

Package ‘EvidenceSynthesis’

July 17, 2025

Type Package

Title Synthesizing Causal Evidence in a Distributed Research Network

Version 0.6.1

Date 2025-07-17

Maintainer Martijn Schuemie <schuemie@ohdsi.org>

Description Routines for combining causal effect estimates and study diagnostics across multiple data sites in a distributed study, without sharing patient-level data.
Allows for normal and non-normal approximations of the data-site likelihood of the effect parameter.

SystemRequirements Java (≥ 8)

Depends survival,
R ($\geq 3.5.0$)

Imports dplyr,
ggplot2,
ggdist,
gridExtra,
meta,
EmpiricalCalibration,
rJava,
BeastJar,
Cyclops ($\geq 3.1.0$),
HDInterval,
coda,
rlang,
methods

Suggests knitr,
rmarkdown,
testthat,
sn,
tidyr

Remotes ohdsi/Cyclops@jni

License Apache License 2.0

URL <https://ohdsi.github.io/EvidenceSynthesis/>, <https://github.com/OHDSI/EvidenceSynthesis>

BugReports <https://github.com/OHDSI/EvidenceSynthesis/issues>

VignetteBuilder knitr

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

Encoding UTF-8

LazyData true

Contents

approximateHierarchicalNormalPosterior	3
approximateLikelihood	4
approximateSimplePosterior	5
biasCorrectionInference	6
buildLabelReferences	8
computeBayesianMetaAnalysis	8
computeConfidenceInterval	10
computeFixedEffectMetaAnalysis	10
computeHierarchicalMetaAnalysis	11
constructDataModel	12
createApproximations	13
createSccsSimulationSettings	13
createSimulationSettings	15
customFunction	16
detectApproximationType	17
extractSourceSpecificEffects	17
fitBiasDistribution	18
generateBayesianHMAsettings	19
hermiteInterpolation	20
hmaLikelihoodList	21
likelihoodProfileLists	22
ncLikelihoods	22
ooiLikelihoods	23
plotBiasCorrectionInference	24
plotBiasDistribution	25
plotCovariateBalances	26
plotEmpiricalNulls	27
plotLikelihoodFit	28
plotMcmcTrace	29
plotMetaAnalysisForest	30
plotPerDbMcmcTrace	32
plotPerDbPosterior	33
plotPosterior	34
plotPreparedPs	35
preparePsPlot	36
prepareSccsIntervalData	37
sequentialFitBiasDistribution	38
simulateMetaAnalysisWithNegativeControls	39
simulatePopulations	40
skewNormal	40
summarizeChain	41
supportsJava8	42

Index**43**

approximateHierarchicalNormalPosterior

*Approximate Bayesian posterior for hierarchical Normal model***Description**

Approximate a Bayesian posterior from a set ofCyclops likelihood profiles under a hierarchical normal model using the Markov chain Monte Carlo engine BEAST.

Usage

```
approximateHierarchicalNormalPosterior(
  likelihoodProfiles,
  chainLength = 1100000,
  burnIn = 1e+05,
  subSampleFrequency = 100,
  effectPriorMean = 0,
  effectPriorSd = 0.5,
  nu0 = 1,
  sigma0 = 1,
  effectStartingValue = 0,
  precisionStartingValue = 1,
  seed = 1
)
```

Arguments

likelihoodProfiles	List of grid likelihoods profiled with Cyclops.
chainLength	Number of MCMC iterations.
burnIn	Number of MCMC iterations to consider as burn in.
subSampleFrequency	Subsample frequency for the MCMC.
effectPriorMean	Prior mean for global parameter
effectPriorSd	Prior standard deviation for the global parameter
nu0	Prior "sample size" for precision (with precision $\sim \text{gamma}(\text{nu0}/2, \text{nu0}*\text{sigma0}/2)$)
sigma0	Prior "variance" for precision (with precision $\sim \text{gamma}(\text{nu0}/2, \text{nu0}*\text{sigma0}/2)$)
effectStartingValue	Initial value for global & local parameter
precisionStartingValue	Initial value for the precision
seed	Seed for the random number generator.

Value

A data frame with the point estimates and 95% credible intervals for the the global and local parameter, as well as the global precision. Attributes of the data frame contain the MCMC trace for diagnostics.

Examples

```
# TBD
```

```
approximateLikelihood Approximate a likelihood function
```

Description

Approximate the likelihood function using a parametric (normal, skew-normal, or custom parametric), or grid approximation. The approximation does not reveal person-level information, and can therefore be shared among data sites. When counts are low, a normal approximation might not be appropriate.

Usage

```
approximateLikelihood(
  cyclopsFit,
  parameter = 1,
  approximation = "custom",
  bounds = c(log(0.1), log(10))
)
```

Arguments

cyclopsFit	A model fitted using the Cyclops::fitCyclopsModel() function.
parameter	The parameter in the cyclopsFit object to profile.
approximation	The type of approximation. Valid options are 'normal', 'skew normal', 'custom', 'grid', 'adaptive grid', or 'grid with gradients'.
bounds	The bounds on the effect size used to fit the approximation.

Value

A vector of parameters of the likelihood approximation.

See Also

[computeConfidenceInterval](#), [computeFixedEffectMetaAnalysis](#), [computeBayesianMetaAnalysis](#)

Examples

```
# Simulate some data for this example:
populations <- simulatePopulations()

cyclopsData <- Cyclops::createCyclopsData(Surv(time, y) ~ x + strata(stratumId),
  data = populations[[1]],
  modelType = "cox"
)
cyclopsFit <- Cyclops::fitCyclopsModel(cyclopsData)
approximation <- approximateLikelihood(cyclopsFit, "x")
approximation
```

```
# (Estimates in this example will vary due to the random simulation)
```

```
approximateSimplePosterior
```

Approximate simple Bayesian posterior

Description

Approximate a Bayesian posterior from a Cyclops likelihood profile and normal prior using the Markov chain Monte Carlo engine BEAST.

Usage

```
approximateSimplePosterior(
  likelihoodProfile,
  chainLength = 1100000,
  burnIn = 1e+05,
  subSampleFrequency = 100,
  priorMean = 0,
  priorSd = 0.5,
  startingValue = 0,
  seed = 1
)
```

Arguments

likelihoodProfile	Named vector containing grid likelihood data from Cyclops.
chainLength	Number of MCMC iterations.
burnIn	Number of MCMC iterations to consider as burn in.
subSampleFrequency	Subsample frequency for the MCMC.
priorMean	Prior mean for the regression parameter
priorSd	Prior standard deviation for the regression parameter
startingValue	Initial state for regression parameter
seed	Seed for the random number generator.

Value

A data frame with the point estimates and 95% credible intervals for the regression parameter. Attributes of the data frame contain the MCMC trace for diagnostics.

Examples

```
# Simulate some data for this example:
population <- simulatePopulations(createSimulationSettings(nSites = 1))[[1]]

# Fit a Cox regression at each data site, and approximate likelihood function:
cyclopsData <- Cyclops::createCyclopsData(Surv(time, y) ~ x + strata(stratumId),
  data = population,
  modelType = "cox"
)
cyclopsFit <- Cyclops::fitCyclopsModel(cyclopsData)
likelihoodProfile <- approximateLikelihood(cyclopsFit, parameter = "x", approximation = "grid")

# Run MCMC
mcmcTraces <- approximateSimplePosterior(
  likelihoodProfile = likelihoodProfile,
  priorMean = 0, priorSd = 100
)

# Report posterior expectation
mean(mcmcTraces$theta)

# (Estimates in this example will vary due to the random simulation)
```

biasCorrectionInference

Bias Correction with Inference

Description

Perform Bayesian posterior inference regarding an outcome of interest with bias correction using negative control analysis. There is an option to not perform bias correction so that un-corrected results can be obtained.

