Package ‘PoDBAY’

January 20, 2025

Type Package

Title Vaccine Efficacy Estimation Package

Version 1.4.3

Depends R (>= 3.6)

Date 2021-09-20

Maintainer Julie Dudasova (MSD) <julie.dudasova@merck.com>

Description Set of functions that implement the PoDBAY method, described in the publication 'A method to estimate probability of disease and vaccine efficacy from clinical trial immunogenicity data' by Julie Dudasova, Regina Laube, Chandni Valiathan, Matthew C. Wiener, Ferdous Gheyas, Pavel Fiser, Justina Ivanauskaite, Frank Liu and Jeffrey R. Sachs (NPJ Vaccines, 2021), <doi:10.1038/s41541-021-00377-6>.

License GPL-3

Copyright 2021 Merck Sharp & Dohme Corp. a subsidiary of Merck & Co., Inc., Kenilworth, NJ, USA

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

LinkingTo Rcpp

Imports Rcpp (>= 1.0.0), ggplot2 (>= 3.1.0), dplyr (>= 0.8.0.1), methods (>= 3.5.2), stats

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr

NeedsCompilation yes

Author Pavel Fiser (MSD) [aut],
Tomas Bartonek (MSD) [aut],
Julie Dudasova (MSD) [aut, cre],
Regina Laube (MSD) [aut]

Repository CRAN

Date/Publication 2021-09-21 13:10:48 UTC

Contents

`assignPoD` *Assign probability of disease (PoD)*

Description

Function assigns subject-level probability of disease based on PoD curve and subject level titer.

Arguments

`x`	numeric vector - vector of estimated PoD values

Details

The input into the function is either calculated using `PoD` function or if the PoD curve is unknown the same arbitrary PoD can be assigned to the whole population.

Value

Subject level probability of disease for the population

`BlindSampling` *Immunogenicity subset: vaccinated, control, non-diseased*

Description

Function creates non-diseased immunogenicity subset, and vaccinated and control immunogenicity subsets based on chosen method. The immunogenicity subsets are provided in the form of population class objects (see the `Population-class` function for more details).

Usage

```
BlindSampling(diseased,
              nondiseased,
              method = list(name = "Full", value = NA))
```

Arguments

`diseased`	`Population-class` object: diseased subjects, created using `ExtractDiseased` function
`nondiseased`	`Population-class` object: non-diseased subjects, created using `ExtractNondiseased` function
`method`	named list: "name" possible inputs "Full", "Ratio", "Fixed"; "value" = numeric value

Details

For details about the method parameter see `ImmunogenicitySubset` function.

Value

- ImmunogenicityVaccinated: vaccinated subjects in the immunogenicity subset, `Population-class` object (N, mean, stdDev, titers)
- ImmunogenicityControl: control subjects in the immunogenicity subset, `Population-class` object (N, mean, stdDev, titers)
- ImmunogenicityNondiseased: non-diseased subjects in the immunogenicity subset, `Population-class` object (N, mean, stdDev, titers)

Examples

```
# Data preparation
data(diseased)
data(nondiseased)

## Example 1
# Creating immunogenicity subset, method = "Full"
ImmunogenicitySubsetFull <-
    BlindSampling(diseased,
                  nondiseased,
                  method = list(name = "Full",
                                value = NA))

## Example 2
# Creating of immunogenicity subset, method = "Ratio"
ImmunogenicitySubsetRatio <-
    BlindSampling(diseased,
                  nondiseased,
                  method = list(name = "Ratio",
                                value = 4))

## Example 3
# Creating of immunogenicity subset, method = "Fixed"
ImmunogenicitySubsetFixed <-
    BlindSampling(diseased,
                  nondiseased,
                  method = list(name = "Fixed",
```

```
                                                     value = 100))
```

`ClinicalTrial` *Clinical trial: estimation of case-count efficacy*

Description

Function assigns disease status (DS) to vaccinated and control groups and based on that calculates the case-count efficacy. Vaccinated and control groups are provided in the form of population class objects (see the `Population-class` function for more details).

Input populations need to contain information about Probability of disease (PoD) for each subject - calculated using `population$assignPoD(PoD(x))`. See `PoD` function for further details.

Usage

```
ClinicalTrial(vaccinated, control, CI = 0.95)
```

Arguments

`vaccinated`	`Population-class` object: vaccinated subjects with assigned PoD
`control`	`Population-class` object: control subjects with assigned PoD
`CI`	numeric: value from (0, 1) interval, confidence level of interest

Value

- vaccinated: vaccinated subjects with assigned DS, `Population-class` object
- control: control subjects with assigned DS, `Population-class` object
- efficacy: case-count efficacy
- confidenceInterval: case-count efficacy confidence interval calculated with `waldCI()` function

Examples

```
# Loading vaccinated, control population data with PoD information
data(vaccinated)
data(control)

# Estimating the disease status and case-count efficacy with 95\% confidence interval
CT <- ClinicalTrial(vaccinated, control, CI = 0.95)

CT$efficacy
CT$confidenceInterval

CT$vaccinated
```

`ClinicalTrialCoverage` *Clinical trial function expanded for usage in simulations when the calculation of coverage probability is needed for three confidence intervals: 80%, 90%, and user-defined*

Description

Function works the same way as `ClinicalTrial` function but it also calculates 80% and 90% confidence intervals.

Usage

```
ClinicalTrialCoverage(vaccinated, control, CI = 0.95)
```

Arguments

`vaccinated`	`Population-class` object: vaccinated subjects with assigned PoD
`control`	`Population-class` object: control subjects with assigned PoD
`CI`	numeric: value from (0, 1) interval, confidence level of interest

Value

- vaccinated: vaccinated subjects with assigned DS, `Population-class` object
- control: control subjects with assigned DS, `Population-class` object
- efficacy: case-count efficacy
- confidenceInterval: confidence interval calculated with `waldCI` function
- confidenceInterval90: 90% confidence interval calculated with `waldCI` function
- confidenceInterval80: 80% confidence interval calculated with `waldCI` function

---

`control` *Dataset containing the information for control subjects*

Description

A dataset containing the N, mean, stdDev, titers of control subjects. The dataset is provided in the form of population class object (see the `Population-class` function for more details).

Usage

```
control
```

Format

Population class object:

N number of subjects

mean mean of titers

stdDev standard deviation of titers

titers subject level titers

`cppMLE` *Maximum likelihood estimation: cpp*

Description

Function calculates the log likelihood value which is used after the initial guesses of the parameters are set in the `PoDMLE` function.

