Package 'BayesCTDesign'

May 6, 2026

Type Package

Title Two Arm Bayesian Clinical Trial Design with and Without Historical Control Data

Version 0.6.1

Description A set of functions to help clinical trial researchers calculate power and sample size for two-arm Bayesian randomized clinical trials that do or do not incorporate historical control data. At some point during the design process, a clinical trial researcher who is designing a basic two-arm Bayesian randomized clinical trial needs to make decisions about power and sample size within the context of hypothesized treatment effects. Through simulation, the simple_sim() function will estimate power and other user specified clinical trial characteristics at user specified sample sizes given user defined scenarios about treatment effect,control group characteristics, and outcome. If the clinical trial researcher has access to historical control data, then the researcher can design a two-arm Bayesian randomized clinical trial that incorporates the historical data. In such a case, the researcher needs to work through the potential consequences of historical and randomized control differences on trial characteristics, in addition to working through issues regarding power in the context of sample size, treatment effect size, and outcome. If a researcher designs a clinical trial that will incorporate historical control data, the researcher needs the randomized controls to be from the same population as the historical controls. What if this is not the case when the designed trial is implemented? During the design phase, the researcher needs to investigate the negative effects of possible historic/randomized control differences on power, type one error, and other trial characteristics. Using this information, the researcher should design the trial to mitigate these negative effects. Through simulation, the historic_sim() function will estimate power and other user specified clinical trial characteristics at user specified sample sizes given user defined scenarios about historical and randomized control differences as well as treatment effects and outcomes. The results from historic_sim() and simple_sim() can be printed with print_table() and graphed with plot_table() methods. Outcomes considered are Gaussian, Poisson, Bernoulli, Lognormal, Weibull, and Piecewise Exponential. The methods are described in Eggleston et al. (2021) <doi:10.18637/jss.v100.i21>.

Depends R (>= 3.5.0)

License GPL-3

Encoding UTF-8

URL https://github.com/begglest/BayesCTDesign

BugReports https://github.com/begglest/BayesCTDesign/issues

Imports eha (>= 2.9.0), ggplot2 (>= 2.2.1), survival (>= 2.41-3), reshape2 (>= 1.4.3), stats (>= 3.5.0)

RoxygenNote 7.1.2

NeedsCompilation no

Author Barry Eggleston [cre, aut], Doug Wilson [aut], Becky McNeil [aut], Joseph Ibrahim [aut], Diane Catellier [fnd, rth, aut]

Maintainer Barry Eggleston <beggleston@rti.org>

Repository CRAN

Date/Publication 2021-11-30 11:20:02 UTC

Contents

`genbernoullidata` *Generating function for Bernoulli Data.*

Description

`genlogisticdata()` function used mainly internally by `logistictrialsimulator()` function to generate data for a two-arm clinical trial, experimental and control groups. Can be used to generate random trial data.

Usage

```
genbernoullidata(sample_size, prob1, odds_ratio)
```

Arguments

sample_size	Number of subjects per arm.
prob1	prob parameter used in call to rbinom(). Used only in control arm.
odds_ratio	Desired Odds Ratio between experimental and control groups.

Value

genlogisticdata() returns a data frame with columns: 'id', 'treatment', and 'y'.

Examples

```
samplehistdata <- genbernoullidata(sample_size=60, prob1=0.6, odds_ratio=0.6)
samplehistdata
```

---

gengaussiandata *Generating function for Gaussian Data.*

Description

gengaussiandata() function used mainly internally by gaussiantrialsimulator() function to generate data for a two-arm clinical trial, experimental and control groups. Can be used to generate random trial data.

Usage

```
gengaussiandata(sample_size, mu1, mean_diff, common_sd)
```

Arguments

sample_size	Number of subjects per arm.
mu1	mean parameter used in call to rnorm(). Used only in control arm.
mean_diff	Desired Mean Difference between experimental and control groups.
common_sd	sd parameter used in call to rnorm(). Used in both arms.

Value

gengaussiandata() returns a data frame with columns: 'id', 'treatment', and 'y'.

Examples

```
samplehistdata <- gengaussiandata(sample_size=60, mu1=25, mean_diff=0, common_sd=3)
samplehistdata
```

genlognormaldata *Generating function for Lognormal Data.*

Description

genlognormaldata() function used mainly internally by lognormaltrialsimulator() and lognormaltrialsimulatornc functions to generate data for a two-arm clinical trial, experimental and control groups. Can be used to generate random trial data.

Usage

```
genlognormaldata(sample_size, mu1, mean_ratio, common_sd, censor_value)
```

Arguments

sample_size	Number of subjects per arm.
mu1	meanlog parameter used in call to rlnorm(). Used only in control arm.
mean_ratio	Desired Mean Ratio between experimental and control groups.
common_sd	sdlog parameter used in call to rlnorm(). Used in both arms.
censor_value	Value at which time-to-event data are right censored.

Value

genlognormaldata() returns a data frame with columns: 'id', 'treatment', 'event_time', and 'status'.

Examples

```
samplehistdata <- genlognormaldata(sample_size=60, mu1=1.06, mean_ratio=0.6,
                                   common_sd=1.25, censor_value=3)
samplehistdata
```

genpoissondata *Generating function for Poisson Data.*

Description

genpoissondata() function mainly used internally by poissontrialsimulator() function to generate data for a two-arm clinical trial, experimental and control groups. Can be used to generate random trial data.

Usage

```
genpoissondata(sample_size, mu1, mean_ratio)
```

Arguments

sample_size	Number of subjects per arm.
mu1	lambda parameter used in call to rpois(). Used only in control arm.
mean_ratio	Desired Mean Ratio between experimental and control groups.

Value

genpoissondata() returns a data frame with columns: 'id', 'treatment', and 'y'.

Examples

```
samplehistdata <- genpoissondata(sample_size=60, mu1=1, mean_ratio=1.0)
samplehistdata
```

---

genpwedata *Generating function for Piece-wise Exponential Data.*

Description

genpwedata() function used mainly internally by pwetrialsimulator() function to generate data for a two-arm clinical trial, experimental and control groups. Can be used to generate random trial data.

Usage

```
genpwedata(sample_size, lambda_vec, hazard_ratio, time_vec, censor_value)
```

Arguments

sample_size	Number of subjects per arm.
lambda_vec	Set of lambdas passed to eha::rpch() through the levels parameter. Used only in control arm.
hazard_ratio	Desired Hazard Ratio between experimental and control groups.
time_vec	Set of cutpoints passed to eha::rpch() through the cuts parameter.
censor_value	Value at which time-to-event data are right censored.

Value

genpwedata() returns a data frame with columns: 'id', 'treatment', 'event_time', 'status', and 'indicator'.

Examples

```
nvalHC <- 60
time.vec <- c(0.3,0.9,1.5,2.1,2.4)
lambdaHC.vec <- c(0.19,0.35,0.56,0.47,0.38,0.34)
censor.value <- 3

SampleHistData <- genpwedata(nvalHC, lambdaHC.vec, 1.0, time.vec, censor.value)
SampleHistData
```

---

genweibulldata *Generating function for Weibull Data.*

Description

genweibulldata() function used mainly internally by weibulltrialsimulator() and weibulltrialsimulatornohist() functions to generate data for a two-arm clinical trial, experimental and control groups. Can be used to generate random trial data.