Usage

```
biasCorrectionInference(
  likelihoodProfiles,
  ncLikelihoodProfiles = NULL,
  biasDistributions = NULL,
  priorMean = 0,
  priorSd = 1,
  numsamps = 10000,
  thin = 10,
  doCorrection = TRUE,
  seed = 1,
  ...
)
```

Arguments

likelihoodProfiles

A list of grid profile likelihoods for the outcome of interest.

ncLikelihoodProfiles	Likelihood profiles for the negative control outcomes. Must be a list of lists of profile likelihoods; if there is only one analysis period, then this must be a length-1 list, with the first item as a list all outcome-wise profile likelihoods.
biasDistributions	Pre-saved bias distribution(s), formatted as the output from fitBiasDistribution() or sequentialFitBiasDistribution() . If NULL, then ncLikelihoodProfiles must be provided.
priorMean	Prior mean for the effect size (log rate ratio).
priorSd	Prior standard deviation for the effect size (log rate ratio).
numsamps	Total number of MCMC samples needed.
thin	Thinning frequency: how many iterations before another sample is obtained?
doCorrection	Whether or not to perform bias correction; default: TRUE.
seed	Seed for the random number generator.
...	Arguments to be passed to sequentialFitBiasDistribution() to fit bias distributions if biasDistributions is NULL.

Value

A dataframe with five columns, including posterior median and mean of log RR effect size estimates, 95% credible intervals (ci95Lb and ci95Ub), posterior probability that log RR > 0 (p1), and the period or group ID (Id).

It is accompanied by the following attributes:

- `samplesCorrected`: all MCMC samples for the bias corrected log RR effect size estimate.
- `samplesRaw`: all MCMC samples for log RR effect size estimate, without bias correction.
- `biasDistributions`: the learned empirical bias distribution from negative control analysis.
- `summaryRaw`: a summary dataframe (same format as in the main result) without bias correction.
- `corrected`: a logical flag indicating if bias correction has been performed; = TRUE if `doCorrection` = TRUE.

See Also

[approximateSimplePosterior](#), [fitBiasDistribution](#)

Examples

```
# load example data
data("ncLikelihoods")
data("ooiLikelihoods")

# perform sequential analysis with bias correction, using the t model
# NOT RUN
# bbcResults = biasCorrectionInference(ooiLikelihoods,
#                                     ncLikelihoodProfiles = ncLikelihoods,
#                                     robust = TRUE,
#                                     seed = 42)

# check out analysis summary
# bbcResults
```

buildLabelReferences	<i>Build a list of references that map likelihood names to integer labels for later use</i>
----------------------	---

Description

Build a list of references that map likelihood names to integer labels for later use

Usage

```
buildLabelReferences(data)
```

Arguments

data	The likelihood data. Can be a single approximation, approximations from multiple sites, or (adaptive) grid profile likelihoods.
------	---

Examples

```
data("likelihoodProfileLists")
refLabs <- buildLabelReferences(likelihoodProfileLists)
```

computeBayesianMetaAnalysis	<i>Compute a Bayesian random-effects meta-analysis</i>
-----------------------------	--

Description

Compute a Bayesian meta-analysis using the Markov chain Monte Carlo (MCMC) engine BEAST. A normal and half-normal prior are used for the mu and tau parameters, respectively, with standard deviations as defined by the priorSd argument.

Usage

```
computeBayesianMetaAnalysis(
  data,
  chainLength = 1100000,
  burnIn = 1e+05,
  subSampleFrequency = 100,
  priorSd = c(2, 0.5),
  alpha = 0.05,
  robust = FALSE,
  df = 4,
  seed = 1,
  showProgressBar = TRUE
)
```


Arguments

data	A data frame containing either normal, skew-normal, custom parametric, or grid likelihood data, with one row per database.
chainLength	Number of MCMC iterations.
burnIn	Number of MCMC iterations to consider as burn in.
subSampleFrequency	Subsample frequency for the MCMC.
priorSd	A two-dimensional vector with the standard deviation of the prior for mu and tau, respectively.
alpha	The alpha (expected type I error) used for the credible intervals.
robust	Whether or not to use a t-distribution model; default: FALSE.
df	Degrees of freedom for the t-model, only used if robust is TRUE.
seed	The seed for the random number generator.
showProgressBar	Showing a progress bar for MCMC?

Value

A data frame with the point estimates and 95% credible intervals for the mu and tau parameters (the mean and standard deviation of the distribution from which the per-site effect sizes are drawn). Attributes of the data frame contain the MCMC trace and the detected approximation type.

See Also

[approximateLikelihood](#), [computeFixedEffectMetaAnalysis](#)

Examples

```
# Simulate some data for this example:
populations <- simulatePopulations()

# Fit a Cox regression at each data site, and approximate likelihood function:
fitModelInDatabase <- function(population) {
  cyclopsData <- Cyclops::createCyclopsData(Surv(time, y) ~ x + strata(stratumId),
    data = population,
    modelType = "cox"
  )
  cyclopsFit <- Cyclops::fitCyclopsModel(cyclopsData)
  approximation <- approximateLikelihood(cyclopsFit, parameter = "x", approximation = "custom")
  return(approximation)
}
approximations <- lapply(populations, fitModelInDatabase)
approximations <- do.call("rbind", approximations)

# At study coordinating center, perform meta-analysis using per-site approximations:
estimate <- computeBayesianMetaAnalysis(approximations)
estimate

# (Estimates in this example will vary due to the random simulation)
```

```
computeConfidenceInterval
```

Compute the point estimate and confidence interval given a likelihood function approximation

Description

Compute the point estimate and confidence interval given a likelihood function approximation

Usage

```
computeConfidenceInterval(approximation, alpha = 0.05)
```

Arguments

approximation	An approximation of the likelihood function as fitted using the <code>approximateLikelihood()</code> function.
alpha	The alpha (expected type I error).

Details

Compute the point estimate and confidence interval given a likelihood function approximation.

Value

A data frame containing the point estimate, and upper and lower bound of the confidence interval.

Examples

```
# Simulate some data for this example:
populations <- simulatePopulations()

cyclopsData <- Cyclops::createCyclopsData(Surv(time, y) ~ x + strata(stratumId),
  data = populations[[1]],
  modelType = "cox"
)
cyclopsFit <- Cyclops::fitCyclopsModel(cyclopsData)
approximation <- approximateLikelihood(cyclopsFit, "x")
computeConfidenceInterval(approximation)
```

```
computeFixedEffectMetaAnalysis
```

Compute a fixed-effect meta-analysis

Description

Compute a fixed-effect meta-analysis using a choice of various likelihood approximations.

Usage

```
computeFixedEffectMetaAnalysis(data, alpha = 0.05)
```

Arguments

<code>data</code>	A data frame containing either normal, skew-normal, custom parametric, or grid likelihood data. One row per database.
<code>alpha</code>	The alpha (expected type I error) used for the confidence intervals.

Value

The meta-analytic estimate, expressed as the point estimate hazard ratio (rr), its 95 percent confidence interval (lb, ub), as well as the log of the point estimate (logRr), and the standard error (seLogRr).

