Package ‘blatent’

December 8, 2023

Type Package

Title Bayesian Latent Variable Models

Version 0.1.2

Maintainer Jonathan Templin <jonathan-templin@uiowa.edu>

Description Estimation of latent variable models using Bayesian methods. Currently estimates the loglinear cognitive diagnosis model of Henson, Templin, and Willse (2009) <doi:10.1007/s11336-008-9089-5>.

License GPL (>= 2)

Encoding UTF-8

Depends coda, mnormt, R (>= 3.0.0), R6, stats, truncnorm

RoxygenNote 7.2.3

LinkingTo Rcpp, RcppArmadillo

Imports Matrix, methods, Rcpp

NeedsCompilation yes

Author Jonathan Templin [aut, cre] (<https://orcid.org/0000-0001-7616-0973>)

Repository CRAN

Date/Publication 2023-12-08 18:40:02 UTC

R topics documented:

`blatent` *blatent: A package for estimating Bayesian latent variable models.*

Description

Estimation of latent variable models using Bayesian methods. Currently supports diagnostic classification models.

`blatentControl` *blatent estimation specifications*

Description

Creates control specifics for estimation options for estimating Bayesian latent variable models.

Usage

```
blatentControl(
  calculateDIC = TRUE,
  calculateWAIC = TRUE,
  defaultPriors = setDefaultPriors(),
  defaultInitializeParameters = setDefaultInitializeParameters(),
  estimateLatents = TRUE,
  estimator = "blatent",
  estimatorType = "R",
  estimatorLocation = "",
  executableName = "",
  fileSaveLocation = paste0(getwd(), "/"),
  HDPIntervalValue = 0.95,
  maxTuneChains = 0,
  minTuneChains = 0,
  missingMethod = "omit",
  nBurnin = 1000,
  nChains = 4,
  nCores = -1,
  nSampled = 1000,
  nThin = 5,
  nTuneIterations = 0,
```

```
  parallel = FALSE,
  posteriorPredictiveChecks = setPosteriorPredictiveCheckOptions(),
  seed = NULL
)
```

Arguments

`calculateDIC` Calculates DIC following Markov chain. DIC will be marginalized for models with latent variables. Defaults to TRUE.

`calculateWAIC` Calculates WAIC following Markov chain. WAIC will be marginalized for models with latent variables. Defaults to TRUE.

`defaultPriors` Sets priors for all parameters that are not specified in priorsList of `blatentEstimate`. Defaults to list set by `setDefaultPriors` function. Values in list currently allowed are

- `normalMean` for the mean of a normal distribution (defaults to `0`).
- `normalVariance` for the variance of a normal distribution (defaults to `1000`).
- `normalCovariance` for the covariance of a multivariate normal distribution (defaults to `0`).

`defaultInitializeParameters`
List of values that sets distributions used to initialize parameters. Defaults to list set by `setDefaultInitializeParameters` function. Values in list currently allowed are:

- `normalMean` for the mean of a normal distribution (defaults to `0`).
- `normalVariance` for the variance of a normal distribution (defaults to `1`).
- `normalCovariance` for the covariance of a multivariate normal distribution (defaults to `0`).

`estimateLatents`
Estimate latent variables summaries for each observation following MCMC estimation. Defaults to `TRUE`.

`estimator` Sets the estimation algorithm to be used. Currently, one option is available that works. The eventual values will be:

- `"blatentEstimator"` Sets the estimation algorithm to be used to the R package `blatentEstimator`, which must be installed (default).
- `"GPDCM"` Gibbs Probit Diagnostic Classification Model is allowed but not functional.

`estimatorType` Sets location of estimator. Currently, only one option (the default) works.

- `"R"` Sets estimation via R packages (default).
- `"external"` for estimation routines external to R. Currently external syntax does not work.

`estimatorLocation`
Sets the path to the location of estimator executable, if `estimatorType` is `"external"`. Currently set to `""`.

`executableName` Sets the name for the executable file for the estimator. Defaults to `""`

`fileSaveLocation`
: Sets the path for output files used for external estimation routines. Only used when `estimatorType = "external"`.

`HDPIntervalValue`
: Sets the value for all highest density posterior interval parameter summaries. Defaults to `0.95`.

`maxTuneChains`
: Sets the maximum number of tuning chains for MCMC sampling algorithm, if needed. Currently, no Metropolis steps exist in algorithm, so is unused. Defaults to `0`.