Usage

```
genweibulldata(sample_size, scale1, hazard_ratio, common_shape, censor_value)
```

Arguments

sample_size	Number of subjects per arm.
scale1	Scale parameter used in call to rweibull(). Used only in control arm.
hazard_ratio	Desired Hazard Ratio between experimental and control groups.
common_shape	Shape parameter used in call to rweibull(). Used in both arms.
censor_value	Value at which time-to-event data are right censored.

Value

genweibulldata() returns a data frame with columns: 'id', 'treatment', 'event_time', and 'status'.

Examples

```
SampleHistData <- genweibulldata(sample_size=60, scale1=2.82487,
                                 hazard_ratio=0.6, common_shape=3,
                                 censor_value=3)
SampleHistData
```

historic_sim *Two Arm Bayesian Clinical Trial Simulation with Historical Data*

Description

historic_sim() returns an S3 object of class bayes_ctd_array, which will contain simulation results for power, statistic estimation, bias, variance, and mse as requested by user.

Usage

```
historic_sim(
  trial_reps = 100,
  outcome_type = "weibull",
  subj_per_arm = c(50, 100, 150, 200, 250),
  a0_vals = c(0, 0.33, 0.67, 1),
  effect_vals = c(0.6, 1, 1.4),
  rand_control_diff = c(0.8, 1, 1.2),
  hist_control_data = NULL,
  time_vec = NULL,
  censor_value = NULL,
  alpha = 0.05,
  get_var = FALSE,
  get_bias = FALSE,
  get_mse = FALSE,
  seedval = NULL,
  quietly = TRUE
)
```

Arguments

trial_reps	Number of trials to replicate within each combination of a0_vals, subj_per_arm, effect_vals, and rand_control_parms. As the number of trials increases, the precision of the estimate will increase. Default is 100.
outcome_type	Outcome distribution. Must be equal to weibull, lognormal, pwe (Piecewise Exponential), gaussian, bernoulli, or poisson. Default is weibull.
subj_per_arm	A vector of sample sizes, all of which must be positive integers. Default is c(50, 100, 150, 200, 250).
a0_vals	A vector of power prior parameters ranging from 0 to 1, where 0 implies no information from historical data should be used, and 1 implies all of the information from historical data should be used. A value between 0 and 1 implies that a proportion of the information from historical data will be used. Default is c(0, 0.33, 0.67, 1).
effect_vals	A vector of effects that should be reasonable for the outcome_type being studied, hazard ratios for Weibull, odds ratios for Bernoulli, mean ratios for Poisson, etc.. When effect_vals contain the null effect for a given outcome_type, the power component of data will contain an estimate of Type One Error. In

order to have a good set of Type One Error estimates, trial_reps need to be at least 10,000. In such a case, if the total number of combinations made up from subj_per_arm, a0_vals, effect_vals, and rand_control_diff is very large, the time to complete the simulation can be substantial. Default is c(0.6, 1, 1.4).

rand_control_diff
: For piecewise exponential and Weibull outcomes, this is a vector of hazard ratios (randomized controls over historical controls) representing differences between historical and randomized controls. For lognormal and Poisson outcomes, this is a vector of mean ratios (randomized controls over historical controls). For a Bernoulli outcome, this is a vector of odds ratios (randomized controls over historical controls). For a Gaussian outcome, this is a vector of mean differences (randomized minus historical controls). Default is c(0.8, 1, 1.2).

hist_control_data
: A dataset of historical data. Default is NULL. For survival outcomes, historical datasets must have 4 columns: id, treatment, event_time, and status. The value of treatment should be 0. For other outcomes, historical datasets must have columns: id, treatment, and y.

time_vec
: A vector of time values which are used to create time periods within which the exponential hazard is constant. Only used for piecewise exponential models. Default is NULL.

censor_value
: A single value at which right censoring occurs when simulating randomized subject outcomes. Used with survival outcomes. Default is NULL, where NULL implies no right censoring.

alpha
: A number ranging between 0 and 1 that defines the acceptable Type 1 error rate. Default is 0.05.

get_var
: A TRUE/FALSE indicator of whether an array of variance estimates will be returned. Default is FALSE.

get_bias
: A TRUE/FALSE indicator of whether an array of bias estimates will be returned. Default is FALSE.

get_mse
: A TRUE/FALSE indicator of whether an array of MSE estimates will be returned. Default is FALSE.

seedval
: A seed value for pseudo-random number generation.

quietly
: A TRUE/FALSE indicator of whether notes are printed to output about simulation progress as the simulation runs. If running interactively in RStudio or running in the R console, quietly can be set to FALSE. If running in a Notebook or knitr document, quietly needs to be set to TRUE. Otherwise each note will be printed on a separate line and it will take up a lot of output space. Default is TRUE.

Details

The object bayes_ctd_array has 6 elements: a list containing simulation results (data), copies of the 4 function arguments subj_per_arm, a0_vals, effect_vals, and rand_control_diff, and finally a objtype value indicating that historic_sim() was used. Each element of data is a four-dimensional array, where each dimension is determined by the length of parameters subj_per_arm,

a0_vals, effect_vals, and rand_control_diff. The size of data depends on which results are requested by the user. At a minimum, at least one of subj_per_arm, a0_vals, effect_vals, or rand_control_diff must contain at least 2 values, while the other three must contain at least 1 value. The data list will always contain two elements: an array of power results (power) and an array of estimation results (est). In addition to power and est, data may also contain elements var, bias, or mse, depending on the values of get_var, get_bias, and get_mse. The values returned in est are in the form of hazard ratios, mean ratios, odds ratios, or mean differences depending on the value of outcome_type. For a Gaussian outcome, the estimation results are differences in group means (experimental group minus control group). For a logistic outcome, the estimation results are odds ratios (experimental group over control group). For lognormal and Poisson outcomes, the estimation results are mean ratios (experimental group over control group). For a piecewise exponential or a Weibull outcome, the estimation results are hazard ratios (experimental group over control group). The values returned in bias, var, and mse are on the scale of the values returned in est.

The object bayes_ctd_array has two primary methods, print() and plot(), for printing and plotting slices of the arrays contained in bayes_ctd_array$data.

As dimensions of the four dimensional array increases, the time required to complete the simulation will increase; however, it will be faster than a similar simulation based on repeated calls to MCMC routines to analyze each simulated trial.

The meaning of the estimation results, and the test used to generate power results, depends on the outcome used. In all cases, power is based on a two-sided test involving a (1-alpha)100% credible interval, where the interval is used to determine if the null hypothesis should be rejected (null value outside of the interval) or not rejected (null value inside the interval). For a Gaussian outcome, the 95% credible interval is an interval for the difference in group means (experimental group minus control group), and the test determines if 0 is in or outside of the interval. For a Bernoulli outcome, the 95% credible interval is an interval for the odds ratio (experimental group over control group), and the test determines if 1 is in or outside of the interval. For a lognormal or a Poisson outcome, the 95% credible interval is an interval for the mean ratio (experimental group over control group), and the test determines if 1 is in or outside of the interval. Finally, for a piecewise exponential or a Weibull outcome, the 95% credible interval is an interval for the hazard ratio (experimental group over control group), and the test determines if 1 is in or outside of the interval.