See Also

[approximateLikelihood](#), [computeBayesianMetaAnalysis](#)

Examples

```
# Simulate some data for this example:
populations <- simulatePopulations()

# Fit a Cox regression at each data site, and approximate likelihood function:
fitModelInDatabase <- function(population) {
  cyclopsData <- Cyclops::createCyclopsData(Surv(time, y) ~ x + strata(stratumId),
    data = population,
    modelType = "cox"
  )
  cyclopsFit <- Cyclops::fitCyclopsModel(cyclopsData)
  approximation <- approximateLikelihood(cyclopsFit, parameter = "x", approximation = "custom")
  return(approximation)
}
approximations <- lapply(populations, fitModelInDatabase)
approximations <- do.call("rbind", approximations)

# At study coordinating center, perform meta-analysis using per-site approximations:
computeFixedEffectMetaAnalysis(approximations)

# (Estimates in this example will vary due to the random simulation)
```

```
computeHierarchicalMetaAnalysis
```

Compute a Bayesian random-effects hierarchical meta-analysis

Description

Compute a Bayesian hierarchical meta-analysis (two-level model) to learn the global effect with bias correction via negative control outcomes analysis. Bayesian inference is performed using the Markov chain Monte Carlo (MCMC) engine BEAST. Normal priors are used for the global effect, outcome-specific effects, and data-source-specific effects; a half normal prior is used for the standard deviation; a gamma prior is used for the precision parameters.

Usage

```
computeHierarchicalMetaAnalysis(
  data,
  settings = generateBayesianHMAsettings(),
  alpha = 0.05,
  seed = 1,
  showProgressBar = TRUE
)
```

Arguments

data	A data frame containing either normal, skew-normal, custom parametric, or grid likelihood data, with one row per database.
settings	Model settings list generated by generateBayesianHMAsettings()
alpha	The alpha (expected type I error) used for the credible intervals.
seed	Seed for the random number generator.
showProgressBar	Showing a progress bar for MCMC?

Value

A data frame with the point estimates, 95% credible intervals and sample standard errors for the (de-biased) global main effect, the average outcome effect, the average data source effect, and precision of random errors. Attributes of the data frame contain the MCMC trace and the detected approximation type.

See Also

[approximateLikelihood](#), [computeBayesianMetaAnalysis](#)

Examples

```
data("hmaLikelihoodList")
estimates <- EvidenceSynthesis::computeHierarchicalMetaAnalysis(
  data = hmaLikelihoodList,
  seed = 666
)
```

constructDataModel	<i>Construct DataModel objects from approximate likelihood or profile likelihood data</i>
--------------------	---

Description

Construct DataModel objects from approximate likelihood or profile likelihood data

Usage

```
constructDataModel(data, labelReferences = NULL)
```

Arguments

data The likelihood data. Can be a single approximation, approximations from multiple sites, or (adaptive) grid profile likelihoods.

labelReferences Optional parameter that provides a reference list that maps string names to integer indices; only applies to "grid" or "adaptive grip" type of data.

Examples

```
data("likelihoodProfileLists")
dataModel <- constructDataModel(likelihoodProfileLists[[1]])
```

`createApproximations` *Create likelihood approximations from individual-trajectory data*

Description

Create likelihood approximations from individual-trajectory data

Usage

```
createApproximations(populations, approximation)
```

Arguments

populations Individual-level population data

approximation Type of approximation method

`createSccsSimulationSettings`
 Create SCCS simulation settings

Description

Create an object specifying a simulation for the Self-Controlled Case Series (SCCS).

Usage

```
createSccsSimulationSettings(
  nSites = 5,
  n = 10000,
  atRiskTimeFraction = 0.1,
  timePartitions = 24,
  timeCovariates = 5,
  timeEffectSize = log(2),
  minBackgroundRate = 0.001,
  maxBackgroundRate = 0.01,
  rateRatio = 2,
  randomEffectSd = 0
)
```

Arguments

nSites	Number of database sites to simulate.
n	Number of subjects per site. Either a single number, or a vector of length nSites.
atRiskTimeFraction	Fraction of patient time when at risk (exposed). Either a single number, or a vector of length nSites.
timePartitions	Number of time partitions for seasonal covariates. Either a single number, or a vector of length nSites.
timeCovariates	Number of covariates to represent seasonality. Either a single number, or a vector of length nSites.
timeEffectSize	Strength of the seasonality effect. Either a single number, or a vector of length nSites.
minBackgroundRate	Minimum background outcome rate. Either a single number, or a vector of length nSites.
maxBackgroundRate	Maximum background outcome rate. Either a single number, or a vector of length nSites.
rateRatio	The incidence rate ratio.
randomEffectSd	Standard deviation of the log(hazardRatio). Fixed effect if equal to 0.

Value

An object of type `simulationSccsSettings`, to be used in the `simulatePopulations()` function.

See Also

[simulatePopulations](#)

Examples

```
settings <- createSccsSimulationSettings(nSites = 1, rateRatio = 2)
populations <- simulatePopulations(settings)

# Fit a SCCS regression for the simulated data site:
cyclopsData <- Cyclops::createCyclopsData(
  y ~ a + x1 + x2 + x3 + x4 + x5 + strata(stratumId) + offset(log(time)),
  data = populations[[1]],
  modelType = "cpr"
)
cyclopsFit <- Cyclops::fitCyclopsModel(cyclopsData)
coef(cyclopsFit)

# (Estimates in this example will vary due to the random simulation)
```

createSimulationSettings
Create simulation settings

Description

Create an object specifying a simulation. Currently only Cox proportional hazard models are supported.

Usage

```
createSimulationSettings(  
  nSites = 5,  
  n = 10000,  
  treatedFraction = 0.2,  
  nStrata = 10,  
  minBackgroundHazard = 2e-07,  
  maxBackgroundHazard = 2e-05,  
  hazardRatio = 2,  
  randomEffectSd = 0,  
  siteEffects = 0  
)
```

Arguments

nSites	Number of database sites to simulate.
n	Number of subjects per site. Either a single number, or a vector of length nSites.
treatedFraction	Fraction of subjects that is treated. Either a single number, or a vector of length nSites.
nStrata	Number of strata per site. Either a single number, or a vector of length nSites.
minBackgroundHazard	Minimum background hazard. Either a single number, or a vector of length nSites.
maxBackgroundHazard	Maximum background hazard. Either a single number, or a vector of length nSites.
hazardRatio	Hazard ratio.
randomEffectSd	Standard deviation of the log(hazardRatio). Fixed effect if equal to 0.
siteEffects	Fixed site effects (if assuming varying site-specific effects). Same effects if 0.

Value

An object of type simulationSettings, to be used in the [simulatePopulations\(\)](#) function.

See Also

[simulatePopulations](#)

Examples

```

settings <- createSimulationSettings(nSites = 1, hazardRatio = 2)
populations <- simulatePopulations(settings)

# Fit a Cox regression for the simulated data site:
cyclopsData <- Cyclops::createCyclopsData(Surv(time, y) ~ x + strata(stratumId),
  data = populations[[1]],
  modelType = "cox"
)
cyclopsFit <- Cyclops::fitCyclopsModel(cyclopsData)
coef(cyclopsFit)

# (Estimates in this example will vary due to the random simulation)

```

customFunction

A custom function to approximate a log likelihood function

Description

A custom function to approximate a log likelihood function

Usage

```
customFunction(x, mu, sigma, gamma)
```

Arguments

x	The log(hazard ratio) for which to approximate the log likelihood.
mu	The position parameter.
sigma	The scale parameter.
gamma	The skew parameter.

Details

A custom parametric function designed to approximate the shape of the Cox log likelihood function. When $\gamma = 0$ this function is the normal distribution.

Value

The approximate log likelihood for the given x.

Examples

```
customFunction(x = 0:3, mu = 0, sigma = 1, gamma = 0)
```

detectApproximationType

Detect the type of likelihood approximation based on the data format

Description

Detect the type of likelihood approximation based on the data format

Usage

```
detectApproximationType(data, verbose = TRUE)
```

Arguments

data	The approximation data. Can be a single approximation, or approximations from multiple sites.
verbose	Should the detected type be communicated to the user?

Value

A character vector with one of the following values: "normal", "custom", "skew normal", "pooled", "grid", or "adaptive grid".