`minTuneChains`
: Sets the minimum number of tuning chains for MCMC sampling algorithm, if needed. Currently, no Metropolis steps exist in algorithm, so is unused. Defaults to `0`.

`missingMethod`
: Sets the way missing observed variables are treated within algorithm. Defaults to `"skip"`. Current options are:
  - `"skip"` Skips all missing variables in model likelihoods. For dependent variables predicted variables with missing values, omits any case with missing values.
  - `"imputeBayes"` Model-based imputation using Bayes theorem.

`nBurnin`
: Sets the number of burnin iterations. Defaults to `1000`.

`nChains`
: Sets the number of independent Markov chains run by the program. Defaults to `4`.

`nCores`
: Sets the number of cores used in parallel processing if option `parallel` is `TRUE`. Defaults to `-1`. Values are semi-indicative of how many processors will be used:
  - `-1` indicates that all but one available processor will be used.
  - `0` indicates that all available processors will be used.
  - `>0` indicates that specific number of processors will be used, if available.

  Note: currently, parallel processing is unavailable, so this is unused.

`nSampled`
: Sets the number of posterior draws to sample, per chain. Defaults to `1000`.

`nThin`
: Sets the thinning interval, saving only the posterior draws that comes at this value. Defaults to `5`.

`nTuneIterations`
: Sets the number of iterations per tuning chain, if needed. Currently, no Metropolis steps exist in algorithm, so is unused. Defaults to `0`.

`parallel`
: If `TRUE`, enables parallel processing of estimation and PPCM analyses. Currently, parallel processing is unavailable, so this is unused. Defaults to `FALSE`.

`posteriorPredictiveChecks`
: List of values that sets options for posterior predictive model checks. Defaults to list set by `setPosteriorPredictiveCheckOptions` function. Values in list currently allowed are:

`seed`
: Sets the random number seed for the analysis. Defaults to `NULL`, which does not set the seed and uses current session value per each analysis.

Value

A list of values containing named entries for all arguments shown above.

`blatentEstimate` *Use blatent to estimate a Bayesian latent variable model. Currently supports estimation of the LCDM (Loglinar Cognitive Diagnosis Model).*

Description

Blatantly runs Bayesian latent variable models.

Usage

```
blatentEstimate(
  dataMat,
  modelText,
  priorsList = NULL,
  options = blatentControl()
)
```

Arguments

`dataMat`	A data frame containing the data used for the analysis.
`modelText`	A character string that contains the specifications for the model to be run. See `blatentSyntax` or more information about syntax formatting.
`priorsList`	A list of priors to be placed on parameters of the model. Defaults to NULL. Currently only accepts NULL. All priors not set in `priorsList` will be set in options using `blatentControl` via the `setDefaultPriors` function.
`options`	A list of options for estimating the model. Use the `blatentControl` function to specify the options. See `blatentControl` for more information and default values.

Value

A blatentModel object (an R6 class).

`blatentPPMC` *blatentPPMC*

Description

Simulates data using parameters from posterior distribution of blatent Markov chain.

Usage

```
blatentPPMC(
  model,
  nSamples,
  seed = model$options$seed,
  parallel = TRUE,
  nCores = 4,
  type = c("mean", "covariance", "univariate", "bivariate", "tetrachoric", "pearson"),
  lowPPMCpercentile = c(0.025, 0.025, 0, 0, 0.025, 0.025),
  highPPMCpercentile = c(0.975, 0.975, 1, 1, 0.975, 0.975)
)
```

Arguments

`model` A blatent MCMC model object.

`nSamples` The number of PPMC samples to be simulated.

`seed` The random number seed. Defaults to the seed set in the blatent model object.

`parallel` If parallelization should be used in PPMC. Defaults to "`TRUE`".

`nCores` If "`parallel == TRUE`", then specifies the number of cores to use. Defaults to four.

`type` The type of statistic to generate, submitted as a character vector. Options include:

- "`mean`" computes and tabulates the mean for the posterior simulated data for each observed variable.
- "`covariance`" computes and tabulates the covariance for the posterior simulated data for all pairs of observed variables.
- "`univariate`" computes and tabulates a Pearson Chi-Square comparing the counts for an observed variable with the counts for a variable from the posterior simulated data, for each observed variable.
- "`bivariate`"computes and tabulates a Pearson Chi-Square comparing the counts for a pair of observed variables with the counts for a pair of variables from the posterior simulated data, for each pair of observed variables.
- "`tetrachoric`" computes and tabulates the tetrachoric correlation for the posterior simulated data for all pairs of observed variables.
- "`pearson`" computes and tabulates the Pearson correlation for the posterior simulated data for all pairs of observed variables.