Please refer to the examples for illustration of package use.

Value

historic_sim() returns an S3 object of class bayes_ctd_array. As noted in details, an object of class bayes_ctd_array has 6 elements: a list of simulation results (data), copies of the 4 function arguments subj_per_arm, a0_vals, effect_vals, and rand_control_diff, and finally objtype indicating that historic_sim() was used. See details for a discussion about the contents of data. Results from the simulation contained in the bayes_ctd_array object can be printed or plotted using the print() and plot() methods. The results can also be accessed using basic list element identification and array slicing. For example, to get the 4-dimensional array of power results from a simulation, one could use the code bayes_ctd_array$data$power, where bayes_ctd_array is replaced with the name of the variable containing the bayes_ctd_array object. If one wanted a table of power for sample size by a0, while holding effect equal to the first considered value and control differences equal to the second considered value, then the code is bayes_ctd_array$data$power[,,1,2], where bayes_ctd_array is replaced with the name of the variable containing the bayes_ctd_array object.

Examples

```
#Generate a sample of historical data for use in example.
set.seed(2250)
SampleHistData <- genweibulldata(sample_size=60, scale1=2.82487,
                                 hazard_ratio=0.6, common_shape=3,
                                 censor_value=3)
histdata <- subset(SampleHistData, subset=(treatment==0))
histdata$id <- histdata$id+10000

#Run a Weibull simulation, using historic_sim().
#For meaningful results, trial_reps needs to be much larger than 2.
weibull_test <- historic_sim(trial_reps = 2, outcome_type = "weibull",
                             subj_per_arm = c(50, 100, 150),
                             a0_vals = c(0, 0.50, 1),
                             effect_vals = c(0.6, 1),
                             rand_control_diff = c(0.8, 1),
                             hist_control_data = histdata, time_vec = NULL,
                             censor_value = 3, alpha = 0.05, get_var = TRUE,
                             get_bias = TRUE, get_mse = TRUE, seedval=123,
                             quietly=TRUE)

#Tabulate the simulation results for power.
test_table <- print(x=weibull_test, measure="power",
                    tab_type="WX|YZ", effect_val=0.6,
                    rand_control_diff_val=1.0)
print(test_table)


#Create a plot of the power simulation results.
plot(x=weibull_test, measure="power", tab_type="WX|YZ",
     smooth=FALSE, plot_out=TRUE, effect_val=0.6,
     rand_control_diff_val=1.0)
#Create a plot of the estimated hazard ratio simulation results.
plot(x=weibull_test, measure="est", tab_type="WX|YZ",
     smooth=FALSE, plot_out=TRUE, effect_val=0.6,
     rand_control_diff_val=1.0)
#Create a plot of the hazard ratio variance simulation results.
plot(x=weibull_test, measure="var", tab_type="WX|YZ",
     smooth=FALSE, plot_out=TRUE, effect_val=0.6,
     rand_control_diff_val=1.0)
#Create a plot of the hazard ratio bias simulation results.
plot(x=weibull_test, measure="bias", tab_type="WX|YZ",
     smooth=FALSE, plot_out=TRUE, effect_val=0.6,
     rand_control_diff_val=1.0)
#Create a plot of the hazard ratio mse simulation results.
plot(x=weibull_test, measure="mse", tab_type="WX|YZ",
     smooth=FALSE, plot_out=TRUE, effect_val=0.6,
     rand_control_diff_val=1.0)

#Create other power plots using different values for tab_type
plot(x=weibull_test, measure="power", tab_type="XY|WZ",
     smooth=FALSE, plot_out=TRUE, subj_per_arm_val=150,
```

```
     rand_control_diff_val=1.0)

plot(x=weibull_test, measure="power", tab_type="XZ|WY",
     smooth=FALSE, plot_out=TRUE, subj_per_arm_val=150, effect_val=0.6)

plot(x=weibull_test, measure="power", tab_type="YZ|WX",
     smooth=FALSE, plot_out=TRUE, subj_per_arm_val=150, a0_val=0.5)

plot(x=weibull_test, measure="power", tab_type="WY|XZ",
     smooth=FALSE, plot_out=TRUE, rand_control_diff_val=1, a0_val=0.5)

plot(x=weibull_test, measure="power", tab_type="WZ|XY",
     smooth=FALSE, plot_out=TRUE, effect_val=0.6, a0_val=0.5)



#Run Poisson simulation, using historic_sim(), but set two design characteristic
# parameters to only 1 value.
#Note: historic_sim() can take a while to run.
#Generate a sample of historical poisson data for use in example.
set.seed(2250)
samplehistdata <- genpoissondata(sample_size=60, mu1=1, mean_ratio=1.0)
histdata <- subset(samplehistdata, subset=(treatment==0))
histdata$id <- histdata$id+10000

#For meaningful results, trial_reps needs to be larger than 100.
poisson_test <- historic_sim(trial_reps = 100, outcome_type = "poisson",
                             subj_per_arm = c(50, 75, 100, 125, 150, 175, 200, 225, 250),
                               a0_vals = c(1),
                               effect_vals = c(0.6),
                               rand_control_diff = c(0.6, 1, 1.6),
                               hist_control_data = histdata, time_vec = NULL,
                               censor_value = 3, alpha = 0.05, get_var = TRUE,
                               get_bias = TRUE, get_mse = TRUE, seedval=123,
                               quietly=TRUE)

#Tabulate the simulation results for power.
test_table <- print(x=poisson_test, measure="power",
                    tab_type=NULL)
print(test_table)

#Create a plot of the power simulation results.
plot(x=poisson_test, measure="power", tab_type=NULL,
     smooth=FALSE, plot_out=TRUE)



#At least one of subj_per_arm, a0_vals, effect_vals, or rand_control_diff
#must contain at least 2 values.
#Generate a sample of historical lognormal data for use in example.
set.seed(2250)
samplehistdata <- genlognormaldata(sample_size=60, mu1=1.06, mean_ratio=0.6, common_sd=1.25,
                                   censor_value=3)
```

```
histdata <- subset(samplehistdata, subset=(treatment==0))
histdata$id <- histdata$id+10000

#Run a Lognormal simulation, using historic_sim().
#For meaningful results, trial_reps needs to be larger than 100.
lognormal_test <- historic_sim(trial_reps = 100, outcome_type = "lognormal",
                               subj_per_arm = c(25,50,75,100,125,150,175,200,225,250),
                               a0_vals = c(1.0),
                               effect_vals = c(0.6),
                               rand_control_diff = c(1.8),
                               hist_control_data = histdata, time_vec = NULL,
                               censor_value = 3, alpha = 0.05, get_var = TRUE,
                               get_bias = TRUE, get_mse = TRUE, seedval=123,
                               quietly=TRUE)

test_table <- print(x=lognormal_test, measure="power",
                    tab_type=NULL)
print(test_table)
#Create a plot of the power simulation results.
plot(x=lognormal_test, measure="power", tab_type=NULL,
     smooth=TRUE, plot_out=TRUE)
```

---

plot.bayes_ctd_array *Plot Data from Two Arm Bayesian Clinical Trial Simulation.*

Description

plot.bayes_ctd_array() takes an S3 object of class bayes_ctd_array, and creates a line plot from a one or two dimensional slice of the data generated by a clinical trial simulation using historic_sim() or simple_sim(). The plotted results can be smoothed or unsmoothed.