Examples

```
detectApproximationType(data.frame(logRr = 1, seLogRr = 0.1))
```

extractSourceSpecificEffects

Compute source-specific biases and bias-corrected estimates from hierarchical meta analysis results

Description

Extract source-specific biases and obtain bias-corrected estimates for each data source, given the results from computeHierarchicalMetaAnalysis().

Usage

```
extractSourceSpecificEffects(estimates, alpha = 0.05)
```

Arguments

estimates	A data frame as output from the computeHierarchicalMetaAnalysis() function.
alpha	The alpha (expected type I error) used for the credible intervals.

Value

A data frame with point estimates, 95% credible intervals and sample standard errors for the effect size after bias correction within each data source.

See Also

[computeHierarchicalMetaAnalysis](#)

fitBiasDistribution	<i>Fit Bias Distribution</i>
---------------------	------------------------------

Description

Learn an empirical distribution on estimation bias by simultaneously analyzing a large set of negative control outcomes by a Bayesian hierarchical model through MCMC. Analysis is based on a list of extracted likelihood profiles.

Usage

```
fitBiasDistribution(
  likelihoodProfiles,
  priorSds = c(2, 0.5),
  numsamps = 10000,
  thin = 10,
  minNCs = 5,
  robust = FALSE,
  df = 4,
  seed = 1
)
```

Arguments

likelihoodProfiles	A list of grid profile likelihoods regarding negative controls.
priorSds	A two-dimensional vector with the standard deviation of the prior for the average bias and the sd/scale parameter, respectively.
numsamps	Total number of MCMC samples needed.
thin	Thinning frequency: how many iterations before another sample is obtained?
minNCs	Minimum number of negative controls needed to fit a bias distribution; default (also recommended): 5.
robust	Whether or not to use a t-distribution model; default: FALSE.
df	Degrees of freedom for the t-model, only used if robust is TRUE.
seed	Seed for the random number generator.

Value

A dataframe with three columns and numsamps number of rows. Column mean includes MCMC samples for the average bias, scale for the sd/scale parameter, and bias for predictive samples of the bias.

See Also

[computeBayesianMetaAnalysis](#)

Examples

```
# load example data
data("ncLikelihoods")

# fit a bias distributions by analyzing a set of negative control outcomes
# for example, for the 5th analysis period, and using the t model
# NOT RUN
# biasDistribution = fitBiasDistribution(ncLikelihoods[[5]], robust = TRUE)
```

```
generateBayesianHMAsettings
```

Generate settings for the Bayesian random-effects hierarchical meta-analysis model

Description

This function generates a settings list for fitting a Bayesian hierarchical meta-analysis model. See `computeHierarchicalMetaAnalysis()` for more details.

Usage

```
generateBayesianHMAsettings(
  primaryEffectPriorStd = 1,
  secondaryEffectPriorStd = 1,
  globalExposureEffectPriorMean = c(0),
  globalExposureEffectPriorStd = c(2),
  primaryEffectPrecisionPrior = c(1, 1),
  secondaryEffectPrecisionPrior = c(1, 1),
  errorPrecisionPrior = c(1, 1),
  errorPrecisionStartValue = 1,
  includeSourceEffect = TRUE,
  includeExposureEffect = TRUE,
  exposureEffectCount = 1,
  separateExposurePrior = FALSE,
  chainLength = 1100000,
  burnIn = 1e+05,
  subSampleFrequency = 100
)
```

Arguments

`primaryEffectPriorStd`

Standard deviation for the average outcome effect.

`secondaryEffectPriorStd`

Standard deviation for the average data-source effect.

globalExposureEffectPriorMean	Prior mean for the global main exposure effect; can be a multiple entry vector if there are multiple outcomes of interest
globalExposureEffectPriorStd	Prior standard deviation for the global main exposure effect; can be a multiple entry vector if there are multiple outcomes of interest
primaryEffectPrecisionPrior	Shape and scale for the gamma prior of the precision term in the random effects model (normal) for individual outcome effects.
secondaryEffectPrecisionPrior	Shape and scale for the gamma prior of the precision term in the random effects model (normal) for individual data-source effects.
errorPrecisionPrior	Shape and scale for the gamma prior of the precision term in the normal model for random errors.
errorPrecisionStartValue	Initial value for the error distribution's precision term.
includeSourceEffect	Whether or not to consider the data-source-specific (secondary) random effects. Default is TRUE.
includeExposureEffect	Whether or not to estimate the main effect of interest. Default is TRUE.
exposureEffectCount	Number of main outcomes of interest to estimate effect for? Default = 1
separateExposurePrior	Use a separable prior on the main exposure effect? Default is FALSE.
chainLength	Number of MCMC iterations.
burnIn	Number of MCMC iterations to consider as burn in.
subSampleFrequency	Subsample ("thinning") frequency for the MCMC.

Value

A list with all the settings to use in the `computeHierarchicalMetaAnalysis()` function.

See Also

[computeHierarchicalMetaAnalysis](#)

hermiteInterpolation	<i>Cubic Hermite interpolation using both values and gradients to approximate a log likelihood function</i>
----------------------	---

Description

Cubic Hermite interpolation using both values and gradients to approximate a log likelihood function

Usage

```
hermiteInterpolation(x, profile)
```

Arguments

x	The log(hazard ratio) for which to approximate the log likelihood.
profile	A profile as created with <code>approximateLikelihood()</code> with <code>approximation = "grid with gradients"</code> . This is a data frame with 3 columns: point, value, and derivative, sorted by point.

Details

Performs spline interpolation using cubic Hermite polynomials (Catmull et al. 1974) between the points specified in the profile. We use linear extrapolation outside the points.

Value

The approximate log likelihood for the given x.

References

Catmull, Edwin; Rom, Raphael (1974), "A class of local interpolating splines", in Barnhill, R. E.; Riesenfeld, R. F. (eds.), *Computer Aided Geometric Design*, New York: Academic Press, pp. 317–326

Examples

```
profile <- data.frame(point = c(1.1, 2.1), value = c(1, 1), derivative = c(0.1, -0.1))
hermiteInterpolation(x = 0:3, profile = profile)
```

hmaLikelihoodList	<i>Example profile likelihoods for hierarchical meta analysis with bias correction</i>
-------------------	--

Description

A list that contains profile likelihoods for two negative control outcomes and a synthetic outcome of interest, across four data sources. Each element of the list contains profile likelihoods for one outcome, where each row provides profile likelihood values (over a grid) from one data source.

Usage

```
hmaLikelihoodList
```

Format

An objects of class `list`; the list contains 3 dataframes, where each dataframe includes four rows of likelihood function values corresponding to the points in the column names.

Examples

```
data("hmaLikelihoodList")
hmaLikEx <- hmaLikelihoodList[[1]]

plot(as.numeric(hmaLikEx[2, ]) ~ as.numeric(names(hmaLikEx)))
```

likelihoodProfileLists

A bigger example of profile likelihoods for hierarchical meta analysis with bias correction

Description

A list that contains profile likelihoods for 10 negative control outcomes and an outcome of interest, across data sources. Each element of the list contains a named list of profile likelihoods for one outcome, where each element is a data frame that provides likelihood values over a grid of parameter values, the element name corresponding to data source name.

Usage

```
likelihoodProfileLists
```

Format

An objects of class `list`; the list contains 11 named lists, each list for one outcome. Each list contains data frames that record profile likelihoods from different data sources. The first 10 list corresponds to 10 negative control outcomes, whereas the last list the outcome of interest.

Examples

```
data("likelihoodProfileLists")
exLP <- likelihoodProfileLists[[1]][[1]]

plot(value ~ point, data = exLP)
```

ncLikelihoods

Example profile likelihoods for negative control outcomes

Description

A list that contain profile likelihoods a large set of negative control outcomes. They are extracted from a real-world observational healthcare database, with the likelihoods profiled using adaptive grids using the Cyclops package.