`lowPPMCpercentile`
A vector of the lower bound percentiles used for flagging statistics against PPMC predictive distributions. Results are flagged if the observed statistics percentile is lower than the number in the vector. Provided in order of each term in "`type`". Defaults to "`c(.025, .025, 0, 0, .025, .025)`".

`highPPMCpercentile`
A vector of the upper bound percentiles used for flagging statistics against PPMC predictive distributions. Results are flagged if the observed statistics percentile is higher than the number in the vector. Provided in order of each term in "`type`". Defaults to "`c(.975, .975, 1, 1, .975, .975)`".

blatentSimulate *Simulates data using blatent syntax and simulated parameters input*

Description

Simulates data from a model specified by blatent syntax and using a set of default parameter specifications.

Usage

```
blatentSimulate(
  modelText,
  nObs,
  defaultSimulatedParameters = setDefaultSimulatedParameters(),
  paramVals = NULL,
  seed = NULL,
  calculateInfo = FALSE
)
```

Arguments

`modelText` A character string that contains the specifications for the model to be run. See `blatentSyntax` or more information about syntax formatting.

`nObs` The number of observations to be simulated.

`defaultSimulatedParameters` The specifications for the generation of the types of parameters in the simulation. Currently comprised of a list of unevaluated expressions (encapsulated in quotation marks; not calls for ease of user input) that will be evaluated by simulation function to generate parameters. Defaults to values generated by `setDefaultSimulatedParameters`. The list of unevaluated expressions must include:

- `observedIntercepts` The data generating function for all intercepts for observed variables.
- `observedMainEffects` The data generating function for the main effects for observed variables.
- `observedInteractions` The data generating function for all interactions for observed variables.
- `latentIntercepts` The data generating function for all intercepts for latent variables.
- `latentMainEffects` The data generating function for the main effects for latent variables.
- `latentInteractions` The data generating function for all interactions for latent variables.

`paramVals` A named vector of parameter values which will be set rather than generated. A named vector of the length parameters of an analysis can be obtained by using `createParameterVector`. The NA values of this vector can be overwritten by values to be used in the simulation.

`seed` The random number seed value used for setting the data. Defaults to `NULL`.

`calculateInfo` A logical variable where `TRUE` indicates information statistics will be calculated (only when a single latent multivariate Bernoulli variable is in the model) and `FALSE` disables calculation.

References

Rupp, A. A., Templin, J., & Henson, R. A. (2010). Diagnostic Measurement: Theory, Methods, and Applications. New York: Guilford.

Examples

```
# Generating data using Q-matrix structure from data example in Chapter 9 of
#  Rupp, Templin, & Henson (2010).

RTHCh9ModelSyntax = "
   item1 ~ A1
   item2 ~ A2
   item3 ~ A3
   item4 ~ A1 + A2 + A1:A2
   item5 ~ A1 + A3 + A1:A3
   item6 ~ A2 + A3 + A2:A3
   item7 ~ A1 + A2 + A3 + A1:A2 + A1:A3 + A2:A3 + A1:A2:A3

   # Latent Variable Specifications:
 A1 A2 A3 <- latent(unit='rows',distribution='bernoulli',structure='univariate',type='ordinal')

   # Observed Variable Specifications:
   item1-item7 <- observed(distribution = 'bernoulli', link = 'probit')
"

simSpecs = setDefaultSimulatedParameters(
   observedIntercepts = "runif(n = 1, min = -1, max = -1)",
   observedMainEffects = "runif(n = 1, min = 2, max = 2)",
   observedInteractions = "runif(n = 1, min = 0, max = 0)",
   latentIntercepts = "runif(n = 1, min = 0, max = 0)",
   latentMainEffects  = "runif(n = 1, min = 0, max = 0)",
   latentInteractions = "runif(n = 1, min = 0, max = 0)"
 )

simulatedData = blatentSimulate(modelText = RTHCh9ModelSyntax, nObs = 1000,
                                defaultSimulatedParameters = simSpecs)

# setting values for specific parameters:
paramVals = createParameterVector(modelText = RTHCh9ModelSyntax)
paramVals["item1.(Intercept)"] = -2
```

```
# creating data
simulatedData2 = blatentSimulate(modelText = RTHCh9ModelSyntax, nObs = 1000,
                                 defaultSimulatedParameters = simSpecs, paramVals = paramVals)
```

`blatentSyntax` *Syntax specifications for blatent*

Description

The blatent model syntax provides the specifications for a Bayesian latent variable model.