Usage

```
## S3 method for class 'bayes_ctd_array'
plot(
  x = NULL,
  measure = "power",
  tab_type = "WX|YZ",
  smooth = FALSE,
  plot_out = TRUE,
  subj_per_arm_val = NULL,
  a0_val = NULL,
  effect_val = NULL,
  rand_control_diff_val = NULL,
  span = 0.75,
  degree = 2,
  family = "gaussian",
```

```
  title = NULL,
  ylim = NULL,
  ...
)
```

Arguments

x	Name of object of class bayes_ctd_array containing data from clinical trial simulation.
measure	Must be equal to power, est, var, bias, or mse. Default is power. Case does not matter.
tab_type	A character string that must equal WX|YZ, WY|XZ, WZ|XY, XY|WZ, XZ|WY, YZ|WX, ZX|WY, XW|YZ, YW|XZ, YX|WZ, ZW|XY, ZX|WY, ZY|WX when x is generated by historic_sim(). Default is WX|YZ. When x is generated by simple_sim(), tab_type is ignored.
smooth	A true/false parameter indicating whether smoothed results should be plotted. Note, smoothing of simulation results requires the length of subj_per_arm_val or a0_val or effect_val or rand_control_diff_val, whichever populates the x-axis on the graph to contain enough elements to justify the smoothing. No checking occurs to determine if enough elements are present to justify smoothing. Default is FALSE.
plot_out	A true/false parameter indicating whether the plot should be produced. This parameter is useful if the user only wants a table of smoothed values. Default is TRUE.
subj_per_arm_val	Must be non-missing, if x is generated by historic_sim() and sample size is being held constant. If x is generated by historic_sim() and sample size is being held constant, subj_per_arm_val must equal a value submitted to historic_sim() within the subj_per_arm parameter. When x is generated by simple_sim(), subj_per_arm_val is ignored.
a0_val	Must be non-missing, if x is generated by historic_sim() and a0, the power prior parameter, is being held constant. If x is generated by historic_sim() and a0 is being held constant, a0_val must equal a value submitted to historic_sim() within the a0_val parameter. When x is generated by simple_sim(), a0_val is ignored.
effect_val	Must be non-missing, if x is generated by historic_sim() and effect is being held constant. If x is generated by historic_sim() and effect is being held constant, effect_val must equal a value submitted to historic_sim() within the effect_vals parameter. When x is generated by simple_sim(), effect_val is ignored.
rand_control_diff_val	Must be non-missing, if x is generated by historic_sim() and differences between randomized and historical controls are being held constant. If x is generated by historic_sim() and control differences are being held constant, rand_control_diff_val must equal a value submitted to historic_sim() within the rand_control_diff parameter. When x is generated by simple_sim(), rand_control_diff_val is ignored.

span	The span parameter value for a call loess(). Default is 0.75. If span is a single number, then that value will be used to smooth the data in all columns of the table being plotted. If span is a vector, then it must have length equal to the number of columns being plotted.
degree	The degree parameter value for a call loess(). Default is 2. The value of degree will be used for all columns being plotted.
family	The family parameter value for a call loess(). Default is "gaussian". The value of family will be used for all columns being plotted.
title	Title for the plot.
ylim	Lower and upper limits for y-axis of plot.
...	further arguments passed to or from other methods.

Details

If the object of class bayes_ctd_array is created by historic_sim(), the function plot() allows the user to create line plots of user-specified 1- or 2- dimensional slices of the simulation results based on slicing code described below. If the object of class bayes_ctd_array is created by simple_sim(), a basic plot of characteristic by sample size and effect is created.

If the object of class bayes_ctd_array is created by simple_sim(), then all four trial characteristics (subj_per_arm_val, a0_vals, effect_val, and rand_control_diff_val) can be ignored as can the parameter defining what type of plot to create through the parameter tab_type. A call to plot() will require the user to specify a measure (power, est, var, bias, or mse).

If the object of class bayes_ctd_array is created by historic_sim(), when calling plot() the user must specify a measure to plot (power, est, var, bias, or mse) and may be required to specify a plot type through the tab_type parameter. A plot type, tab_type, will be required if 3 of the 4 trial characteristics are equal to a vector of 2 or more values. This plot type specification uses the letters W, X, Y, and Z. The letter W represents the subject per arm dimension. The letter X represents the a0 dimension. The letter Y represents the effect dimension. The letter Z represents the control difference dimension. To plot a slice of the 4-dimensional array, these letters are put into an AB|CD pattern just like in print(). The two letters to the right of the vertical bar define which variables are held constant. The two letters to the left of the vertical bar define which variables are going to show up in the plot. The first letter defines the x-axis variable and the second letter defines the stratification variable. The result is a plot of power, estimate, variance, bias, or mse by the trial characteristic represented by the first letter. On this plot, one line will be created for each value of the trial characteristic represented by the second letter. For example if tab_type equals WX|YZ, then effect and control differences will be held constant, while sample size will be represented along the horizontal axis and a0 values will be represented by separate lines. The actual values that are plotted on the y-axis depend on what measure is requested in the parameter measure.

- tab_type='WX|YZ', Sample Size by a0
- tab_type='WY|XZ', Sample Size by Effect
- tab_type='WZ|XY', Sample Size by Control Differences
- tab_type='XY|WZ', a0 by Effect
- tab_type='XZ|WY', a0 by Control Differences
- tab_type='YZ|WX', Effect by Control Differences

- tab_type='ZX|WY', Control Differences by a0
- tab_type='XW|YZ', a0 by Sample Size
- tab_type='YW|XZ', Effect by Sample Size
- tab_type='YX|WZ', Effect by a0
- tab_type='ZW|XY', Control Differences by Sample Size
- tab_type='ZY|WX', Control Differences by Effect

It is very important to populate the values of subj_per_arm_val, a0_val, effect_val, and rand_control_diff_val correctly given the value of tab_type, when the object of class bayes_ctd_array is created by historic_sim() and at least 3 of the four parameters have more than one value. On, the other hand, if 2 or more of the four parameters have only one value, then subj_per_arm_val, a0_vals, effect_val, rand_control_diff_val, as well as tab_type can be ignored. If the last two letters are YZ, then effect_val and rand_control_diff_val must be populated. If the last two letters are XZ, then a0_val and rand_control_diff_val must be populated. If the last two letters are XY, then a0_val and effect_val must be populated. If the last two letters are WZ, then sample_val and rand_control_diff_val must be populated. If the last two letters are WY, then sample_size_val and effect_val must be populated. If the last two letters are WX, then sample_size_val and a0_val must be populated.