Usage

```
ncLikelihoods
```

Format

An object of class `list` containing 12 lists, where each list includes several dataframes ith column `point` and `value` for adaptive grid profile likelihoods.

References

Schuemie et al. (2022). Vaccine safety surveillance using routinely collected healthcare data—an empirical evaluation of epidemiological designs. *Frontiers in Pharmacology*.

Examples

```
data("ncLikelihoods")
ncLikEx <- ncLikelihoods[["5"]][[1]]

plot(value ~ point, data = ncLikEx)
```

ooiLikelihoods

Example profile likelihoods for a synthetic outcome of interest

Description

A list that contain profile likelihoods for a synthetic outcome of interest. They are extracted from a real-world observational healthcare database, with the likelihoods profiled using adaptive grids using the Cyclops package.

Usage

```
ooiLikelihoods
```

Format

An objects of class `list`; the list contains 12 lists, where each list includes several dataframes with column `point` and `value` for adaptive grid profile likelihoods.

References

Schuemie et al. (2022). Vaccine safety surveillance using routinely collected healthcare data—an empirical evaluation of epidemiological designs. *Frontiers in Pharmacology*.

Examples

```
data("ooiLikelihoods")
ooiLikEx <- ooiLikelihoods[["5"]][[1]]

plot(value ~ point, data = ooiLikEx)
```

plotBiasCorrectionInference
Plot bias correction inference

Description

Plot bias correction inference

Usage

```
plotBiasCorrectionInference(  
  bbcResult,  
  type = "raw",  
  ids = bbcResult$Id,  
  limits = c(-3, 3),  
  logScale = FALSE,  
  numericId = TRUE,  
  fileName = NULL  
)
```

Arguments

bbcResult	A (sequential) analysis object generated by the biasCorrectionInference() function.
type	The type of plot. Must be one of <code>c("corrected", "raw", "compare")</code> .
ids	IDs of the periods/groups to plot result for; default is all IDs.
limits	The limits on log RR for plotting.
logScale	Whether or not to show bias in log-RR; default FALSE (shown in RR).
numericId	Whether or not to treat Id as a numeric variable; default: TRUE.
fileName	Name of the file where the plot should be saved, for example 'plot.png'. See the function ggplot2::ggsave in the ggplot2 package for supported file formats.

Details

Plot empirical bias distributions learned from analyzing negative controls.

Value

A ggplot object. Use the [ggplot2::ggsave](#) function to save to file.

See Also

[biasCorrectionInference](#)

Examples

```
# Perform sequential analysis using Bayesian bias correction for this example:
data("ncLikelihoods")
data("ooiLikelihoods")
# NOT RUN
# bbcSequential = biasCorrectionInference(ooiLikelihoods, ncLikelihoodProfiles = ncLikelihoods)

# Plot it
# NOT RUN
# plotBiasCorrectionInference(bbcSequential, type = "corrected")
```

plotBiasDistribution *Plot bias distributions*

Description

Plot bias distributions

Usage

```
plotBiasDistribution(
  biasDist,
  limits = c(-2, 2),
  logScale = FALSE,
  numericId = TRUE,
  fileName = NULL
)
```

Arguments

biasDist	A bias distribution object generated by the fitBiasDistribution() or sequentialFitBiasDistribution() function.
limits	The lower and upper limits in log-RR to plot.
logScale	Whether or not to show bias in log-RR; default FALSE (shown in RR).
numericId	(For sequential or group case only) whether or not to treat Id as a numeric variable; default: TRUE.
fileName	Name of the file where the plot should be saved, for example 'plot.png'. See the function ggplot2::ggsave in the ggplot2 package for supported file formats.

Details

Plot empirical bias distributions learned from analyzing negative controls.

Value

A ggplot object. Use the [ggplot2::ggsave](#) function to save to file.

See Also

[fitBiasDistribution](#), [sequentialFitBiasDistribution](#)

Examples

```
# Fit a bias distribution for this example:
data("nclikelihoods")
# NOT RUN
# singleBiasDist = fitBiasDistribution(nclikelihoods[[5]], seed = 1)

# Plot it
# NOT RUN
# plotBiasDistribution(singleBiasDist)
```

plotCovariateBalances *Plot covariate balances*

Description

Plots the covariate balance before and after matching for multiple data sources.

Usage

```
plotCovariateBalances(
  balances,
  labels,
  threshold = 0,
  beforeLabel = "Before matching",
  afterLabel = "After matching",
  fileName = NULL
)
```

Arguments

balances	A list of covariate balance objects as created using the <code>computeCovariateBalance()</code> function in the <code>CohortMethod</code> package. Each balance object is expected to be a data frame with at least these two columns: <code>beforeMatchingStdDiff</code> and <code>afterMatchingStdDiff</code> .
labels	A vector containing the labels for the various sources.
threshold	Show a threshold value for the standardized difference.
beforeLabel	Label for before matching / stratification / trimming.
afterLabel	Label for after matching / stratification / trimming.
fileName	Name of the file where the plot should be saved, for example 'plot.png'. See the function ggplot2::ggsave for supported file formats.

Details

Creates a plot showing the covariate balance before and after matching. Balance distributions are displayed as box plots combined with scatterplots.

Value

A Ggplot object. Use the [ggplot2::ggsave](#).

Examples

```
# Some example data:
balance1 <- data.frame(
  beforeMatchingStdDiff = rnorm(1000, 0.1, 0.1),
  afterMatchingStdDiff = rnorm(1000, 0, 0.01)
)
balance2 <- data.frame(
  beforeMatchingStdDiff = rnorm(1000, 0.2, 0.1),
  afterMatchingStdDiff = rnorm(1000, 0, 0.05)
)
balance3 <- data.frame(
  beforeMatchingStdDiff = rnorm(1000, 0, 0.1),
  afterMatchingStdDiff = rnorm(1000, 0, 0.03)
)
plotCovariateBalances(
  balances = list(balance1, balance2, balance3),
  labels = c("Site A", "Site B", "Site C")
)
```

plotEmpiricalNulls	<i>Plot empirical null distributions</i>
--------------------	--

Description

Plot the empirical null distribution for multiple data sources.

Usage

```
plotEmpiricalNulls(
  logRr,
  seLogRr,
  labels,
  xLabel = "Relative risk",
  limits = c(0.1, 10),
  showCis = TRUE,
  fileName = NULL
)
```

Arguments

logRr	A numeric vector of effect estimates for the negative controls on the log scale.
seLogRr	The standard error of the log of the effect estimates. Hint: often the standard error = (log(lower bound 95 percent confidence interval) - log(effect estimate))/qnorm(0.025).
labels	A vector containing the labels for the various sources. Should be of equal length as logRr and seLogRr.
xLabel	The label on the x-axis: the name of the effect estimate.
limits	The limits of the effect size axis.
showCis	Show the 95 percent confidence intervals on the null distribution and distribution parameter estimates?

`fileName` Name of the file where the plot should be saved, for example 'plot.png'. See the function `ggplot2::ggsave()` for supported file formats.

Details

Creates a plot showing the empirical null distributions. Distributions are shown as mean plus minus one standard deviation, as well as a distribution plot.

Value

A Ggplot object. Use the `ggplot2::ggsave()` function to save to file.

