueBiomarker = trueBiomarker)
```

---

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

```
##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),Effdose=c(25,300),
                  sigma2=c(a=0.1,b=0.1),sigma2betaW=c(a=20,b=50),smooth="RW2",data=data)


##specified the next best
mynextbest<-NextBestMaxGainSamples(DLEDuringTrialtarget=0.35,
                                   DLEEndOfTrialtarget=0.3,
                                   TDderive=function(TDsamples){
                                     quantile(TDsamples,prob=0.3)},
                                   Gstarderive=function(Gstarsamples){
                                     quantile(Gstarsamples,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,
                                     cohortSize=mySize,
                                     startingDose=25,
                                     model=DLEmodel,
                                     Effmodel=Effmodel,
                                     data=data,
                                     stopping=myStopping,
                                     increments=myIncrements)
##specified the true DLE curve and the true expected efficacy values at all dose levels
myTruthDLE<- function(dose)
{ DLEmodel@prob(dose, 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)
```

---

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 your simulations when no samples are used
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)

##The efficacy model of 'ModelEff' (e.g 'Effloglog') class
Effmodel<-Effloglog(Eff=c(1.223,2.513),Effdose=c(25,300),
                    nu=c(a=1,b=0.025),data=data)

##The escalation rule using the 'NextBestMaxGain' class
mynextbest<-NextBestMaxGain(DLEDuringTrialtarget=0.35,
                            DLEEndOfTrialtarget=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,
                              Effmodel=Effmodel,
                              stopping=myStopping,
                              increments=myIncrements,
                              cohortSize=mySize,
                              data=data,startingDose=25)
##Specify the true DLE and efficacy curves
myTruthDLE<- function(dose)
{ DLEmodel@prob(dose, phi1=-53.66584, phi2=10.50499)
}

myTruthEff<- function(dose)
{Effmodel@ExpEff(dose,theta1=-4.818429,theta2=3.653058)
}

## Then specified the simulations and generate the trial for 2 times
mySim <-simulate(object=design,
                 args=NULL,
                 trueDLE=myTruthDLE,
                 trueEff=myTruthEff,
                 trueNu=1/0.025,
                 nsim=2,
                 seed=819,
                 parallel=FALSE)

##Then produce a summary of your simulations
summary(mySim,
        trueDLE=myTruthDLE,
        trueEff=myTruthEff)
##If DLE and efficacy samples are involved
##Please refer to design-method 'simulate DualResponsesSamplesDesign' examples for details
##specified the next best
mynextbest<-NextBestMaxGainSamples(DLEDuringTrialtarget=0.35,
                                   DLEEndOfTrialtarget=0.3,
                                   TDderive=function(TDsamples){
                                     quantile(TDsamples,prob=0.3)},
                                   Gstarderive=function(Gstarsamples){
                                     quantile(Gstarsamples,prob=0.5)})
##specified the design
design <- DualResponsesSamplesDesign(nextBest=mynextbest,
                                     cohortSize=mySize,
                                     startingDose=25,
                                     model=DLEmodel,
                                     Effmodel=Effmodel,
                                     data=data,
                                     stopping=myStopping,
                                     increments=myIncrements)
##options for MCMC
##For illustration purpose, we will use 50 burn-ins to generate 200 samples
options<-McmcOptions(burnin=50,step=2,samples=200)
##The simulations
##For illustration purpose only 2 simulation is produced (nsim=2).
mySim<-simulate(design,
```

```
                  args=NULL,
                  trueDLE=myTruthDLE,
                  trueEff=myTruthEff,
                  trueNu=1/0.025,
                  nsim=2,
                  mcmcOptions=options,
                  seed=819,
                  parallel=FALSE)

##Then produce a summary of your simulations
summary(mySim,
        trueDLE=myTruthDLE,
        trueEff=myTruthEff)
```