If the object of class bayes_ctd_array is created by simple_sim(), the parameters tab_type, subj_per_arm_val, a0_val, effect_val, and rand_control_diff_val are ignored.

Value

plot() returns a plot for a two dimensional array of simulation results. If smooth is TRUE, then the plot is based on a smoothed version of the simulation results. If smooth is FALSE, then the plot is based on the raw data from the simulation results. What actually is plotted depends on the value of measure. If plot_out is FALSE, the plot is not created. This option is useful when the user wants a table of smoothed simulation results but does not want the plot. Smoothing of simulation results requires the length of subj_per_arm_val or a0_val or effect_val or rand_control_diff_val, whichever populates the x-axis on the graph to contain enough elements to justify the smoothing. No checking occurs to determine if enough elements are present to justify smoothing.

Examples

```
#Run a Weibull simulation, using simple_sim().
#For meaningful results, trial_reps needs to be much larger than 2.
weibull_test <- simple_sim(trial_reps = 2, outcome_type = "weibull",
                           subj_per_arm = c(50, 100, 150, 200),
                           effect_vals = c(0.6, 1),
                           control_parms = c(2.82487,3), time_vec = NULL,
                           censor_value = NULL, alpha = 0.05,
                           get_var = TRUE, get_bias = TRUE, get_mse = TRUE,
                           seedval=123, quietly=TRUE)

#Create a plot of the power simulation results.
plot(x=weibull_test, measure="power", tab_type=NULL,
     smooth=FALSE, plot_out=TRUE, subj_per_arm_val=NULL, a0_val=NULL,
     effect_val=NULL, rand_control_diff_val=NULL)
#Create a plot of the hazard ratio simulation results.
```

```
plot(x=weibull_test, measure="est", tab_type=NULL,
     smooth=FALSE, plot_out=TRUE, subj_per_arm_val=NULL, a0_val=NULL,
     effect_val=NULL, rand_control_diff_val=NULL)
#Create a plot of the hazard ratio variance simulation results.
plot(x=weibull_test, measure="var", tab_type=NULL,
     smooth=FALSE, plot_out=TRUE, subj_per_arm_val=NULL, a0_val=NULL,
     effect_val=NULL, rand_control_diff_val=NULL)
#Create a plot of the hazard ratio bias simulation results.
plot(x=weibull_test, measure="bias", tab_type=NULL,
     smooth=FALSE, plot_out=TRUE, subj_per_arm_val=NULL, a0_val=NULL,
     effect_val=NULL, rand_control_diff_val=NULL)
#Create a plot of the hazard ratio mse simulation results.
plot(x=weibull_test, measure="mse", tab_type=NULL,
     smooth=FALSE, plot_out=TRUE, subj_per_arm_val=NULL, a0_val=NULL,
     effect_val=NULL, rand_control_diff_val=NULL)


#Run a second Weibull simulation, using simple_sim() and smooth the plot.
#For meaningful results, trial_reps needs to be larger than 100.
weibull_test2 <- simple_sim(trial_reps = 100, outcome_type = "weibull",
                            subj_per_arm = c(50, 75, 100, 125, 150, 175, 200, 225, 250),
                            effect_vals = c(0.6, 1, 1.4),
                            control_parms = c(2.82487,3), time_vec = NULL,
                            censor_value = NULL, alpha = 0.05, get_var = TRUE,
                            get_bias = TRUE, get_mse = TRUE, seedval=123,
                            quietly=TRUE)

#Create a plot of the power simulation results.
plot(x=weibull_test2, measure="power", tab_type=NULL,
     smooth=TRUE, plot_out=TRUE, subj_per_arm_val=NULL, a0_val=NULL,
     effect_val=NULL, rand_control_diff_val=NULL, span=c(1,1,1))



#Run a third weibull simulation, using historic_sim().
#Note: historic_sim() can take a while to run.
#Generate a sample of historical data for use in example.
set.seed(2250)
SampleHistData <- genweibulldata(sample_size=60, scale1=2.82487,
                                 hazard_ratio=0.6, common_shape=3,
                                 censor_value=3)
histdata <- subset(SampleHistData, subset=(treatment==0))
histdata$id <- histdata$id+10000

#For meaningful results, trial_reps needs to be larger than 100.
weibull_test3 <- historic_sim(trial_reps = 100, outcome_type = "weibull",
                              subj_per_arm = c(50, 100, 150, 200, 250),
                              a0_vals = c(0, 0.33, 0.67, 1),
                              effect_vals = c(0.6, 1, 1.4),
                              rand_control_diff = c(0.8, 1, 1.2),
                              hist_control_data = histdata, time_vec = NULL,
                              censor_value = 3, alpha = 0.05, get_var = TRUE,
                              get_bias = TRUE, get_mse = TRUE, seedval=123,
```

```
                                quietly=TRUE)

#Create a plot of the power simulation results.
plot(x=weibull_test3, measure="power", tab_type="WX|YZ",
     smooth=FALSE, plot_out=TRUE, effect_val=0.6,
     rand_control_diff_val=1.0)



#Run a Gaussian simulation, using historic_sim()
#Generate a sample of historical Gaussian data for use in example.
set.seed(2250)
samplehistdata <- gengaussiandata(sample_size=60, mu1=25, mean_diff=0, common_sd=3)
histdata <- subset(samplehistdata, subset=(treatment==0))
histdata$id <- histdata$id+10000

#For meaningful results, trial_reps needs to be larger than 100.
gaussian_test <- historic_sim(trial_reps = 100, outcome_type = "gaussian",
                              subj_per_arm = c(150),
                              a0_vals = c(1.0),
                              effect_vals = c(0.15),
                              rand_control_diff = c(-4.0,-3.5,-3.0,-2.5,-2.0,
                                                    -1.5,-1.0,-0.5,0,0.5,1.0),
                              hist_control_data = histdata, time_vec = NULL,
                              censor_value = 3, alpha = 0.05, get_var = TRUE,
                              get_bias = TRUE, get_mse = TRUE, seedval=123,
                              quietly=TRUE)
test_table <- print(x=gaussian_test, measure="power",
                          tab_type=NULL, effect_val=NULL,
                          subj_per_arm_val=NULL)
print(test_table)
#Create a plot of the power simulation results.
plot(x=gaussian_test, measure="power", tab_type=NULL,
     smooth=TRUE, plot_out=TRUE, effect_val=NULL,
     rand_control_diff_val=NULL)



#Generate a sample of historical pwe data for use in example.
set.seed(2250)
nvalHC <- 60
time.vec <- c(0.3,0.9,1.5,2.1,2.4)
lambdaHC.vec <- c(0.19,0.35,0.56,0.47,0.38,0.34)
censor.value <- 3

SampleHistData <- genpwedata(nvalHC, lambdaHC.vec, 1.0, time.vec, censor.value)
histdata <- subset(SampleHistData, subset=(treatment==0))
histdata$indicator <- 2 #If set to 2, then historical controls will be collapsed with
#randomized controls, when time_vec is re-considered and
#potentially restructured.  If set to 1, then historical
#controls will be treated as a separated cohort when
#time_vec is being assessed for restructuring.
histdata$id <- histdata$id+10000
```

```
#Run a pwe simulation, using historic_sim().
#For meaningful results, trial_reps needs to be larger than 100.
pwe_test <- historic_sim(trial_reps = 100, outcome_type = "pwe",
                         subj_per_arm = c(25,50,75,100,125,150,175,200,225,250),
                         a0_vals = c(1.0),
                         effect_vals = c(0.6),
                         rand_control_diff = c(1.8),
                         hist_control_data = histdata, time_vec = time.vec,
                         censor_value = 3, alpha = 0.05, get_var = TRUE,
                         get_bias = TRUE, get_mse = TRUE, seedval=123,
                         quietly=TRUE)

#Create a plot of the power simulation results.
plot(x=pwe_test, measure="power", tab_type=NULL,
     smooth=TRUE, plot_out=TRUE, effect_val=NULL,
     rand_control_diff_val=NULL)
```