See Also

[EmpiricalCalibration::fitNull](#), [EmpiricalCalibration::fitMcmcNull](#)

Examples

```
# Some example data:
site1 <- EmpiricalCalibration::simulateControls(n = 50, mean = 0, sd = 0.1, trueLogRr = 0)
site1$label <- "Site 1"
site2 <- EmpiricalCalibration::simulateControls(n = 50, mean = 0.1, sd = 0.2, trueLogRr = 0)
site2$label <- "Site 2"
site3 <- EmpiricalCalibration::simulateControls(n = 50, mean = 0.15, sd = 0.25, trueLogRr = 0)
site3$label <- "Site 3"
sites <- rbind(site1, site2, site3)

plotEmpiricalNulls(logRr = sites$logRr, seLogRr = sites$seLogRr, labels = sites$label)
```

plotLikelihoodFit	<i>Plot the likelihood approximation</i>
-------------------	--

Description

Plot the likelihood approximation

Usage

```
plotLikelihoodFit(
  approximation,
  cyclopsFit,
  parameter = "x",
  logScale = TRUE,
  xLabel = "Hazard Ratio",
  limits = c(0.1, 10),
  fileName = NULL
)
```

Arguments

approximation	An approximation of the likelihood function as fitted using the approximateLikelihood() function.
cyclopsFit	A model fitted using the Cyclops::fitCyclopsModel() function.
parameter	The parameter in the cyclopsFit object to profile.
logScale	Show the y-axis on the log scale?
xLabel	The title of the x-axis.
limits	The limits on the x-axis.
fileName	Name of the file where the plot should be saved, for example 'plot.png'. See the function ggplot2::ggsave in the ggplot2 package for supported file formats.

Details

Plots the (log) likelihood and the approximation of the likelihood. Allows for reviewing the approximation.

Value

A Ggplot object. Use the [ggplot2::ggsave](#) function to save to file.

Examples

```
# Simulate a single database population:
population <- simulatePopulations(createSimulationSettings(nSites = 1))[[1]]

# Approximate the likelihood:
cyclopsData <- Cyclops::createCyclopsData(Surv(time, y) ~ x + strata(stratumId),
  data = population,
  modelType = "cox"
)
cyclopsFit <- Cyclops::fitCyclopsModel(cyclopsData)
approximation <- approximateLikelihood(cyclopsFit, parameter = "x", approximation = "custom")

plotLikelihoodFit(approximation, cyclopsFit, parameter = "x")
```

plotMcmcTrace

Plot MCMC trace

Description

Plot MCMC trace

Usage

```
plotMcmcTrace(
  estimate,
  showEstimate = TRUE,
  dataCutoff = 0.01,
  fileName = NULL
)
```

Arguments

estimate	An object as generated using the computeBayesianMetaAnalysis() function.
showEstimate	Show the parameter estimates (mode) and 95 percent confidence intervals?
dataCutoff	This fraction of the data at both tails will be removed.
fileName	Name of the file where the plot should be saved, for example 'plot.png'. See the function ggplot2::ggsave in the ggplot2 package for supported file formats.

Details

Plot the samples of the posterior distribution of the mu and tau parameters. Samples are taken using Markov-chain Monte Carlo (MCMC).

Value

A Ggplot object. Use the [ggplot2::ggsave](#) function to save to file.

See Also

[computeBayesianMetaAnalysis](#)

Examples

```
# Simulate some data for this example:
populations <- simulatePopulations()

# Fit a Cox regression at each data site, and approximate likelihood function:
fitModelInDatabase <- function(population) {
  cyclopsData <- Cyclops::createCyclopsData(Surv(time, y) ~ x + strata(stratumId),
    data = population,
    modelType = "cox"
  )
  cyclopsFit <- Cyclops::fitCyclopsModel(cyclopsData)
  approximation <- approximateLikelihood(cyclopsFit, parameter = "x", approximation = "custom")
  return(approximation)
}
approximations <- lapply(populations, fitModelInDatabase)
approximations <- do.call("rbind", approximations)

# At study coordinating center, perform meta-analysis using per-site approximations:
estimate <- computeBayesianMetaAnalysis(approximations)
plotMcmcTrace(estimate)
```

plotMetaAnalysisForest

Create a forest plot

Description

Creates a forest plot of effect size estimates, including the summary estimate.

Usage

```
plotMetaAnalysisForest(
  data,
  labels,
  estimate,
  xLabel = "Relative risk",
  summaryLabel = "Summary",
  limits = c(0.1, 10),
  alpha = 0.05,
  showLikelihood = TRUE,
  fileName = NULL
)
```

Arguments

<code>data</code>	A data frame containing either normal, skew-normal, custom parametric, or grid likelihood data. One row per database.
<code>labels</code>	A vector of labels for the data sources.
<code>estimate</code>	The meta-analytic estimate as created using either [<code>'computeFixedEffectMetaAnalysis()'</code>] or [<code>'computeBayesianMetaAnalysis()'</code>] function.
<code>xLabel</code>	The label on the x-axis: the name of the effect estimate.
<code>summaryLabel</code>	The label for the meta-analytic estimate.
<code>limits</code>	The limits of the effect size axis.
<code>alpha</code>	The alpha (expected type I error).
<code>showLikelihood</code>	Show the likelihood curve for each estimate?
<code>fileName</code>	Name of the file where the plot should be saved, for example <code>'plot.png'</code> . See the function ggplot2::ggsave ifor supported file formats.

Details

Creates a forest plot of effect size estimates, including a meta-analysis estimate.

Value

A Ggplot object. Use the [ggplot2::ggsave](#) function to save to file.

Examples

```
# Simulate some data for this example:
populations <- simulatePopulations()
labels <- paste("Data site", LETTERS[1:length(populations)])

# Fit a Cox regression at each data site, and approximate likelihood function:
fitModelInDatabase <- function(population) {
  cyclopsData <- Cyclops::createCyclopsData(Surv(time, y) ~ x + strata(stratumId),
    data = population,
    modelType = "cox"
  )
  cyclopsFit <- Cyclops::fitCyclopsModel(cyclopsData)
  approximation <- approximateLikelihood(cyclopsFit, parameter = "x", approximation = "custom")
  return(approximation)
}
```

```

}
approximations <- lapply(populations, fitModelInDatabase)
approximations <- do.call("rbind", approximations)

# At study coordinating center, perform meta-analysis using per-site approximations:
estimate <- computeBayesianMetaAnalysis(approximations)
plotMetaAnalysisForest(approximations, labels, estimate)

# (Estimates in this example will vary due to the random simulation)

```

plotPerDbMcmcTrace	<i>Plot MCMC trace for individual databases</i>
--------------------	---

Description

Plot MCMC trace for individual databases

Usage

```

plotPerDbMcmcTrace(
  estimate,
  showEstimate = TRUE,
  dataCutoff = 0.01,
  fileName = NULL
)

```

Arguments

estimate	An object as generated using the computeBayesianMetaAnalysis() function.
showEstimate	Show the parameter estimates (mode) and 95 percent confidence intervals?
dataCutoff	This fraction of the data at both tails will be removed.
fileName	Name of the file where the plot should be saved, for example 'plot.png'. See the function ggplot2::ggsave in the ggplot2 package for supported file formats.

Details

Plot the samples of the posterior distribution of the theta parameter (the estimated log hazard ratio) at each site. Samples are taken using Markov-chain Monte Carlo (MCMC).

Value

A Ggplot object. Use the [ggplot2::ggsave](#) function to save to file.