Usage

```
cppMLE(params,
       nondiseasedTiters,
       diseasedTiters,
       adjustTiters = FALSE,
       adjustFrom = log2(10),
       adjustTo = log2(5))
```

Arguments

`params`	named numeric vector: PoD curve parameters ("et50", "slope", "pmax")
`nondiseasedTiters`	numeric vector: non-diseased subjects titers
`diseasedTiters`	numeric vector: diseased subjects titers
`adjustTiters`	boolean: set to TRUE if titer values should be adjusted, for details see `PoD` function
`adjustFrom`	numeric: value specifying the detection limit, all values below the detection limit will be adjusted to adjustTo value
`adjustTo`	numeric: value to which titers below the detection limit will be adjusted

Details

cppMLE function is used inside of PoDMLE function and estimates the PoD curve paramers.

Based on the provided titers for diseased and non-diseased groups the PoD curve parameters which maximize the log likelihood are chosen as optimal.

Difference between MLE and cppMLE is only that cppMLE use cppPoD function instead of PoD. This step significantly improves the computation speed and provides the same results.

Value

log likelihood, numeric value

Examples

```
# Data preparation
data(diseased)
data(nondiseased)
data(PoDParams)

# MLE calculation
cppMLE(PoDParams, nondiseased$titers, diseased$titers)
```

`cppPoD` *Probability of disease calculation*

Description

Function calculates probability of disease (PoD) for given titers according to a PoD curve.

Usage

```
cppPoD(titer, pmax, et50, slope, adjustTiters = FALSE, adjustFrom = 0, adjustTo = 0)
```

Arguments

`titer`	numeric vector: vector of subject level titers
`pmax`	numeric: maximum PoD
`et50`	numeric: titer value corresponding to pmax/2 value, PoD(et50) = pmax/2
`slope`	numeric: slope of the PoD curve
`adjustTiters`	boolean: set to TRUE if titer values should be adjusted, for details see `PoD` function
`adjustFrom`	numeric: value specifying the detection limit, all values below the detection limit will be adjusted to adjustTo value
`adjustTo`	numeric: value to which titers below the detection limit will be adjusted

Details

See `PoD` function for more details. These two functions are equivalent. Usage of cppPoD significantly improves the computation speed over the `PoD` function.

Value

vector of PoDs

`diseased` *Dataset containing the information for diseased subjects*

Description

A dataset containing the N, mean, stdDev, titers of diseased subjects. The dataset is provided in the form of population class object (see the `Population-class` function for more details).

Usage

```
diseased
```

Format

Population class object:

N number of subjects

mean mean of titers

stdDev standard deviation of titers

titers subject level titers

`EfficacyCI` *PoDBAY efficacy summary: mean, median, confidence intervals*

Description

Function summarizes PoDBAY efficacy statistics (mean, median, confidence intervals) based on the set of estimated efficacies and chosen condfidence level. (Set of efficacies is a vector obtained by number of replications specified by repeatCount. These replications are performed for calculation of a confidence interval. For more details, see the supplementary material of the article).

Usage

```
EfficacyCI(efficacySet, ci = 0.95)
```

Arguments

`efficacySet`	numeric vector: estimated PoDBAY efficacies from `PoDBAYEfficacy` function.
`ci`	numeric: required confidence level

Details

Confidence intervals are calculated using quantiles of estimated efficacies.

Value

named list: mean, median, CILow, CIHigh

Examples

```
## Data preparation
data(efficacySet)

## Example 1
EfficacyCI(efficacySet, ci = 0.95)
```

`EfficacyCICoverage` *PoDBAY efficacy summary at three confidence levels*

Description

Function summarizes PoDBAY efficacy statistics (mean, median, confidence intervals) at 80%, 90% and user-defined confidence levels, based on the set of estimated efficacies. (Set of efficacies is a vector obtained by number of replications specified by repeatCount. These replications are performed for calculation of a confidence interval. For more details, see the supplementary material of the article).

Usage

```
EfficacyCICoverage(efficacySet, ci = 0.95)
```

Arguments

`efficacySet`	numeric vector: estimated PoDBAY efficacies from `PoDBAYEfficacy` function.
`ci`	numeric: value from (0, 1) interval, confidence level of interest

Details

Confidence intervals are calculated using quantiles of estimated efficacies.

Value

named list: mean, median, CILow, CIHigh

Examples

```
## Data preparation
data(efficacySet)

## Example 1
EfficacyCICoverage(efficacySet, ci = 0.95)
```

`efficacyComputation` *PoDBAY efficacy equation*

Description

Function calculates the PoDBAY efficacy based on the PoD curve parameters and titer distribution parameters (mean, sd) for vaccinated and control groups.

Usage

```
efficacyComputation(PoDParameters,
                    means = NA,
                    standardDeviations = NA,
                    adjustTiters = FALSE,
                    adjustFrom = NA,
                    adjustTo = NA)
```

Arguments

`PoDParameters`	named data frame ("pmax", "slope", "et50"): PoD curve parameters
`means`	named list ("vaccinated", "control"): mean values of vaccinated and control subjects titers
`standardDeviations`	named list ("vaccinated", "control"): standard deviations of vaccinated and control subjects titers
`adjustTiters`	boolean: set to TRUE if titer values should be adjusted, for details see `PoD` function
`adjustFrom`	numeric: value specifying the detection limit, all values below the detection limit will be adjusted to adjustTo value
`adjustTo`	numeric: value to which titers below the detection limit will be adjusted

Details

$$Efficacy = 1 - \frac{E[PoD_{vaccinated}]}{E[PoD_{control}]}$$

.

E[PoD] for each group is calculated as integral from -Inf to Inf of (titer density function) * (PoD Curve); for further details see Example2 and`ExpectedPoD` function.

Value

efficacy: numeric value

Examples

```
## Data preparation
data(vaccinated)
data(control)
data(PoDParams)

## Example 1
means <- list(vaccinated = vaccinated$mean, control = control$mean)

standardDeviations <- list(vaccinated = vaccinated$stdDev, control = control$stdDev)

efficacyComputation(PoDParams, means, standardDeviations)
```

`efficacySet` *Estimated PoDBAY efficacies*

Description

A dataset containing estimated set of PoDBAY efficacies. (Set of efficacies is a vector obtained by number of replications specified by repeatCount. These replications are performed for calculation of a confidence interval. For more details, see the supplementary material of the article).

Usage

```
efficacySet
```

Format

vector

numeric vector PoDBAY efficacies

`efficacySquaredError` *Optimization objective function: efficacy squared error*

Description

Function calculates squared difference between input (reference value, or for example true in the simulation setup) efficacy and efficacy calculated based on input parameters of PoD curve and input titer distributions of vaccinated and control groups.