`print.bayes_ctd_array` *Print Data from Two Arm Bayesian Clinical Trial Simulation.*

Description

`print.bayes_ctd_array()` takes an S3 object of class `bayes_ctd_array`, and prints a two dimensional slice from the data generated by a clinical trial simulation using `historic_sim()` or `simple_sim()`.

Usage

```
## S3 method for class 'bayes_ctd_array'
print(
  x = NULL,
  measure = "power",
  tab_type = "WX|YZ",
  subj_per_arm_val = NULL,
  a0_val = NULL,
  effect_val = NULL,
  rand_control_diff_val = NULL,
  print_chg_warn = 1,
  ...
)
```

Arguments

`x`	Name of object of class `bayes_ctd_array` containing data from clinical trial simulation.

measure — Must be equal to power, est, var, bias, or mse. Default is power. Case does not matter.

tab_type — A character string that must equal WX|YZ, WY|XZ, WZ|XY, XY|WZ, XZ|WY, YZ|WX, ZX|WY, XW|YZ, YW|XZ, YX|WZ, ZW|XY, ZX|WY, ZY|WX when x is generated by historic_sim(). Default is WX|YZ. When x is generated by simple_sim(), tab_type is ignored.

subj_per_arm_val — Must be non-missing, if x is generated by historic_sim() and sample size is being held constant. If x is generated by historic_sim() and sample size is being held constant, subj_per_arm_val must equal a value submitted to historic_sim() within the subj_per_arm parameter. When x is generated by simple_sim(), subj_per_arm_val is ignored.

a0_val — Must be non-missing, if x is generated by historic_sim() and a0, the power prior parameter, is being held constant. If x is generated by historic_sim() and a0 is being held constant, a0_val must equal a value submitted to historic_sim() within the a0_vals parameter. When x is generated by simple_sim(), a0_val is ignored.

effect_val — Must be non-missing, if x is generated by historic_sim() and effect is being held constant. If x is generated by historic_sim() and effect is being held constant, effect_val must equal a value submitted to historic_sim() within the effect_vals parameter. When x is generated by simple_sim(), effect_val is ignored.

rand_control_diff_val — Must be non-missing, if x is generated by historic_sim() and differences between randomized and historical controls are being held constant. If x is generated by historic_sim() and control differences are being held constant, rand_control_diff_val must equal a value submitted to historic_sim() within the rand_control_diff parameter. When x is generated by simple_sim(), rand_control_diff_val is ignored.

print_chg_warn — A parameter not used by the user, but is used by plot() to ensure warnings are not printed twice.

... — further arguments passed to or from other methods.

Details

If the object of class bayes_ctd_array is created by historic_sim(), then the function print() allows the user to print user-specified 1- and 2- dimensional slices of the simulation results based on slicing code described below. If the object of class bayes_ctd_array is created by simple_sim(), a basic table of characteristic by sample size and effect is created.

If the object of class bayes_ctd_array is created by simple_sim(), then all four trial characteristics (subj_per_arm_val, a0_vals, effect_val, and rand_control_diff_val) can be ignored, as can the parameter defining what type of table to print, tab_type. A call to print() will require the user to specify a measure (power, est, var, bias, or mse).

If the object of class bayes_ctd_array is created by historic_sim(), a call to print() will require the user to specify a measure (power, est, var, bias, or mse) and may require the user to specify a table type. A table type, tab_type, will be required if 3 of the 4 trial characteristics are

equal to a vector of 2 or more values. The table type specification uses the letters W, X, Y, and Z. The letter W represents the subject per arm dimension. The letter X represents the a0 dimension. The letter Y represents the effect dimension. The letter Z represents the control difference dimension. To define a slice of the 4-dimensional array, these letters are put into an AB|CD pattern. The two letters to the right of the vertical bar define which variables are held constant. The two letters to the left of the vertical bar define which variables are going to show up in the rows (first letter) and in the columns (second letter). For example if tab_type equals `WX|YZ`, then effect and control differences will be held constant, while sample size will be represented by the rows in the generated table and a0 values will be represented by the columns. The actual values that are printed in the tables depend on what measure is requested in the parameter `measure`.

- `tab_type='WX|YZ'`, Sample Size by a0
- `tab_type='WY|XZ'`, Sample Size by Effect
- `tab_type='WZ|XY'`, Sample Size by Control Differences
- `tab_type='XY|WZ'`, a0 by Effect
- `tab_type='XZ|WY'`, a0 by Control Differences
- `tab_type='YZ|WX'`, Effect by Control Differences
- `tab_type='ZX|WY'`, Control Differences by a0
- `tab_type='XW|YZ'`, a0 by Sample Size
- `tab_type='YW|XZ'`, Effect by Sample Size
- `tab_type='YX|WZ'`, Effect by a0
- `tab_type='ZW|XY'`, Control Differences by Sample Size
- `tab_type='ZY|WX'`, Control Differences by Effect

It is very important to populate the values of `subj_per_arm_val`, `a0_vals`, `effect_val`, and `rand_control_diff_val` correctly given the value of tab_type, when the object of class `bayes_ctd_array` is created by `historic_sim()` and at least 3 of the four parameters have more than one value. On the other hand, if 2 or more of the four parameters have only one value, then `subj_per_arm_val`, `a0_vals`, `effect_val`, `rand_control_diff_val`, as well as `tab_type` can be ignored. If the last two letters are `YZ`, then `effect_val` and `rand_control_diff_val` must be populated. If the last two letters are `XZ`, then `a0_vals` and `rand_control_diff_val` must be populated. If the last two letters are `XY`, then `a0_vals` and `effect_val` must be populated. If the last two letters are `WZ`, then `sample_val` and `rand_control_diff_val` must be populated. If the last two letters are `WY`, then `sample_size_val` and `effect_val` must be populated. If the last two letters are `WX`, then `sample_size_val` and `a0_vals` must be populated.

If the object of class `bayes_ctd_array` is created by `simple_sim()`, the parameters `tab_type`, `subj_per_arm_val`, `a0_vals`, `effect_val`, and `rand_control_diff_val` are ignored.