---

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

```
##If only DLE responses are considered in the simulations
##Specified your simulations when no DLE samples are used
## 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(targetDuringTrial=0.35,
                         targetEndOfTrial=0.3)

##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,
                   cohortSize=mySize,
                   data=data,startingDose=25)

##Specify the truth of the DLE responses
myTruth <- function(dose)
{ model@prob(dose, phi1=-53.66584, phi2=10.50499)
}



##(Please refer to desgin-method 'simulate TDDesign' examples for details)
##For illustration purpose only 1 simulation is produced (nsim=1).
mySim <- simulate(design,
                  args=NULL,
                  truth=myTruth,
                  nsim=1,
                  seed=819,
                  parallel=FALSE)
##Then produce a summary of your simulations
summary(mySim,
```

```
        truth=myTruth)
```

```
##If DLE samples are involved
##specify the next best
tdNextBest<-NextBestTDsamples(targetDuringTrial=0.35,
                              targetEndOfTrial=0.3,
                             derive=function(TDsamples){quantile(TDsamples,probs=0.3)})
##The design
design <- TDsamplesDesign(model=model,
                          nextBest=tdNextBest,
                          stopping=myStopping,
                          increments=myIncrements,
                          cohortSize=mySize,
                          data=data,startingDose=25)
##options for MCMC
##For illustration purpose, we will use 50 burn-ins to generate 200 samples
options<-McmcOptions(burnin=50,step=2,samples=200)
##The simulations
## For illustration purpose we will only generate 2 trials (nsim=2)
mySim <- simulate(design,
                  args=NULL,
                  truth=myTruth,
                  nsim=2,
                  seed=819,
                  mcmcOptions=options,
                  parallel=FALSE)
##Then produce a summary of your simulations
summary(mySim,
        truth=myTruth)
```

---

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

```
# 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),
                           refDose=56)

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

# Choose the rule for the cohort-size
mySize1 <- CohortSizeRange(intervals=c(0, 30),
                           cohortSize=c(1, 3))
mySize2 <- CohortSizeDLT(DLTintervals=c(0, 1),
                         cohortSize=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,
```

```
                  cohortSize=mySize,
                  data=emptydata,
                  startingDose=3)

## define the true function
myTruth <- function(dose)
{
  model@prob(dose, 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))[3]

# Summarize the Results of the Simulations
summary(mySims,truth=myTruth)
```

TDDesign *Initialization function for 'TDDesign' class*

Description

Initialization function for 'TDDesign' class

Usage

```
TDDesign(model, stopping, increments, PLcohortSize = CohortSizeConst(0L), ...)
```

Arguments

model	please refer to TDDesign class object
stopping	please refer to TDDesign class object
increments	please refer to TDDesign class object
PLcohortSize	see TDDesign
...	additional arguments for RuleDesign

Value

the TDDesign class object

TDDesign-class *Design class using DLE responses only based on the pseudo DLE model without sample*

Description

This is a class of design based only on DLE responses using the 'LogisticIndepBeta' class model are used without samples. In addition to the slots in the more simple RuleDesign, objects of this class contain:

Slots

`model` the pseudo DLE model to be used, an object class of ModelTox

`stopping` stopping rule(s) for the trial, an object class of Stopping

`increments` how to control increments between dose levels, an object class of Increments

`PLcohortSize` rules for the cohort sizes for placebo, if any planned an object of class CohortSize

Examples

```
##Specified the design to run simulations
##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 no 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(targetDuringTrial=0.35,
                         targetEndOfTrial=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)
##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,
                    cohortSize=mySize,
                    data=data,startingDose=25)
```

TDsamplesDesign *Initialization function for 'TDsamplesDesign' class*

Description

Initialization function for 'TDsamplesDesign' class

Usage

```
TDsamplesDesign(
  model,
  stopping,
  increments,
  PLcohortSize = CohortSizeConst(0L),
  ...
)
```

Arguments

model	see TDsamplesDesign
stopping	see TDsamplesDesign
increments	see TDsamplesDesign
PLcohortSize	see TDsamplesDesign
...	additional arguments for RuleDesign

Value

the TDsamplesDesign class object

TDsamplesDesign-class *This is a class of design based only on DLE responses using the 'LogisticIndepBeta' class model and DLE samples are also used. In addition to the slots in the more simple* RuleDesign*, objects of this class contain:*

Description

This is a class of design based only on DLE responses using the 'LogisticIndepBeta' class model and DLE samples are also used. In addition to the slots in the more simple RuleDesign, objects of this class contain:

Slots

`model` the pseudo DLE model to be used, an object class of ModelTox

`stopping` stopping rule(s) for the trial, an object class of Stopping

`increments` how to control increments between dose levels, an object class of Increments