Details

The model syntax, encapsulated in quotation marks, consists of up to three components:

1. ***Model Formulae:*** R model-like formulae specifying the model for all observed and latent variables in the model. See `formula` for R formula specifics. Blatent model formulae differ only in that more than one variable can be provided to the left of the ~.

   In this section of syntax, there are no differences between latent and observed variables. Model statements are formed using the linear predictor for each variable. This means that to specify a measurement model, the latent variables will appear to the right-hand side of the ~.

   Examples:

   - Measurement model where one latent variable (LV) predicts ten items (item1-item10, implying item1, item2, ..., item10):
     `item1-item10 ~ LV`
   - One observed variable (X) predicting another observed variable (Y):
     `Y ~ X`
   - Two items (itemA and itemB) measuring two latent variables (LV1, LV2) with a latent variable interaction:
     `itemA itemB ~ LV1 + LV2 + LV1:LV2`
   - Two items (itemA and itemB) measuring two latent variables (LV1, LV2) with a latent variable interaction (R `formula` shorthand):
     `itemA itemB ~ LV1*LV2`
   - Measurement model with seven items (item1-item7) measuring three latent variables (A1, A2, A3) from Chapter 9 of Rupp, Templin, Henson (2010):
     `item1 ~ A1`
     `item2 ~ A2`
     `item3 ~ A3`
     `item4 ~ A1 + A2 + A1:A2`
     `item5 ~ A1 + A3 + A1:A3`
     `item6 ~ A2 + A3 + A2:A3`
     `item7 ~ A1 + A2 + A3 + A1:A2 + A1:A3 + A2:A3 + A1:A2:A3`

2. ***Latent Variable Specifications:*** Latent variables are declared using a unevaluated function call to the `latent` function. Here, only the latent variables are declared along with options for their estimation. See `latent` for more information.

   `A1 A2 A3 <- latent(unit = 'rows', distribution = 'mvbernoulli', structure = 'joint', type = 'ordinal', jointName = 'class')`

   Additionally, blatent currently uses a Bayesian Inference Network style of specifying the distributional associations between latent variables: Model statements must be given to specify any associations between latent variables. By default, all latent variables are independent, which is a terrible assumption. To fix this, for instance, as shown in Hu and Templin (2020), the following syntax will give a model that is equivalent to the saturated model for a DCM:

   ```
   # Structural Model
   A1 ~ 1
   A2 ~ A1
   A3 ~ A1 + A2 + A1:A2
   ```

3. ***Observed Variable Specifications:*** Observed variables are declared using a unevaluated function call to the `observed` function. Here, only the observed variables are declared along with options for their estimation. See `observed` for more information.

   `item1-item7 <- observed(distribution = 'bernoulli', link = 'probit')`

Continuing with the syntax example from above, the full syntax for the model in Chapter 9 of Rupp, Templin, Henson (2010) is:

```
modelText = "
# Measurement Model

item1 ~ A1
item2 ~ A2
item3 ~ A3
item4 ~ A1 + A2 + A1:A2
item5 ~ A1 + A3 + A1:A3
item6 ~ A2 + A3 + A2:A3
item7 ~ A1 + A2 + A3 + A1:A2 + A1:A3 + A2:A3 + A1:A2:A3

# Structural Model
A1 ~ 1
A2 ~ A1
A3 ~ A1 + A2 + A1:A2

A1 A2 A3 <- latent(unit = 'rows', distribution = 'bernoulli', structure = 'univariate', type = 'ordinal'

# Observed Variable Specifications:
item1-item7 <- observed(distribution = 'bernoulli', link = 'probit')
"
```

References

Rupp, A. A., Templin, J., & Henson, R. A. (2010). Diagnostic Measurement: Theory, Methods, and Applications. New York: Guilford.