Value

`print()` returns a two dimensional array of simulation results.

Examples

```
#Run a Weibull simulation, using simple_sim().
#For meaningful results, trial_reps needs to be much larger than 2.
weibull_test <- simple_sim(trial_reps = 2, outcome_type = "weibull",
                           subj_per_arm = c(50, 100, 150, 200),
                           effect_vals = c(0.6, 1, 1.4),
                           control_parms = c(2.82487,3),
                           time_vec = NULL, censor_value = NULL,
                           alpha = 0.05, get_var = TRUE,
                           get_bias = TRUE, get_mse = TRUE,
                           seedval=123, quietly=TRUE)

#Tabulate the simulation results for power.
test_table <- print(x=weibull_test, measure="power",
                    tab_type=NULL, subj_per_arm_val=NULL, a0_val=NULL,
                    effect_val=NULL, rand_control_diff_val=NULL)
print(test_table)

#Tabulate the simulation results for estimates.
print(x=weibull_test, measure="est")

#Tabulate the simulation results for variance.
print(x=weibull_test, measure="var")

#Tabulate the simulation results for bias.
print(x=weibull_test, measure="bias")

#Tabulate the simulation results for mse.
print(x=weibull_test, measure="mse")


#Run another weibull simulation, using historic_sim().
#Note: historic_sim() can take a while to run.
#Generate a sample of historical data for use in example.
set.seed(2250)
SampleHistData <- genweibulldata(sample_size=60, scale1=2.82487,
                                 hazard_ratio=0.6, common_shape=3,
                                 censor_value=3)
histdata <- subset(SampleHistData, subset=(treatment==0))
histdata$id <- histdata$id+10000

#For meaningful results, trial_reps needs to be larger than 100.
weibull_test2 <- historic_sim(trial_reps = 100, outcome_type = "weibull",
                              subj_per_arm = c(50, 100, 150, 200, 250),
                              a0_vals = c(0, 0.33, 0.67, 1),
                              effect_vals = c(0.6, 1, 1.4),
                              rand_control_diff = c(0.8, 1, 1.2),
                              hist_control_data = histdata, time_vec = NULL,
                              censor_value = 3, alpha = 0.05, get_var = TRUE,
                              get_bias = TRUE, get_mse = TRUE, seedval=123,
                              quietly=TRUE)
```

```
#Tabulate the simulation results for power.
test_table <- print(x=weibull_test2, measure="power",
                    tab_type="WX|YZ", effect_val=0.6,
                    rand_control_diff_val=1.0)
print(test_table)

#Tabulate the simulation results for estimates.
print(x=weibull_test2, measure="est", tab_type="WX|YZ",
      effect_val=0.6, rand_control_diff_val=1.0)

#Tabulate the simulation results for variance.
print(x=weibull_test2, measure="var", tab_type="WX|YZ",
      effect_val=0.6, rand_control_diff_val=1.0)

#Tabulate the simulation results for bias.
print(x=weibull_test2, measure="bias", tab_type="WX|YZ",
      effect_val=0.6, rand_control_diff_val=1.0)

#Tabulate the simulation results for mse.
print(x=weibull_test2, measure="mse", tab_type="WX|YZ",
      effect_val=0.6, rand_control_diff_val=1.0)



#Run a Bernoulli simulation, using historic_sim().
#Generate a sample of historical Bernoulli data for use in example.
set.seed(2250)
samplehistdata <- genbernoullidata(sample_size=60, prob1=0.6, odds_ratio=0.6)
histdata <- subset(samplehistdata, subset=(treatment==0))
histdata$id <- histdata$id+10000

#For meaningful results, trial_reps needs to be larger than 100.
bernoulli_test <- historic_sim(trial_reps = 100, outcome_type = "bernoulli",
                               subj_per_arm = c(150),
                               a0_vals = c(1.0),
                               effect_vals = c(0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0),
                               rand_control_diff = c(1.8),
                               hist_control_data = histdata, time_vec = NULL,
                               censor_value = 3, alpha = 0.05, get_var = TRUE,
                               get_bias = TRUE, get_mse = TRUE, seedval=123,
                               quietly=TRUE)
test_table <- print(x=bernoulli_test, measure="power",
                    tab_type=NULL, effect_val=NULL,
                    subj_per_arm_val=NULL)
print(test_table)

#If only one or two of the subj_per_arm, a0_vals, effect_vals, or
#rand_control_diff parameters have length greater than 1, then
#only bayes_ctd_array and measure parameters are needed.
#Tabulate the simulation results for estimates.
print(x=bernoulli_test, measure="est")

#Tabulate the simulation results for variance.
```

```
print(x=bernoulli_test, measure="var")

#Tabulate the simulation results for bias.
print(x=bernoulli_test, measure="bias")

#Tabulate the simulation results for mse.
print(x=bernoulli_test, measure="mse")
```

`simple_sim` *Two Arm Bayesian Clinical Trial Simulation without Historical Data*

Description

`simple_sim()` returns an S3 object of class `bayes_ctd_array`, which will contain simulation results for power, statistic estimation, bias, variance, and mse as requested by user.

Usage

```
simple_sim(
  trial_reps = 100,
  outcome_type = "weibull",
  subj_per_arm = c(50, 100, 150, 200, 250),
  effect_vals = c(0.6, 1, 1.4),
  control_parms = NULL,
  time_vec = NULL,
  censor_value = NULL,
  alpha = 0.05,
  get_var = FALSE,
  get_bias = FALSE,
  get_mse = FALSE,
  seedval = NULL,
  quietly = TRUE
)
```

Arguments

`trial_reps`	Number of trials to replicate within each combination of `a0_vals`, `subj_per_arm`, `effect_vals`, and `rand_control_parms`. As the number of trials increases, the precision of the estimate will increase. Default is 100.
`outcome_type`	Outcome distribution. Must be equal to `weibull`, `lognormal`, `pwe` (Piecewise Exponential), `gaussian`, `bernoulli`, or `poisson`. Default is `weibull`.
`subj_per_arm`	A vector of sample sizes, all of which must be positive integers. Default is `c(50, 100, 150, 200, 250)`.

effect_vals — A vector of effects that should be reasonable for the outcome_type being studied, hazard ratios for Weibull, odds ratios for Bernoulli, mean ratios for Poisson, etc.. When effect_vals contain the null effect for a given outcome_type, the power component of data will contain an estimate of Type One Error. In order to have a good set of Type One Error estimates, trial_reps need to be at least 10,000. In such a case, if the total number of combinations made up from subj_per_arm, a0_vals, effect_vals, and rand_control_diff is very large, the time to complete the simulation can be substantial. Default is c(0.6, 1, 1.4).

control_parms — A vector of parameter values defining the outcome distribution for randomized controls. See Details for what is required for each outcome_type.

time_vec — A vector of time values that are used to create time periods within which the exponential hazard is constant. Only used for piecewise exponential models. Default is NULL.

censor_value — A single value at which right censoring occurs when simulating randomized subject outcomes. Used with survival outcomes. Default is NULL, where NULL implies no right censoring.

alpha — A number ranging between 0 and 1 that defines the acceptable Type 1 error rate. Default is 0.05.

get_var — A TRUE/FALSE indicator of whether an array of variance estimates will be returned. Default is FALSE.

get_bias — A TRUE/FALSE indicator of whether an array of bias estimates will be returned. Default is FALSE.

get_mse — A TRUE/FALSE indicator of whether an array of MSE estimates will be returned. Default is FALSE.

seedval — A seed value for pseudo-random number generation.

quietly — A TRUE/FALSE indicator of whether notes are printed to output about simulation progress as the simulation runs. If running interactively in RStudio or running in the R console, quietly can be set to FALSE. If running in a Notebook or knitr document, quietly needs to be set to TRUE. Otherwise each note will be printed on a separate line and it will take up a lot of output space. Default is TRUE.