Usage

```
efficacySquaredError(params,
                       TrueEfficacy,
                       titerFun,
                       adjustTiters = FALSE,
                       adjustFrom = 0,
                       adjustTo = 0)
```

Arguments

`params`	numeric vector: vector of et50 and slope; efficacy calculation is independent of Pmax and thus Pmax is excluded
`TrueEfficacy`	numeric value: input efficacy value
`titerFun`	list: list of probability density functions for vaccinated and control groups
`adjustTiters`	boolean: set to TRUE if titer values should be adjusted, for details see `PoD` function
`adjustFrom`	numeric: value specifying the detection limit, all values below the detection limit will be adjusted to adjustTo value
`adjustTo`	numeric: value to which titers below the detection limit will be adjusted

Details

Function is used inside the `PoDEfficacySquaredError` function for calculation of the PoD parameters.

Value

Squared difference between calculated and reference efficacy

Examples

```
## Example 1
data(vaccinated)
data(control)
data(PoDParams)

# Choosing et50 and slope as the inputs
params <- list("et50" = 4, "slope" = 6)

# Using probability density function from the populations

titerFun <-
  list(
      function(x) {dnorm(x, mean = vaccinated$mean, sd = vaccinated$stdDev)},
      function(x) {dnorm(x, mean = control$mean, sd = control$stdDev)}
      )

# Assigning true efficacy
TrueEfficacy <- 0.53
```

```
# Sqaured difference between true and calcuated efficacy
efficacySquaredError(params, TrueEfficacy, titerFun)
```

`estimatedParameters` *Estimated PoD curve parameters*

Description

A dataset containing estimated set of PoD curve parameters. (Set of PoD curve parameters is a vector obtained by number of replications specified by repeatCount. These replications are performed for calculation of a confidence interval. For more details, see the supplementary material of the article).

Usage

```
estimatedParameters
```

Format

data frame

pmax pmax: maximum PoD

et50 et50: titer value corresponding to the pmax/2

slope slope: slope of the PoD curve

`ExpectedPoD` *Expected probability of disease*

Description

Function calculates the integral of multiplication of two functions: PoD curve and titer probability density function.

Usage

```
ExpectedPoD(f.pod, f.titer)
```

Arguments

`f.pod`	function(x): PoD curve, estimated sigmoid function relating titers to a probability of disease
`f.titer`	function(x): titer probability density function, distribution of titer values in a group.

Details

Function calculates integral from -Inf to +Inf of titer probability density function multiplied by the PoD curve.

It is used mainly in the PoDBAY efficacy calculation `efficacyComputation`.

Value

Value of the integral of the multiplication of the two functions

Examples

```
# Example 1
data(vaccinated)
data(control)
data(PoDParams)

# Defining the PoD curve
funPoD <- function(x) PoD(x, pmax = PoDParams$pmax, et50 = PoDParams$et50, slope = PoDParams$slope)

# Defining the titer distribution for vaccinated and control groups
funVaccinated <- function(x) dnorm(x, mean = vaccinated$mean, sd = vaccinated$stdDev)
funControl <- function(x) dnorm(x, mean = control$mean, sd = control$stdDev)

# Calculating the expected probability of disease
aucVaccinated <- ExpectedPoD(funPoD, funVaccinated)
aucControl <- ExpectedPoD(funPoD, funControl)

# PoDBAY efficacy estimation
efficacy <- 1 - aucVaccinated/aucControl
```

`ExtractDiseased` *Diseased subjects extraction*

Description

Function extracts diseased subjects from vaccinated and control groups if the data have assigned disease status (for example using `ClinicalTrial` function). The vaccinated and control data are provided in the form of population class objects (see the `Population-class` function for more details).

Usage

```
ExtractDiseased(vaccinated, control)
```

Arguments

`vaccinated`	`Population-class` object: vaccinated subjects with assigned disease status
`control`	`Population-class` object: control subjects with assigned disease status

Value

diseased subjects, `Population-class` object: a subset of control and vaccinated subjects with disease status = TRUE.

Examples

```
## Example 1
# Data preparation
data(vaccinated)
data(control)

# Estimating the disease status and case-count efficacy with CI
ClinicalTrial(vaccinated, control, CI = 0.95)

# Extracting the disease cases
ExtractDiseased(vaccinated, control)
```

`ExtractNondiseased` *Non-diseased subjects extraction*

Description

Function extracts non-diseased subjects from vaccinated and control groups if the data have assigned disease status (for example using `ClinicalTrial` function). The vaccinated and control data are provided in the form of population class objects (see the `Population-class` function for more details).

Usage

```
ExtractNondiseased(vaccinated, control)
```

Arguments

`vaccinated`	`Population-class` object: vaccinated subjects with assigned disease status
`control`	`Population-class` object: control subjects with assigned disease status

Value

non-diseased subjects, `Population-class` object: a subset of control and vaccinated subjects with disease status = FALSE.

Examples

```
## Example 1
# Data preparation
data(vaccinated)
data(control)

# Estimating the disease status and case-count efficacy with CI
ClinicalTrial(vaccinated, control, CI = 0.95)

# Extracting the non-diseased subjects
ExtractNondiseased(vaccinated, control)
```

`fitPoD` *PoD curve: fitting function*

Description

Function calculates the root mean squared error (RMSE) between provided PoD values and calculated PoD values. The latter are calculated using for provided titers and provided PoD curve parameters.

By using the input titers `PoDParamPointEstimation` function and median of the estimated set of PoD curve parameters (output of `PoDParamEstimation` function), the point estimate of PoD curve can be obtained (for details see `PoDParamPointEstimation` function).

Usage

```
fitPoD(params, TitersInput, CurveTitersMedian)
```

Arguments

`params`	named data frame ("pmax", "slope", "et50"): provided PoD curve parameters
`TitersInput`	numeric vector: provided titers
`CurveTitersMedian`	numeric vector: provided PoD values

Details

$$RMSE = \sqrt{\frac{\sum_i^N (PoD_{median}(titers) - PoD_{optimized}(titers))^2}{N}}$$

Value

negative RMSE

Examples

```
## Data preparation
data(estimatedParameters)
data(PoDParams)

## Example 1

# grid of titers
TitersInput <- seq(from = 0, to = 20, by = 0.01)

# for each estimated PoD curve calculate functional values
functionValues <-
  matrix(NA,
         nrow = nrow(estimatedParameters$resultsPriorReset),
         ncol = length(TitersInput))

for (i in 1:nrow(estimatedParameters$resultsPriorReset)) {
  functionValues[i,] <- PoD(TitersInput,
  pmax = estimatedParameters$resultsPriorReset[i,1],
  et50 = estimatedParameters$resultsPriorReset[i,3],
  slope = estimatedParameters$resultsPriorReset[i,2], adjustTiters = FALSE)
}

# functional values corresponding to the median of the estimated PoD curve parameters
CurveTitersMedian <- apply(functionValues, 2, median)

# squared error of CurveTitersMedian and functional values of "params" curve
fitPoD(PoDParams, TitersInput, CurveTitersMedian)
```

`GenerateNondiseased` *Generation of upsampled non-diseased subjects titers*

Description

Function upsamples (by random sampling with replacement) titers from the immunogenicity subset to the required size.