See Also

[computeBayesianMetaAnalysis](#)

Examples

```
# Simulate some data for this example:
populations <- simulatePopulations()

# Fit a Cox regression at each data site, and approximate likelihood function:
fitModelInDatabase <- function(population) {
  cyclopsData <- Cyclops::createCyclopsData(Surv(time, y) ~ x + strata(stratumId),
    data = population,
    modelType = "cox"
  )
  cyclopsFit <- Cyclops::fitCyclopsModel(cyclopsData)
  approximation <- approximateLikelihood(cyclopsFit, parameter = "x", approximation = "custom")
  return(approximation)
}
approximations <- lapply(populations, fitModelInDatabase)
approximations <- do.call("rbind", approximations)

# At study coordinating center, perform meta-analysis using per-site approximations:
estimate <- computeBayesianMetaAnalysis(approximations)
plotPerDbMcmcTrace(estimate)
```

plotPerDbPosterior	<i>Plot posterior density per database</i>
--------------------	--

Description

Plot posterior density per database

Usage

```
plotPerDbPosterior(
  estimate,
  showEstimate = TRUE,
  dataCutoff = 0.01,
  fileName = NULL
)
```

Arguments

estimate	An object as generated using the computeBayesianMetaAnalysis() function.
showEstimate	Show the parameter estimates (mode) and 95 percent confidence intervals?
dataCutoff	This fraction of the data at both tails will be removed.
fileName	Name of the file where the plot should be saved, for example 'plot.png'. See the function ggplot2::ggsave in the ggplot2 package for supported file formats.

Details

Plot the density of the posterior distribution of the theta parameter (the estimated log hazard ratio) at each site.

Value

A Ggplot object. Use the [ggplot2::ggsave](#) function to save to file.

Examples

```
# Simulate some data for this example:
populations <- simulatePopulations()

# Fit a Cox regression at each data site, and approximate likelihood function:
fitModelInDatabase <- function(population) {
  cyclopsData <- Cyclops::createCyclopsData(Surv(time, y) ~ x + strata(stratumId),
    data = population,
    modelType = "cox"
  )
  cyclopsFit <- Cyclops::fitCyclopsModel(cyclopsData)
  approximation <- approximateLikelihood(cyclopsFit, parameter = "x", approximation = "custom")
  return(approximation)
}
approximations <- lapply(populations, fitModelInDatabase)
approximations <- do.call("rbind", approximations)

# At study coordinating center, perform meta-analysis using per-site approximations:
estimate <- computeBayesianMetaAnalysis(approximations)
plotPerDbPosterior(estimate)
```

plotPosterior

Plot posterior density

Description

Plot posterior density

Usage

```
plotPosterior(
  estimate,
  showEstimate = TRUE,
  dataCutoff = 0.01,
  fileName = NULL
)
```

Arguments

estimate	An object as generated using the computeBayesianMetaAnalysis() function.
showEstimate	Show the parameter estimates (mode) and 95 percent confidence intervals?
dataCutoff	This fraction of the data at both tails will be removed.
fileName	Name of the file where the plot should be saved, for example 'plot.png'. See the function ggplot2::ggsave in the ggplot2 package for supported file formats.

Details

Plot the density of the posterior distribution of the mu and tau parameters.

Value

A Ggplot object. Use the [ggplot2::ggsave](#) function to save to file.

See Also

[computeBayesianMetaAnalysis](#)

Examples

```
# Simulate some data for this example:
populations <- simulatePopulations()

# Fit a Cox regression at each data site, and approximate likelihood function:
fitModelInDatabase <- function(population) {
  cyclopsData <- Cyclops::createCyclopsData(Surv(time, y) ~ x + strata(stratumId),
    data = population,
    modelType = "cox"
  )
  cyclopsFit <- Cyclops::fitCyclopsModel(cyclopsData)
  approximation <- approximateLikelihood(cyclopsFit, parameter = "x", approximation = "custom")
  return(approximation)
}
approximations <- lapply(populations, fitModelInDatabase)
approximations <- do.call("rbind", approximations)

# At study coordinating center, perform meta-analysis using per-site approximations:
estimate <- computeBayesianMetaAnalysis(approximations)
plotPosterior(estimate)
```

plotPreparedPs

Plot the propensity score distribution

Description

Plot the propensity score distribution

Usage

```
plotPreparedPs(
  preparedPsPlots,
  labels,
  treatmentLabel = "Target",
  comparatorLabel = "Comparator",
  fileName = NULL
)
```

Arguments

preparedPsPlots	list of prepared propensity score data as created by the preparePsPlot() function.
labels	A vector containing the labels for the various sources.

treatmentLabel A label to us for the treated cohort.
 comparatorLabel A label to us for the comparator cohort.
 fileName Name of the file where the plot should be saved, for example 'plot.png'. See the function [ggplot2::ggsave](#) for supported file formats.

Value

A ggplot object. Use the [ggplot2::ggsave](#) function to save to file in a different format.

See Also

[preparePsPlot](#)

Examples

```
# Simulate some data for this example:
treatment <- rep(0:1, each = 100)
propensityScore <- c(rnorm(100, mean = 0.4, sd = 0.25), rnorm(100, mean = 0.6, sd = 0.25))
data <- data.frame(treatment = treatment, propensityScore = propensityScore)
data <- data[data$propensityScore > 0 & data$propensityScore < 1, ]
preparedPlot <- preparePsPlot(data)

# Just reusing the same data three times for demonstration purposes:
preparedPsPlots <- list(preparedPlot, preparedPlot, preparedPlot)
labels <- c("Data site A", "Data site B", "Data site C")

plotPreparedPs(preparedPsPlots, labels)
```

```
preparePsPlot
```

```
Prepare to plot the propensity score distribution
```

Description

Prepare to plot the propensity (or preference) score distribution. It computes the distribution, so the output does not contain person-level data.

Usage

```
preparePsPlot(data, unfilteredData = NULL, scale = "preference")
```

Arguments

data A data frame with at least the two columns described below
 unfilteredData To be used when computing preference scores on data from which subjects have already been removed, e.g. through trimming and/or matching. This data frame should have the same structure as data.
 scale The scale of the graph. Two scales are supported: scale = 'propensity' or scale = 'preference'. The preference score scale is defined by Walker et al. (2013).

Details

The data frame should have at least the following two columns:

- **treatment** (integer): Column indicating whether the person is in the treated (1) or comparator (0) group.
- **propensityScore** (numeric): Propensity score.

Value

A data frame describing the propensity score (or preference score) distribution at 100 equally-spaced points.

References

Walker AM, Patrick AR, Lauer MS, Hornbrook MC, Marin MG, Platt R, Roger VL, Stang P, and Schneeweiss S. (2013) A tool for assessing the feasibility of comparative effectiveness research, *Comparative Effective Research*, 3, 11-20

See Also

[plotPreparedPs](#)

Examples

```
# Simulate some data for this example:
treatment <- rep(0:1, each = 100)
propensityScore <- c(rnorm(100, mean = 0.4, sd = 0.25), rnorm(100, mean = 0.6, sd = 0.25))
data <- data.frame(treatment = treatment, propensityScore = propensityScore)
data <- data[data$propensityScore > 0 & data$propensityScore < 1, ]

preparedPlot <- preparePsPlot(data)
```

```
prepareSccsIntervalData
```

Prepare SCCS interval data for pooled analysis

Description

Prepare SCCS interval data for pooled analysis

Usage

```
prepareSccsIntervalData(sccsIntervalData, covariateId)
```

Arguments

`sccsIntervalData`

An object of type `SccsIntervalData` as created using the `createSccsIntervalData` function in the `OHDSI SelfControlledCaseSeries` package.

`covariateId`

The ID of the covariate of interest, for which the estimate will be synthesized. All other covariates will be considered nuisance variables.

Value

A tibble that can be used in the `computeBayesianMetaAnalysis()` function.

`sequentialFitBiasDistribution`

Fit Bias Distribution Sequentially or in Groups

Description

Learn empirical bias distributions sequentially or in groups; for each sequential step or analysis group, bias distributions is learned by by simultaneously analyzing a large set of negative control outcomes by a Bayesian hierarchical model through MCMC.

Usage

```
sequentialFitBiasDistribution(LikelihoodProfileList, ...)
```

Arguments

`LikelihoodProfileList`

A list of lists, each of which is a set of grid profile likelihoods regarding negative controls, indexed by analysis period ID for sequential analyses or group ID for group analyses.