Details

The object bayes_ctd_array has 6 elements: a list containing simulation results (data), copies of the 4 function arguments subj_per_arm, a0_vals, effect_vals, and rand_control_diff, and finally a objtype value indicating that simple_sim() was used. Each element of data is a four-dimensional array, where each dimension is determined by the length of parameters subj_per_arm, a0_vals, effect_vals, and rand_control_diff. The size of data depends on which results are requested by the user. At a minimum, at least one of subj_per_arm, a0_vals, effect_vals, or rand_control_diff must contain at least 2 values, while the other three must contain at least 1 value. The data list will always contain two elements: an array of power results (power) and an array of estimation results (est). In addition to power and est, data may also contain elements var, bias, or mse, depending on the values of get_var, get_bias, and get_mse. The values returned in est are in the form of hazard ratios, mean ratios, odds ratios, or mean differences depending on the

value of outcome_type. For a Gaussian outcome, the estimation results are differences in group means (experimental group minus control group). For a logistic outcome, the estimation results are odds ratios (experimental group over control group). For lognormal and Poisson outcomes, the estimation results are mean ratios (experimental group over control group). For a piecewise exponential or a Weibull outcome, the estimation results are hazard ratios (experimental group over control group). The values returned in bias, var, and mse are on the scale of the values returned in est.

The object bayes_ctd_array has two primary methods, print() and plot(), for printing and plotting slices of the arrays contained in bayes_ctd_array$data.

As dimensions of the four dimensional array increases, the time required to complete the simulation will increase; however, it will be faster than a similar simulation based on repeated calls to MCMC routines to analyze each simulated trial.

The meaning of the estimation results, and the test used to generate power results, depends on the outcome used. In all cases, power is based on a two-sided test involving a (1-alpha)100% credible interval, where the interval is used to determine if the null hypothesis should be rejected (null value outside of the interval) or not rejected (null value inside the interval). For a Gaussian outcome, the 95% credible interval is an interval for the difference in group means (experimental group minus control group), and the test determines if 0 is in or outside of the interval. For a Bernoulli outcome, the 95% credible interval is an interval for the odds ratio (experimental group over control group), and the test determines if 1 is in or outside of the interval. For a lognormal or a Poisson outcome, the 95% credible interval is an interval for the mean ratio (experimental group over control group), and the test determines if 1 is in or outside of the interval. Finally, for a piecewise exponential or a Weibull outcome, the 95% credible interval is an interval for the hazard ratio (experimental group over control group), and the test determines if 1 is in or outside of the interval.

For a Gaussian outcome, the control_parms values should be (mean, sd), where mean is the mean parameter for the control group used in a call to rnorm(), and sd is the common sd parameter for both groups used in a call torlnorm().

For a Bernoulli outcome, the control_parms values should be (prob), where prob is the event probability for the control group used in a call to rbinom().

For a lognormal outcome, the control_parms values should be (meanlog, sdlog), where meanlog is the meanlog parameter for the control group used in a call to rlnorm(), and sdlog is the common sdlog parameter for both groups used in a call to rlnorm().

For a Poisson outcome, the control_parms value should be (lambda), where lambda is the lambda parameter for the control group used in a call to rpois() and is equal to the mean of a Poisson distribution.

For a Weibull outcome, the control_parms values should be (scale, shape), where scale is the scale parameter for the control group used in a call to rweibull(), and shape is the common shape parameter for both groups used in a call to rweibull().

For a piecewise exponential outcome, the control_parms values should be a vector of lambdas used in a call to eha::rpch(). Each element in control_parms is a hazard for an interval defined by the time_vec parameter.

Please refer to the examples for illustration of package use.

Value

simple_sim() returns an S3 object of class bayes_ctd_array. As noted in Details, an object of

class bayes_ctd_array has 6 elements: a list containing simulation results (data), copies of the 4 function arguments subj_per_arm, a0_vals, effect_vals, and rand_control_diff, and finally objtype indicating that simple_sim() was used. See Details for a discussion about the contents of data. Results from the simulation contained in the bayes_ctd_array object can be printed or plotted using the print() and plot() methods. The results can also be accessed using basic list element identification and array slicing. For example, to get the power results from a simulation, one could use the code bayes_ctd_array$data$power, where bayes_ctd_array is replaced with the name of the variable containing the bayes_ctd_array object. Even though this is a 4-dimensional array, the power results only occupy a single 2-dimensional table. To print this 2-dimensional table, one would use the code bayes_ctd_array$data$power[,1,,1], where bayes_ctd_array is replaced with the name of the variable containing the bayes_ctd_array object.

Examples

```
#Run a Weibull simulation, using simple_sim().
#For meaningful results, trial_reps needs to be much larger than 2.
weibull_test <- simple_sim(trial_reps = 2, outcome_type = "weibull",
                           subj_per_arm = c(50, 100, 150, 200),
                           effect_vals = c(0.6, 1, 1.4),
                           control_parms = c(2.82487,3), time_vec = NULL,
                           censor_value = NULL, alpha = 0.05,
                           get_var = TRUE, get_bias = TRUE, get_mse = TRUE,
                           seedval=123, quietly=TRUE)

#Tabulate the simulation results for power.
test_table <- print(x=weibull_test, measure="power",
                    tab_type=NULL, subj_per_arm_val=NULL, a0_val=NULL,
                    effect_val=NULL, rand_control_diff_val=NULL)
print(test_table)

#Create a plot of the power simulation results.
plot(x=weibull_test, measure="power", tab_type=NULL,
     smooth=FALSE, plot_out=TRUE)
#Create a plot of the estimated hazard ratio simulation results.
plot(x=weibull_test, measure="est", tab_type=NULL,
     smooth=FALSE, plot_out=TRUE)
#Create a plot of the hazard ratio variance simulation results.
plot(x=weibull_test, measure="var", tab_type=NULL,
     smooth=FALSE, plot_out=TRUE)
#Create a plot of the hazard ratio bias simulation results.
plot(x=weibull_test, measure="bias", tab_type=NULL,
     smooth=FALSE, plot_out=TRUE)
#Create a plot of the hazard ratio mse simulation results.
plot(x=weibull_test, measure="mse", tab_type=NULL,
     smooth=FALSE, plot_out=TRUE)
```

Index