If the size of the immunogenicity subset matches the required size, nothing happens and the original titers from the immunogenicity subset are returned.

Usage

```
GenerateNondiseased(blindNondiseasedTiters, nondiseasedCount)
```

Arguments

`blindNondiseasedTiters`	numeric vector: vector of non-diseased subjects titer values

`nondiseasedCount`
numeric: total number of non-diseased subjects, required size of the non-diseased population

Details

The inputs should come from immunogenicity subset. "nondiseasedCount" represents number of all non-diseased patients in the clinical trial.

Immunogenicity subset populations are obtained from function `BlindSampling`. Immunogenicity subset represents a sample from the non-diseased population.

In this function, sampling with replacement to the required "nondiseasedCount" of the immunogenecitry subset is performed. The function is used inside `PoDParamEstimation` function.

Value

nondiseasedTiters: numeric vector of all non-diseased subjects titers

Examples

```
## Data preparation
data(nondiseased)

## Example 1
# Creating imunogenicity subset, method = "Full"
NondiseasedImmunogenicitySubset <-
    ImmunogenicitySubset(diseased,
                         nondiseased,
                         method = list(name = "Full",
                                       value = "NA"))

# Number of all non-diseased subjects in the clinical trial
nondiseasedGenerationCount <- nondiseased$N

# Upsampling of non-diseased titers
GenerateNondiseased(NondiseasedImmunogenicitySubset$titers, nondiseasedGenerationCount)
```

`generatePopulation` *Population class object generation*

Description

Function generates the population class object using provided summary statistics.

Usage

```
generatePopulation(N, mean, stdDev, unknownDistribution = FALSE, UDFunction = NULL)
```

Arguments

`N`	numeric: number of subjects in the population
`mean`	numeric: mean of titers
`stdDev`	numeric: standard deviation of titers
`unknownDistribution`	logical: TRUE if there is an unknown factor affacting the shape of titer distribution
`UDFunction`	function: function defining the unknown factor affecting the shape of titer distribution

Value

generated population class object with all its characteristics defined in the input parameters

Examples

```
# Example 1: empty population
population0 <- generatePopulation()

# Example 2
population1 <- generatePopulation(N = 100,
                                  mean = 5,
                                  stdDev = 2)
```

`getDiseasedCount` *Diseased count*

Description

Function calculates the number of diseased subjects (disease status = TRUE) in the `Population-class` object.

Details

Input into the function, "diseaseStatus", is taken from the `Population-class` object attribute. Information about disease status is written into the `Population-class` object by the `ClinicalTrial()` function.

Value

numeric: number of the diseased subjects in the `Population-class` object

`getDiseasedTiters` *Diseased titers*

Description

Function returns titers of diseased subjects (disease status = TRUE) in the `Population-class` object.

Details

Input into the function, "diseaseStatus", is taken from the `Population-class` object attribute. Information about disease status is written into the `Population-class` object by the `ClinicalTrial()` function.

Value

numeric vector: titers of diseased subjects in the `Population-class` object

---

`getNondiseasedCount` *Non-diseased count*

Description

Function calculates the number of non-diseased subjects (disease status = FALSE) in the `Population-class` object.

Details

Input into the function, "diseaseStatus", is taken from the `Population-class` object attribute. Information about disease status is written into the `Population-class` object by the `ClinicalTrial()` function.

Value

numeric: number of the non-diseased subjects in the `Population-class` object

`getNondiseasedTiters` *Non-diseased titers*

Description

Function returns titers of non-diseased subjects (disease status = FALSE) in the `Population-class` object.

Details

Input into the function, "diseaseStatus", is taken from the `Population-class` object attribute. Information about disease status is written into the `Population-class` object by the `ClinicalTrial()` function.

Value

numeric vector: titers of non-diseased subjects in the `Population-class` object

`getTiters` *Subject level titers*

Description

Returns subject level titers. If titers are not yet generated, the function generates them based on `Population-class` object attributes: N, mean, stdDev.

Details

Inputs into the function (N, mean, stdDev) are taken from the `Population-class` object attributes.

Value

Subject level titers

`getUnknown` *Generate unknown*

Description

Function generates unknown part of the titers which is eventually added to the original titers in `popX` and to the original titer distribution in `popFun`.

Arguments

`n`	numeric: number of subjects in the population

Details

Input into the function: UDFunction is taken from the `Population-class` object. UDFunction is used for generating the unknown part of the titer distribution.

Value

unknown part of the titers

`ImmunogenicitySubset` *Immunogenicity subset*

Description

Function creates the immunogenicity subset based on the chosen method.

Usage

```
ImmunogenicitySubset(diseased,
                     nondiseased,
                     method = list(name = "Full", value = NA))
```

Arguments

`diseased`	`Population-class` object: diseased subjects with assigned vaccination status
`nondiseased`	`Population-class` object: non-diseased subjects with assigned vacination status
`method`	named list: a selected method for creating the immunogenicity subset method$name • Full: subject level titer information is available for all diseased and all non-diseased subjects, i.e. immunogenicity subset is the full clinical trial • Ratio: subject level titer information is available for all diseased and some non-diseased subjects.

- Fixed: subject level titer information is available for all diseased and some non-diseased subjects.

method$value

- Full: value = NA; immunogenicity sample is the full clinical trial (non-diseased subset contains all non-diseased in the trial; diseased subset contains all disease cases in the trial)
- Ratio: value = number of non-diseased divided by number of diseased subjects; ratio of diseased vs. non-diseased subjects in the immunogenicity subset (non-diseased subset contains only non-diseased subjects, as the selection is done in the end of the study, when the disease status is known; diseased subset contains all disease cases in the trial)
- Fixed: value = size of the immunogenicity subset, pre-defined number of subjects assayed for titers independently of their future disease status (non-diseased subset could rarely contain some diseased subjects, as the selection is done at the enrollment and prior the knowledge of future disease status; diseased subset contains all disease cases in the trial)

Details

The total immunogenicity subset consists of the diseased immunogenicity subset and non-diseased immunogenicity subset. For all three methods implemented, we assume that the diseased immunogenicity subset contains all disease cases in the trial. Based on the chosen method, the the size of the non-diseaded immunogenicity subset can be derived as follows:

Size = number of subjects in the non-diseased immunogenicity subset

Titers = values of titers from which we want to sample in order to simulate the non-diseased immunogenicity subset

#Diseased = total number of diseased in the clinical trial

#Nondiseased = total number of non-diseased in the clinical trial

- method$name = "Full"

  Size = #Nondiseased

  Titers = Nondiseased Titers
- method$name = "Ratio"

  Size = method$value * #Diseased

  Titers = Nondiseased Titers
- method$name = "Fixed"

  Size = method$value

  Titers = Nondiseased Titers + Diseased Titers

Value

Immunogenicity subset with subject level information about vaccination status and disease status, provided in the form of `Population-class` object

Examples

```
## Example 1
# Data preparation
data(diseased)
data(nondiseased)

ImmunogenicitySubset(diseased,
                     nondiseased,
                     method = list(name = "Ratio",
                                   value = 4))
```

`incorrectInput` *Error message*

Description

Error meassage: the input value for "name" is incorrent

Usage

```
incorrectInput(name)
```

Arguments

`name`	name of the input value

Value

error message: "the input value for "name" is incorrect"

`incorrectPopulationInput` *Population class error message*

Description

Error meassage: the input value for "name" is incorrect.

Usage

```
incorrectPopulationInput(name)
```

Arguments

`name`	name of the input value

Value

error message: "The input value for "name" is incorrect. Input needs to be a population class object."

`JitterMean` *Population mean jittering*

Description

Function jitters the mean of the population.

Jittering is adding noise to the mean. The jittered mean is sampled from the distribution with the population mean and population standard deviation divided by the number of subjects in the population. The input population is provided in the form of population class objects (see the `Population-class` function for more details).

$$Mean_{jitter} \sim N(mean, \frac{sd}{N})$$

Usage

```
JitterMean(blindPopulation)
```

Arguments

`blindPopulation`	`Population-class` object with N, mean, stdDev attributes

Value

Jittered mean, numeric value

Examples

```
## Data preparation
data(vaccinated)

## Example 1
vaccinated$mean
JitterMean(vaccinated)
```

`MLE` *Maximum Likelihood estimation*

Description

Function calculates the log likelihood value which is used after the initial guesses of the parameters are set in the `PoDMLE` function.

Usage

```
MLE(params,
    nondiseasedTiters,
    diseasedTiters,
    adjustTiters = FALSE,
    adjustFrom = log2(10),
    adjustTo = log2(5))
```

Arguments

`params`	named numeric vector: PoD curve parameters (et50, slope, pmax)
`nondiseasedTiters`	numeric vector: non-diseased subjects titers
`diseasedTiters`	numeric vector: diseased subjects titers
`adjustTiters`	boolean: set to TRUE if titer values should be adjusted, for details see `PoD` function
`adjustFrom`	numeric: value specifying the detection limit, all values below the detection limit will be adjusted to adjustTo value
`adjustTo`	numeric: value to which titers below the detection limit will be adjusted

Details

MLE function is used inside of PoDMLE function and esimates the PoD curve parameters.

Based on the provided titers for diseased and non-diseased subjects the PoD curve parameters which maximize the log likelihood are chosen as optimal estimates of parameters.

Value

log likelihood, numeric value

Examples

```
# Data preparation
data(diseased)
data(nondiseased)
data(PoDParams)
```

```
# MLE calculation
MLE(PoDParams, nondiseased$titers, diseased$titers)
```

`nondiseased` *Dataset containing the information for non-diseased subjects*

Description

A dataset containing the N, mean, stdDev, titers of non-diseased subjects. The dataset is provided in the form of population class object (see the `Population-class` function for more details).

Usage

```
nondiseased
```

Format

Population class object:

N number of subjects

mean mean of titers

stdDev standard deviation of titers

titers subject level titers

`numToBool` *Numeric to boolean*

Description

Converts numeric format to boolean format.

Usage

```
numToBool(x)
```

Arguments

`x`	numeric value (0, 1)

Details

If the function is supposed to be used on a vector, the form `sapply("vector", numToBool)` needs to be applied.

Value

boolean value (T, F)

Examples

```
dStatus <- c(0,0,1,1,0,1)
sapply(dStatus, numToBool)
```

`PmaxEstimation` *PoD curve paramater, pmax, estimation*

Description

Function finds the pmax parameter of the PoD curve using control subjects summary statistics (mean, sd), observed incidence rate and previsouly estimated et50 and slope by `PoDEfficacySquaredError` function.

Usage

```
PmaxEstimation(IncidenceRate,
               params,
               control,
               adjustTiters = FALSE,
               adjustFrom = NA,
               adjustTo = NA)
```

Arguments

`IncidenceRate`	numeric: observed incidence rate in overall (control) subjects
`params`	numeric vector: et50 and slope
`control`	`Population-class` object: control subjects (mean, sd)
`adjustTiters`	boolean: set to TRUE if titer values should be adjusted, for details see `PoD` function
`adjustFrom`	numeric: value specifying the detection limit, all values below the detection limit will be adjusted to adjustTo value
`adjustTo`	numeric: value to which titers below the detection limit will be adjusted

Value

PoD curve parameter pmax

Examples

```
## Example 1
data(vaccinated)
data(control)

# Assigning true efficacy
TrueEfficacy <- 0.53

# PoD curve parameters (et50, slope) estimation
params <- PoDEfficacySquaredError(TrueEfficacy, vaccinated, control)

# Assigning incidence rate (observed incidence rate)
IncidenceRate <- 0.2

# pmax estimation
pmax <- PmaxEstimation(IncidenceRate, params, control)

# combining PoD curve parameters
PoDParams <- unlist(c(params, pmax))
```

`PoD` *Probability of disease calculation*

Description

Function calculates probability of disease (PoD) corresponding to given titers according to a sigmoid PoD curve.

Usage

```
PoD(titer, pmax, et50, slope, adjustTiters = FALSE, adjustFrom = 0, adjustTo = 0)
```

Arguments

`titer`	numeric vector: subject level titers
`pmax`	numeric: maximum PoD
`et50`	numeric: titer values corresponding to pmax/2 value, PoD(et50) = pmax/2
`slope`	numeric: slope of the PoD curve
`adjustTiters`	boolean: set to TRUE if titer values should be adjusted, for details see `PoD` function
`adjustFrom`	numeric: value specifying the detection limit, all values below the detection limit will be adjusted to adjustTo value
`adjustTo`	numeric: value to which titers below the detection limit will be adjusted

Details

PoD is calculated as:

$$PoD = p_{max} \frac{(\frac{et50}{titer})^{\gamma}}{1 + (\frac{et50}{titer})^{\gamma}},\ for\ titers\ > 0$$

and

$$PoD = pmax,\ for\ titers\ <= 0$$

.