`...`

Arguments passed to the `fitBiasDistribution()` function.

Value

A (long) dataframe with four columns. Column `mean` includes MCMC samples for the average bias, `scale` for the `sd/scale` parameter, `bias` for predictive samples of the bias, and `Id` for the period ID or group ID.

See Also

[fitBiasDistribution](#), [computeBayesianMetaAnalysis](#)

Examples

```
# load example data
data("ncLikelihoods")

# fit bias distributions over analysis periods
# NOT RUN
# biasDistributions = sequentialFitBiasDistribution(ncLikelihoods, seed = 42)
```

```
simulateMetaAnalysisWithNegativeControls
```

Simulate survival data across a federated data network, with negative control outcomes as well.

Description

A function to simulate patient-level survival data for a hypothetical exposure, with simulated bias and data-source-specific random effects. Patient-level data for negative control outcomes are simulated as well to reflect systematic error.

Usage

```
simulateMetaAnalysisWithNegativeControls(
  meanExposureEffect = log(2),
  meanBias = 0.5,
  biasStd = 0.2,
  meanSiteEffect = 0,
  siteEffectStd = 0.1,
  mNegativeControls = 10,
  nSites = 10,
  sitePop = 2000,
  seed = 42,
  ...
)
```

Arguments

<code>meanExposureEffect</code>	Average exposure effect; has to be on the log-scale
<code>meanBias</code>	Average bias for the bias distribution
<code>biasStd</code>	Standard deviation for the bias distribution
<code>meanSiteEffect</code>	Average of the data-source-specific effects (typically should be zero)
<code>siteEffectStd</code>	Standard deviation for data-source-specific effects
<code>mNegativeControls</code>	Number of negative control outcomes
<code>nSites</code>	Number of data sources
<code>sitePop</code>	Population size per data source
<code>seed</code>	Random seed
<code>...</code>	Arguments that will be passed to other functions

See Also

[computeHierarchicalMetaAnalysis](#)

simulatePopulations	<i>Simulate survival data for multiple databases</i>
---------------------	--

Description

Simulate survival data for multiple databases

Usage

```
simulatePopulations(settings = createSimulationSettings())
```

Arguments

settings	Either an object of type <code>simulationSettings</code> , created by the <code>createSimulationSettings()</code> function or an object of type <code>sccsSimulationSettings</code> as created by the <code>createSccsSimulationSettings()</code> function.
----------	---

Value

A object of class `simulation`, which is a list of population data frames. Depending on the type of simulation, the data frames have different columns: Cox simulations will have the columns `rowId`, `stratumId`, `x`, `time`, and `y`. SCCS simulations will have the columns `stratumId`, `a`, `x1...xN`, `time`, and `y`.

Examples

```
settings <- createSimulationSettings(nSites = 1, hazardRatio = 2)
populations <- simulatePopulations(settings)

# Fit a Cox regression for the simulated data site:
cyclopsData <- Cyclops::createCyclopsData(Surv(time, y) ~ x + strata(stratumId),
  data = populations[[1]],
  modelType = "cox"
)
cyclopsFit <- Cyclops::fitCyclopsModel(cyclopsData)
coef(cyclopsFit)

# (Estimates in this example will vary due to the random simulation)
```

skewNormal	<i>The skew normal function to approximate a log likelihood function</i>
------------	--

Description

The skew normal function to approximate a log likelihood function

Usage

```
skewNormal(x, mu, sigma, alpha)
```


Arguments

x	The log(hazard ratio) for which to approximate the log likelihood.
mu	The position parameter.
sigma	The scale parameter.
alpha	The skew parameter.

Details

The skew normal function. When $\alpha = 0$ this function is the normal distribution.

Value

The approximate log likelihood for the given x.

References

Azzalini, A. (2013). The Skew-Normal and Related Families. Institute of Mathematical Statistics Monographs. Cambridge University Press.

Examples

```
skewNormal(x = 0:3, mu = 0, sigma = 1, alpha = 0)
```

summarizeChain	<i>Utility function to summarize MCMC samples (posterior mean, median, HDI, std, etc.)</i>
----------------	--

Description

Utility function to summarize MCMC samples (posterior mean, median, HDI, std, etc.)

Usage

```
summarizeChain(chain, alpha = 0.05)
```

Arguments

chain	A vector of posterior samples from MCMC.
alpha	Alpha level for the credible interval.

`supportsJava8`*Determine if Java virtual machine supports Java*

Description

Tests Java virtual machine (JVM) `java.version` system property to check if version `>= 8`.

Usage

```
supportsJava8()
```

Value

Returns `TRUE` if JVM supports Java `>= 8`.

Examples

```
supportsJava8()
```

Index

* datasets

- hmalLikelihoodList, [21](#)
- likelihoodProfileLists, [22](#)
- ncLikelihoods, [22](#)
- ooiLikelihoods, [23](#)

approximateHierarchicalNormalPosterior, [3](#)

approximateLikelihood, [4](#), [9](#), [11](#), [12](#)

approximateLikelihood(), [10](#), [29](#)

approximateSimplePosterior, [5](#), [7](#)

biasCorrectionInference, [6](#), [24](#)

biasCorrectionInference(), [24](#)

buildLabelReferences, [8](#)

computeBayesianMetaAnalysis, [4](#), [8](#), [11](#), [12](#), [19](#), [30](#), [32](#), [35](#), [38](#)

computeBayesianMetaAnalysis(), [30](#), [32–34](#)

computeConfidenceInterval, [4](#), [10](#)

computeFixedEffectMetaAnalysis, [4](#), [9](#), [10](#)

computeHierarchicalMetaAnalysis, [11](#), [18](#), [20](#), [39](#)

constructDataModel, [12](#)

createApproximations, [13](#)

createSccsSimulationSettings, [13](#)

createSccsSimulationSettings(), [40](#)

createSimulationSettings, [15](#)

createSimulationSettings(), [40](#)

customFunction, [16](#)

Cyclops::fitCyclopsModel(), [4](#), [29](#)

detectApproximationType, [17](#)

EmpiricalCalibration::fitMcmcNull, [28](#)

EmpiricalCalibration::fitNull, [28](#)

extractSourceSpecificEffects, [17](#)

fitBiasDistribution, [7](#), [18](#), [25](#), [38](#)

fitBiasDistribution(), [7](#), [25](#), [38](#)

generateBayesianHMAsettings, [19](#)

ggplot2::ggsave, [24–26](#), [29–36](#)

ggplot2::ggsave(), [28](#)

hermiteInterpolation, [20](#)

hmalLikelihoodList, [21](#)

likelihoodProfileLists, [22](#)

ncLikelihoods, [22](#)

ooiLikelihoods, [23](#)

plotBiasCorrectionInference, [24](#)

plotBiasDistribution, [25](#)

plotCovariateBalances, [26](#)

plotEmpiricalNulls, [27](#)

plotLikelihoodFit, [28](#)

plotMcmcTrace, [29](#)

plotMetaAnalysisForest, [30](#)

plotPerDbMcmcTrace, [32](#)

plotPerDbPosterior, [33](#)

plotPosterior, [34](#)

plotPreparedPs, [35](#), [37](#)

preparePsPlot, [36](#), [36](#)

preparePsPlot(), [35](#)

prepareSccsIntervalData, [37](#)

sequentialFitBiasDistribution, [25](#), [38](#)

sequentialFitBiasDistribution(), [7](#), [25](#)

simulateMetaAnalysisWithNegativeControls, [39](#)

simulatePopulations, [14](#), [15](#), [40](#)

simulatePopulations(), [14](#), [15](#)

skewNormal, [40](#)

summarizeChain, [41](#)

supportsJava8, [42](#)