`PLcohortSize` rules for the cohort sizes for placebo, if any planned an object of class CohortSize

Examples

```
##Specified the design to run simulations
##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)

samples <- mcmc(data=data, model=model, options=McmcOptions(burnin=100,step=2,samples=200))
##Then the escalation rule
tdNextBest<-NextBestTDsamples(targetDuringTrial=0.35,targetEndOfTrial=0.3,
                              derive=function(TDsamples){quantile(TDsamples,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)
##Now specified the design with all the above information and starting with a dose of 25
design <- TDsamplesDesign(model=model,
```

```
                              nextBest=tdNextBest,
                              stopping=myStopping,
                              increments=myIncrements,
                              cohortSize=mySize,
                              data=data,startingDose=25)
```

ThreePlusThreeDesign *Creates a new 3+3 design object from a dose grid*

Description

Creates a new 3+3 design object from a dose grid

Usage

```
ThreePlusThreeDesign(doseGrid)
```

Arguments

doseGrid	the dose grid to be used

Value

the object of class RuleDesign with the 3+3 design

Author(s)

Daniel Sabanes Bove <sabanesd@roche.com>

Examples

```
# inizialing a 3+3 design
myDesign <- ThreePlusThreeDesign(doseGrid=c(5, 10, 15, 25, 35, 50, 80))
```

update,Data-method *Update method for the "Data" class*

Description

Add new data to the Data object

Usage

```
## S4 method for signature 'Data'
update(
  object,
  x,
  y,
  ID = (if (length(object@ID)) max(object@ID) else 0L) + seq_along(y),
  newCohort = TRUE,
  ...
)
```

Arguments

object	the old Data object
x	the dose level (one level only!)
y	the DLT vector (0/1 vector), for all patients in this cohort
ID	the patient IDs
newCohort	logical: if TRUE (default) the new data are assigned to a new cohort
...	not used

Value

the new Data object

Examples

```
# Create some data of class 'Data'
myData <- 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
myData <- update(myData,
                 x=20,
                 y=c(0,1,1))
```

update,DataDual-method

Update method for the "DataDual" class

Description

Add new data to the DataDual object

Usage

```
## S4 method for signature 'DataDual'
update(
  object,
  x,
  y,
  w,
  newCohort = TRUE,
  ID = (if (length(object@ID)) max(object@ID) else 0L) + seq_along(y),
  ...
)
```

Arguments

object	the old DataDual object
x	the dose level (one level only!)
y	the DLT vector (0/1 vector), for all patients in this cohort
w	the biomarker vector, for all patients in this cohort
newCohort	logical: if TRUE (default) the new data are assigned to a new cohort
ID	the patient IDs
...	not used

Value

the new DataDual object

Examples

```
# Create some data of class 'DataDual'
myData <- 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
myData <- update(myData,
```

```
                    x=20,
                    y=c(0,1,1),
                    w=c(0.4,1.2,2.2))
```

---

update,DataParts-method

Update method for the "DataParts" class

Description

Add new data to the DataParts object

Usage

```
## S4 method for signature 'DataParts'
update(
  object,
  x,
  y,
  ID = (if (length(object@ID)) max(object@ID) else 0L) + seq_along(y),
  ...
)
```

Arguments

object	the old DataParts object
x	the dose level (one level only!)
y	the DLT vector (0/1 vector), for all patients in this cohort
ID	the patient IDs
...	not used

Value

the new DataParts object

Examples

```
# create an object of class 'DataParts'
myData <- 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
# to be noted that since we reached the last level from part1Ladder then
# nextPart is switched from '1' to '2'
myData <- update(myData,
                 x=10,
                 y=c(0))
```

---

update,EffFlexi-method

Update method for the 'EffFlexi' Model class. This is a method to update estimates both for the flexible form model and the random walk model (see details in `EffFlexi` *class object) when new data or new observations of responses are available and added in.*

Description

Update method for the 'EffFlexi' Model class. This is a method to update estimates both for the flexible form model and the random walk model (see details in `EffFlexi` class object) when new data or new observations of responses are available and added in.