Value

vector of PoDs

Examples

```
data(vaccinated)
data(PoDParams)

PoD(vaccinated$titers, pmax = PoDParams$pmax, et50 = PoDParams$et50, slope = PoDParams$slope)
```

`PoDBAY` *PoDBAY*

Description

PoDBAY package accompanies the article 'A method to estimate probability of disease and vaccine efficacy from clinical trial immunogenicity data'. It helps to setup the workflow for vaccine efficacy estimation and clinical trial simulation using the PoDBAY method.

Details

It has two main applications:

- Estimation of vaccine efficacy using subject level immunogenicity data
- Simulation of clinical trial

Author(s)

Pavel Fiser, Tomas Bartonek, Julie Dudasova

`PoDBAYEfficacy` *PoDBAY efficacy estimation*

Description

Function calculates the PoDBAY efficacy based on the set of PoD curve parameters calculated in `PoDParamEstimation` function, vaccinated and control immunogenicity subset means and standard deviations.

Usage

```
PoDBAYEfficacy(estimatedParameters,
              blindVaccinated,
              blindControl,
              adjustTiters = FALSE,
              adjustFrom = log2(10),
              adjustTo = log2(5))
```

Arguments

`estimatedParameters`	named data frame ("pmax", "slope", "et50"): set of estimated PoD curve parameters
`blindVaccinated`	`Population-class` object: vaccinated subjects from immunogenicity subset, containing N, mean, standard deviation information
`blindControl`	`Population-class` object: control subjects from immunogenicity subset, containing N, mean, standard deviation information
`adjustTiters`	boolean: set to TRUE if titer values should be adjusted, for details see `PoD` function
`adjustFrom`	numeric: value specifying the detection limit, all values below the detection limit will be adjusted to adjustTo value
`adjustTo`	numeric: value to which titers below the detection limit will be adjusted

Details

Application of `efficacyComputation` function to the all PoD curves (each characterized by three PoD parameters) estimated by `PoDParamEstimation` function.

Inputs into the `efficacyComputation` are:

- PoDParameters: i'th estimated PoD parameters from `PoDParamEstimation`. i = 1, ..., N, where N = number of estimations in which MLE converges. See `PoDMLE` for details.
- means: jittered means of immunogenicity subset. See `JitterMeans` for details.
- standardDeviations: standard deviations of the vaccinated and control subjects from the immunogenicity subset.

Value

efficacySet, set of PoDBAY effficacies corresponding to estimated set of PoD curve parameters

Examples

```
## Data preparation
data(diseased)
data(nondiseased)
data(estimatedParameters)

## Example 1
# Creating imunogenicity subset, method = "Ratio", value = 4
ImmunogenicitySubset <-
  BlindSampling(diseased,
                nondiseased,
                method = list(name = "Ratio",
                              value = 4))

# Estimating PoD curve parameters
nondiseasedGenerationCount <- nondiseased$N

estimatedParameters <- PoDParamEstimation(diseased$titers,
                       ImmunogenicitySubset$ImmunogenicityNondiseased$titers,
                       nondiseasedGenerationCount,
                       repeatCount = 10)

# Estimating PoDBAY efficacy
PoDBAYEfficacy(estimatedParameters$results,
              ImmunogenicitySubset$ImmunogenicityVaccinated,
              ImmunogenicitySubset$ImmunogenicityControl)
```

`PoDCI` *PoD curve confidence ribbon*

Description

Supplementary function for `PoDCurvePlot` function. Function calculates the confidence ribbon around the PoD curve.

Usage

```
PoDCI(data, ci = 0.95)
```

Arguments

`data`	numeric vector for which we the confidence intervals should be calculated
`ci`	numeric: required confidence level

Value

lower bound of CI median value upper bound of CI

Examples

```
## Data preparation
data <- 0:100

## Example 1
PoDCI(data,
      ci = 0.95)
```

`PoDCurvePlot` *PoD curve: plot*

Description

Supplementary function for plotting the PoD curve with the confidence ribbon (of a required level). Input values are related to PoDBAY package structure. See `vignette("EfficacyEstimation", package = "PoDBAY")` for an example of application of this function.

Usage

```
PoDCurvePlot(titers,
             estimatedParameters,
             ci = 0.95)
```

Arguments

`titers`	numeric vector: grid of titers at which the confidence ribbon should be calculated
`estimatedParameters`	estimatedParameters named data frame (pmax, slope, et50): set of estimated PoD curve parameters, output of `PoDParamEstimation` function.
`ci`	numeric, required confidence level

Value

PoD curve plot

Examples

```
## Data preparation
library(ggplot2)
data(PoDParams)
data(estimatedParameters)

## Example 1
# titers for which we want calculate the confidence intervals
titers <- seq(from = 0, to = 15, by = 0.01)

# squared error of CurveTitersMedian and functional values of "params" curve
PoDCurvePlot(titers,
             estimatedParameters,
             ci = 0.95)
```

`PoDEfficacySquaredError` *Optimization function: finds PoD curve paramaters (et50, slope)*

Description

Function finds PoD curve parameters (et50, slope) using population summary statistics (mean, sd) and input (reference value, or for example true in the simulation setup) efficacy. Efficacy is independent of pmax parameter thus pmax is estimated separately using `PmaxEstimation` function.

Usage

```
PoDEfficacySquaredError(TrueEfficacy,
                           vaccinated,
                           control,
                           initialSlope = 6,
                           adjustTiters = FALSE,
                           adjustFrom = NA,
                           adjustTo = NA)
```

Arguments

`TrueEfficacy`	numeric: input reference efficacy
`vaccinated`	`Population-class` object: vaccinated group (mean, sd)
`control`	`Population-class` object: control group (mean, sd)
`initialSlope`	numeric: initial slope parameter for the optimization function
`adjustTiters`	boolean: set to TRUE if titer values should be adjusted, for details see `PoD` function
`adjustFrom`	numeric: value specifying the detection limit, all values below the detection limit will be adjusted to adjustTo value
`adjustTo`	numeric: value to which titers below the detection limit will be adjusted

Details

Function returns et50 and slope PoD curve parameters obtained using `efficacySquaredError` i.e. the opimal (output) parameters et50 and slope correspond to the minimal squared difference between input reference efficacy and calculated efficacy.

Pmax parameter is not obtained as efficacy is independent on pmax.

The `optim` function is used for optimization with method = "L-BFGS-B", 1000 maximum itiretations, (0.1,Inf) boundaries for et50 and (-slopeBoundary, slopeBoundary) boundaries for slope.