Hu, B., & Templin, J. (2020). Using diagnostic classification models to validate attribute hierarchies and evaluate model fit in Bayesian networks. Multivariate Behavioral Research. https://doi.org/10.1080/00273171.2019.1632

`calculateDIC` *calculateDIC*

Description

Calculates DIC for a given model using model object specs.

Usage

```
calculateDIC(model)
```

Arguments

`model`	A blatent MCMC model object.

`calculateWAIC` *calculateWAIC*

Description

Calculates WAIC for a given model using model object specs.

Usage

```
calculateWAIC(model)
```

Arguments

`model`	A blatent MCMC model object.

createParameterVector *Creates named numeric vector with parameter names for analysis specified by modelText*

Description

Creates named numeric vector with parameter names for analysis specified by modelText.

Usage

```
createParameterVector(modelText)
```

Arguments

modelText	A character string that contains the specifications for the model to be run. See blatentSyntax or more information about syntax formatting.

Examples

```
# Generating parameters for data using Q-matrix structure from data example in Chapter 9 of
#  Rupp, Templin, & Henson (2010).

RTHCh9ModelSyntax = "
   item1 ~ A1
   item2 ~ A2
   item3 ~ A3
   item4 ~ A1 + A2 + A1:A2
   item5 ~ A1 + A3 + A1:A3
   item6 ~ A2 + A3 + A2:A3
   item7 ~ A1 + A2 + A3 + A1:A2 + A1:A3 + A2:A3 + A1:A2:A3

   # Latent Variable Specifications:
  A1 A2 A3 <- latent(unit='rows',distribution='bernoulli',structure='univariate',type='ordinal')

   # Observed Variable Specifications:
   item1-item7 <- observed(distribution = 'bernoulli', link = 'probit')
"
paramVals = createParameterVector(modelText = RTHCh9ModelSyntax)
```

latent *Declares latent variables in a blatent model*

Description

Used in blatentSyntax to declare latent variables as an unevaluated function call. Sets specifications used in estimation.

Usage

```
latent(
  unit = "rows",
  distribution = "bernoulli",
  structure = "univariate",
  link = "probit",
  type = "ordinal",
  meanIdentification = NULL,
  varianceIdentification = NULL,
  joint = NULL,
  vars = NULL
)
```

Arguments

`unit` Attaches the unit (person) ID number or label to observations in data. Currently only allows "rows" which indicates each row of the data is a separate unit in the model. Defaults to `"rows"`.

`distribution` Specifies the distribution of the latent variable(s) to which the function points. Defaults to `"bernoulli"`. Distributions currently available are:

- `"bernoulli"`: Specifies each variable follows a Bernoulli distribution (structure must be "univariate").
- `"mvbernoulli"`: Specifies that set of variables follow a multivariate Bernoulli distribution (structure must be "joint").

`structure` Specifies the type of distributional structure for the latent variables. Defaults to `"univariate"`. Structures current available are:

- `"univariate"`: Specifies each variable is modeled using a univariate (marginal or conditional) distribution.
- `"joint"`: Specifies that variables are modeled using a joint distribution (distribution must be "mvbernoulli")

`link` Specifies the link function used for any latent variable model where the latent variable is predicted. Defaults to `"probit"`. Link functions currently available are:

- `"probit"`: Uses a probit link function. Available for variables where `distribution = "bernoulli"` only.

`type` Specifies the type of latent variable to be estimated. Defaults to `"ordinal"`. Types currently available are:

- `"ordinal"`: Specifies that latent variables have ordinal categories. Available for variables where `distribution = "bernoulli"` only.

`meanIdentification` Reserved for future use.

`varianceIdentification` Reserved for future use.

`joint` Specifies the name of the joint distribution of latent variables. Defaults to `NULL`. Used only when structure is "joint".

`vars` Reserved for future use.

observed — *Declares observed variables in a blatent model*

Description

Used in blatentSyntax to declare the distribution and link function for observed variables as an unevaluated function call. Sets specifications used in estimation.

Usage

```
observed(distribution = "bernoulli", link = "probit")
```

Arguments

`distribution`	Specifies the distribution of the observed variable(s) to which the function points. Defaults to `"bernoulli"`. Distributions currently available are: • `"bernoulli"`: Specifies each variable follows a Bernoulli distribution.
`link`	Specifies the link function used for any observed variable model where the observed variable is predicted. Defaults to `"probit"`. Link functions currently available are: • `"probit"`: Uses a probit link function. Available for variables where `distribution = "bernoulli"` only.