Usage

```
## S4 method for signature 'EffFlexi'
update(object, data, ...)
```

Arguments

`object`	is the model which follow `EffFlexi` class object
`data`	all currently available data and responses of `DataDual` class object
`...`	unused

Value

the new `EffFlexi` class object

Examples

```
##Update the 'EffFlexi' model with new data
## first define the data and the model
emptydata<-DataDual(doseGrid=seq(25,300,25))
data<-emptydata

Effmodel<- EffFlexi(Eff=c(1.223, 2.513),Effdose=c(25,300),
                    sigma2=c(a=0.1,b=0.1),sigma2betaW=c(a=20,b=50),smooth="RW2",data=data)
```

```
##Then we have some new observations 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))

##update the model to get new estimates
newEffModel <- update(object=Effmodel,data=data)
```

`update,Effloglog-method`

Update method for the 'Effloglog' Model class. This is a method to update the modal estimates of the model parameters θ_1 (theta1), θ_2 (theta2) and ν (nu, the precision of the efficacy responses) when new data or new observations of responses are available and added in.

Description

Update method for the 'Effloglog' Model class. This is a method to update the modal estimates of the model parameters θ_1 (theta1), θ_2 (theta2) and ν (nu, the precision of the efficacy responses) when new data or new observations of responses are available and added in.

Usage

```
## S4 method for signature 'Effloglog'
update(object, data, ...)
```

Arguments

`object`	the `Effloglog` class object
`data`	all currently available data or responses of `DataDual` class object
`...`	unused

Value

the new `Effloglog` class object

Examples

```
##Update the 'Effloglog' model with new data
## first define the data and the model
emptydata<-DataDual(doseGrid=seq(25,300,25),placebo=FALSE)
data<-emptydata

Effmodel<-Effloglog(Eff=c(1.223,2.513),Effdose=c(25,300),nu=c(a=1,b=0.025),data=data,c=0)

##Then we have some new observations 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))

##update the model to get new estimates
newEffModel <- update(object=Effmodel,data=data)
```

`update,LogisticIndepBeta-method`

Update method for the 'LogisticIndepBeta'Model class. This is a method to update the modal estimates of the model parameters ϕ_1 (phi1) and ϕ_2 (phi2) when new data or new observations of responses are available and added in.

Description

Update method for the 'LogisticIndepBeta'Model class. This is a method to update the modal estimates of the model parameters ϕ_1 (phi1) and ϕ_2 (phi2) when new data or new observations of responses are available and added in.

Usage

```
## S4 method for signature 'LogisticIndepBeta'
update(object, data, ...)
```

Arguments

`object`	the model of `LogisticIndepBeta` class object
`data`	all currently availabvle of `Data` class object
`...`	unused

Value

the new `LogisticIndepBeta` class object

Examples

```
##Update the 'LogisticIndepBeta' model with new data
## first define the data and the model
emptydata<-Data(doseGrid=seq(25,300,25))
data<-emptydata

model<-LogisticIndepBeta(binDLE=c(1.05,1.8),DLEweights=c(3,3),DLEdose=c(25,300),data=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),
           doseGrid=seq(from=25,to=300,by=25))
##update the model to get new estimates
newModel <- update(object=model,data=data)
```

Validate — *A Reference Class to help programming validation for new S4 classes*

Description

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` the message character vector

writeModel — *Creating a WinBUGS model file*

Description

Convert R function to a **WinBUGS** model file. BUGS models follow closely S syntax. It is therefore possible to write most BUGS models as R functions. As a difference, BUGS syntax allows truncation specification like this: `dnorm(...) I(...)` but this is illegal in R. To overcome this incompatibility, use dummy operator `%_%` before `I(...)`: `dnorm(...) %_% I(...)`. The dummy operator `%_%` will be removed before the BUGS code is saved. In S-PLUS, a warning is generated when the model function is defined if the last statement in the model is an assignment. To avoid this warning, add the line `invisible()` to the end of the model definition. This line will be removed before the BUGS code is saved.

Usage

```
writeModel(model, con = "model.bug", digits = 5)
```

Arguments

`model`	R function containing the BUGS model in the BUGS model language, for minor differences see Section Details.
`con`	passed to `writeLines` which actually writes the model file
`digits`	number of significant digits used for **WinBUGS** input, see `formatC`

Value

Nothing, but as a side effect, the model file is written

Author(s)

original idea by Jouni Kerman, modified by Uwe Ligges, Daniel Sabanes Bove removed S-PLUS part

&,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