NOTE: The reason for slope boundary settings is because from certain value of slope parameter the shape of the PoD curve and the corresponding PoD values for given titers are almost identical. This parameter is supposed to limit the resulting slope value and help MLE to converge to optimal parameters. The value of "slopeBoundaries" is calculated from data according to Dunning, 2015 (https://doi.org/10.1186/s12874-015-0096-9).

Value

PoD curve parameters (et50, slope)

Examples

```
## Example 1
data(vaccinated)
data(control)

# Assigning reference efficacy
TrueEfficacy <- 0.53

# PoD curve parameter estimation
PoDEfficacySquaredError(TrueEfficacy, vaccinated, control)
```

`PoDMLE` *Setup for the maximum likelihood estimation (MLE)*

Description

Function estimates the optimal PoD curve parameters (pmax, et50, slope) using diseased and nondiseased titers. Initial guess of the slope parameter needs to be provided as an input to the optimization, as well as the lowTiterPercent parameter, which is needed for initial guess of the pmax parameter calculation.

Usage

```
PoDMLE(nondiseasedTiters,
               diseasedTiters,
               adjustTiters = FALSE,
               adjustFrom = log2(10),
               adjustTo = log2(5),
```

```
                initialSlope = 6,
                lowTiterPercent = 0.2)
```

Arguments

`nondiseasedTiters`	numeric vector: non-diseased subjects titers
`diseasedTiters`	numeric vector: diseased subjects titers
`adjustTiters`	boolean: set to TRUE if titer values should be adjusted, for details see `PoD` function
`adjustFrom`	numeric: value specifying the detection limit, all values below the detection limit will be adjusted to adjustTo value
`adjustTo`	numeric: value to which titers below the detection limit will be adjusted
`initialSlope`	numeric: initial guess of the slope parameter for the optimization function
`lowTiterPercent`	numeric: value in the interval (0,1) - it represents a fraction of bottom titer values of the whole clinical trial used for calculation of inital guess of the pmax parameter.

Details

Initial guess of pmax = (number of diseased in the bottom titers + 0.5) / (number of non-diseased and diseased in the bottom titers + 0.5), Initial et50 = intersection point of distributions of non-diseased and diseased groups. If L-BFGS-B optimization fails to converge, a new et50 initial guess is set to median value of all titers.

PoDMLE function estimates the PoD curve parameters by maximizing the likelihood value (see `MLE` function for details) based on the provided titers for diseased and non-diseased groups.

The `optim` function is used for optimization with method = "L-BFGS-B", 500 maximum iterations, (0.1,Inf) boundaries for et50, (1e-6,1) boundaries for pmax and (-slopeBoundary, slopeBoundary) boundaries for slope.

NOTE: The reason for slope boundary settings is because from certain value of slope parameter the shape of the PoD curve and the corresponding PoD values for given titers are almost identical. This parameter is expected to limit the resulting slope value and help MLE to converge to optimal parameters. The value of "slopeBoundaries" is calculated as described by Dunning, 2015 (https://doi.org/10.1186/s12874-015-0096-9).

Value

list("et50", "slope", "pmax"), named list of PoD paraters: if MLE converges.

Null: if MLE does not converge.

Examples

```
## EXAMPLE 1:
# Data preparation
data(diseased)
data(nondiseased)
```

```
# PoD curve parameter estimation
PoDMLE(nondiseased$titers,
       diseased$titers)

## EXAMPLE 2:
## initialSlope and lowTiterPercent variables are adjusted.
PoDMLE(nondiseased$titers,
       diseased$titers,
       initialSlope = 5,
       lowTiterPercent = 0.3)
```

`PoDParamEstimation` *PoD curve parameters estimation*

Description

Function estimates the PoD curve parameters (pmax, slope, et50) using `PoDMLE` function. Number of PoD curves estimated equals to the repeatCount input parameter.

The estimation is performed using provided diseased and non-diseased subject level data.

Usage

```
PoDParamEstimation(diseasedTiters,
                   nondiseasedTiters,
                   nondiseasedGenerationCount,
                   repeatCount = 500,
                   adjustTiters = FALSE,
                   adjustFrom = log2(10),
                   adjustTo = log2(5))
```

Arguments

`diseasedTiters`	numeric vector: all diseased titers, subject level data
`nondiseasedTiters`	numeric vector: non-diseased titers from immunogenicity subset, subject level data
`nondiseasedGenerationCount`	numeric: total number of non-diseased subjects in the clinical trial
`repeatCount`	numeric: how many times is the dataset bootstrapped and the PoD curve parameter estimation performed
`adjustTiters`	boolean: set to TRUE if titer values should be adjusted, for details see `PoD` function
`adjustFrom`	numeric: value specifying the detection limit, all values below the detection limit will be adjusted to adjustTo value
`adjustTo`	numeric: value to which titers below the detection limit will be adjusted

Details

diseasedTiters: subject level titers of all diseased in the clinical trial

nondiseasedTiters: subject level titers of non-diseased subjects in the immunogenicity subset

There are two possible scenarios

- Full: Full information about non-diseased titers is available, i.e subject level data for all non-diseased subjects from the clinical trial (nondiseasedGenerationCount = number of all non-diseased subjects in the clinical trial).
- Ratio or Fixed: Information about non-diseased titers is available only for the immunogenicity subset. In order to compensate for these missing titers we upsampling of this subset to the total number of non-diseased (nondiseasedGenerationCount) in the trial is needed.

nondiseasedGenerationCount: number of all non-diseased subjects in the clinical trial

NOTE: Number of estimated parameters can be lower than repeatCount as MLE does not necessary converge in all estimations; failcount (number of iterations in which MLE failed to converge) is also returned; for details see `MLE` function.

Function steps

- Upsample non-diseased if needed (needed for methods Ratio and Fixed) - from immunogenicity subset size (N = NondiseasedImmunogenicitySubset$N) to the whole trial size (N = nondiseasedGenerationCount). For details see `GenerateNondiseased` function.
- Estimate PoD curve: resultsPriorReset
- Reset disease status: the purpose is to estimate the confidence intervals of the PoD curve and its parameters

  Part of the reset disease status procedure is the non-parametric bootstrap: titers of diseased and non-diseased subjects are pooled, and associated PoDs are calculated using their titer values and estimated PoD curve. Based on the subject level probabilities (PoDs), the disease status is reestimated.
- Re-estimate PoD curve: new diseased and non-diseased titers are used to reestimate the PoD curve

Value

results: PoD curve parameters after resetting the disease status, named data.frame of estimated PoD curve parameters (pmax, slope, et50); see details for more information

resultsPriorReset: PoD curve parameters prior to resetting the status, named data.frame of estimated PoD curve parameters (pmax, slope, et50); see details for more information

failcount: number of iterations in which MLE failed to converge; see details for more information

Examples

```
## Data preparation
data(diseased)
data(nondiseased)

## Example 1
```

```
# Creating imunogenicity subset, method = "Full"
NondiseasedImmunogenicitySubset <-
    ImmunogenicitySubset(diseased,
                         nondiseased,
                         method = list(name = "Full",
                                       value = "NA"))

# Number of all non-diseased subjects in the clinical trial
nondiseasedGenerationCount <- nondiseased$N

PoDParamEstimation(diseased$titers,
                   NondiseasedImmunogenicitySubset$titers,
                   nondiseasedGenerationCount,
                   repeatCount = 10)

## Example 2
# Creating imunogenicity subset, method = "Ratio", value = 4
NondiseasedImmunogenicitySubset <-
    ImmunogenicitySubset(diseased,
                         nondiseased,
                         method = list(name = "Ratio",
                                       value = 4))

# Number of all non-diseased subjects in the clinical trial
nondiseasedGenerationCount <- nondiseased$N

PoDParamEstimation(diseased$titers,
                   NondiseasedImmunogenicitySubset$titers,
                   nondiseasedGenerationCount,
                   repeatCount = 10)
```

`PoDParamPointEstimation` *PoD curve point estimate*

Description

Function returns PoD curve parameters corresponding to the point estimate of PoD curve.

Usage

```
PoDParamPointEstimation(resultsPriorReset,
                        titers = seq(from = 0, to = 20, by = 0.01),
                        optim_titers = FALSE)
```

Arguments

`resultsPriorReset`	named data frame ("pmax", "slope", "et50"): set of estimated PoD curve parameters before resetting the disease status; for further details see `PoDParamEstimation` function.
`titers`	numeric vector: a grid of titers for PoD curve point estimate calculation
`optim_titers`	logical: TRUE for a predefined sequence of titers

Details

For each of estimated PoD curves in resultsPriorReset, the function values (probabilities of disease, PoD) for provided grid of titers are calculated.

Median of function values (PoDs) at each provided titer is calculated.

Subsequently, the PoD curve model is fitted to the median datapoins using `fitPoD` function, in order to get PoD curve parameters close to this median curve.

Value

paramsPointEstimate: named data frame of PoD curve parameters corresponding to the PoD curve point estimate

Examples

```
## Data preparation
data(estimatedParameters)

## Example 1
# titers for which we want to optimize the functional values
titers <- seq(from = 0, to = 20, by = 0.01)

# Point estimate of PoD curve
PoDParamPointEstimation(estimatedParameters$resultsPriorReset, titers)
```

`PoDParams` *PoD curve parameters*

Description

A dataset containing PoD curve parameters

Usage

```
PoDParams
```

Format

data frame

pmax pmax: maximum PoD

et50 et50: titer value corresponding to the pmax/2

slope slope: slope of the PoD curve

`PoDParamsCI` *Confidence intervals of PoD curve parameters*

Description

Function calculates confidence intervals of the PoD curve parameters (pmax, et50, slope) at user-defined confidence level.

Usage

```
PoDParamsCI(estimatedParameters, ci = 0.95)
```

Arguments

`estimatedParameters`	output of `PoDParamEstimation` function
`ci`	numeric: value from (0, 1) interval, confidence level of interest

Value

CI of all PoD curve parameters

`PoDParamsCICoverage` *Confidence intervals of PoD curve parameters at three confidence levels*

Description

Function calculates confidence intervals (80%, 90% and user-defined) of the PoD curve parameters (pmax, et50, slope).

Usage

```
PoDParamsCICoverage(estimatedParameters, ci = 0.95)
```

Arguments

`estimatedParameters`	output of `PoDParamEstimation` function
`ci`	numeric: value from (0, 1) interval, confidence level of interest

Value

CI of all PoD curve parameters

`popFun` *Population function*

Description

Function describing the titer distribution of the population: mean, standard deviation and an additional unknown factor affecting the shape of the distribution (e.g. mixture of two normals or other shapes defined by user).

Details

Inputs into the function (mean, stdDev, Unknowndistribution) and getUnknown method are taken from the `Population-class` object.

Value

Titer distribution function

`Population-class` *Population class*

Description

Population reference class which provides summary and subject level information about the population

Fields

`N` numeric: number of subjects in the population

`mean` numeric: mean value of titers

`stdDev` numeric: standard deviation of titers

`unknownDistribution` logical: TRUE if titer distribution is not normally /log-normally distributed; titer disrtibution function needs to be defined by user

`UDFunction` function: user-defined titer distribution

`titers` numeric: subject level titers, generated with `getTiters` method

`PoDs` numeric: subject level probability of disease (PoD), generated with `assginPoD` method

`diseaseStatus` logical: subject level disease status (TRUE if diseased), generated with `ClinicilaTrial` function

---

`popX` *Add noise to population titers*

Description

Function adds noise to population titers accounting for an unknown factor affecting the titer distibution.

Details

Inputs into the function: N, unknownDistribution and getUnknown() method are taken from the `Population-class` object.

Value

subject level titers

`vaccinated` *Dataset containing the information for vaccinated subjects*

Description

A dataset containing the N, mean, stdDev, titers of vaccinated subjects. The dataset is provided in the form of population class object (see the `Population-class` function for more details).

Usage

```
vaccinated
```

Format

Population class object:

N number of subjects

mean mean of titers

stdDev standard deviation of titers

titers subject level titers

`waldCI` *Wald confidence interval estimation*

Description

Function calculates and returns case-count efficacy confidence intervals estimated using Wald's method.

Input data need to contain information about disease status on individual level.

Usage

```
waldCI(vaccinated, control, confLevel = 0.95)
```

Arguments

`vaccinated`	`Population-class` object: vaccinated subjects, containing information about disease status
`control`	`Population-class` object: control subjects, containing information about disease status
`confLevel`	numeric: value from (0, 1) interval, confidence level of interest

Details

Confidence interval of the relative risk is calculated using the Wald method. (Wald, A. Tests of statistical hypotheses concerning several parameters when the number of observations is large. Transactions of the American Mathematical Society 54, 426-482 (1943)).

Value

Named list of lower and upper confidence interval bound

Examples

```
# Loading vaccinated and control populations data with PoD information
data(vaccinated)
data(control)

# Estimating the disease status and case-count efficacy with 95\% confidence interval
set.seed(1)
CT <- ClinicalTrial(vaccinated, control, CI = 0.95)

waldCI(vaccinated, control)
```

Index