QmatrixToBlatentSyntax — *Convert a rectangular Q-matrix into blatent model syntax*

Description

Converts a rectangular Q-matrix into blatent model syntax. Q-matrix must have observed variables listed across columns and latent variables listed across rows.

Usage

```
QmatrixToBlatentSyntax(
  Qmatrix,
  observedVariables = "rownames",
  latentVariables = "colnames",
  lvDist = "joint"
)
```

Arguments

`Qmatrix` A data frame or matrix containing a Q-matrix.

`observedVariables`
If `Qmatrix` is data.frame, the variable in the data frame that has the names of the observed variables. Defaults to `"rownames"`, which uses `rownames(Qmatrix)` and works for data.frame or matrix types of `Qmatrix`.

`latentVariables`
A vector of the variable or column names of the latent variables. Defaults to `"colnames"`, which uses `colnames(Qmatrix)` and works for data.frame or matrix types of `Qmatrix`.

`lvDist` A character that indicates the type of latent variable distribution to be used. `"joint"` for joint distributions (multivariate Bernoulli) or `"univariate"` for univariate Bernoulli using BayesNets parameterization. The latter also builds blatent syntax for the BayesNets model terms.

Value

A character vector containing blatent model syntax.

Examples

```
# Example 1: Joint distribution using data.frame
# empty data.frame

exampleQmatrixDF = data.frame(matrix(data = 0, nrow = 10, ncol = 3))

# name columns of Qmatrix

names(exampleQmatrixDF) = c("observedVariableName", "Attribute1", "Attribute2")

# names of observed variables

exampleQmatrixDF[1:10, "observedVariableName"] = paste0("Item",1:10)

# Entries for Qmatrix

exampleQmatrixDF[1:5,"Attribute1"] = 1
exampleQmatrixDF[3:10,"Attribute2"] = 1

# produce blatentSyntax using QmatrixToBlatentSyntax() function

blatentSyntaxJoint = QmatrixToBlatentSyntax(
   Qmatrix = exampleQmatrixDF,
   observedVariables = "observedVariableName",
   latentVariables = c("Attribute1", "Attribute2"),
   lvDist = "joint"
 )
cat(blatentSyntaxJoint)
```

```
# Example 2: Univariate distributions using matrix
# empty data.frame

exampleQmatrixM = matrix(data = 0, nrow = 10, ncol = 2)

# name columns of Qmatrix as latent variable names

colnames(exampleQmatrixM) = c("Attribute1", "Attribute2")

# name rows of Qmatrix as observed variable names

rownames(exampleQmatrixM) = paste0("Item",1:10)

# Entries for Qmatrix

exampleQmatrixM[1:5,"Attribute1"] = 1
exampleQmatrixM[3:10,"Attribute2"] = 1

# produce blatentSyntax using QmatrixToBlatentSyntax() function
#  (with default options for observedVariables and latentVariables)

blatentSyntaxM = QmatrixToBlatentSyntax(Qmatrix = exampleQmatrixM,  lvDist = "univariate")
cat(blatentSyntaxM)
```

`setDefaultInitializeParameters`

Sets the distribution parameters for initializing all parameters

---

Description

All parameters are initialized with distributions using these parameters. Used to quickly set priors for sets of parameters.

Usage

```
setDefaultInitializeParameters(
  normalMean = 0,
  normalVariance = 1,
  normalCovariance = 0,
  dirichletAlpha = 1
)
```

Arguments

`normalMean`	Sets the initialization distribution mean for all parameters with normal distributions. Defaults to `0`.

`normalVariance` Sets the initialization distribution variance for all parameters with normal distributions. Defaults to `10`.

`normalCovariance` Sets the initialization distribution covariance for all parameters with multivariate normal distributions. Defaults to `0`.

`dirichletAlpha` Sets the initialization of the alpha parameters for all parameters with a categorical distribution. Defaults to `1`.

Value

A list containing named values for each argument in the function.

`setDefaultPriors` *Sets the prior distribution parameters for all parameters not named in priorsList*

Description

All parameters not named in priorsList, an input argument to `blatentEstimate`, recieve these parameters if their prior distributions are of the same family. Used to quickly set priors for sets of parameters.

Usage

```
setDefaultPriors(
  normalMean = 0,
  normalVariance = 1,
  normalCovariance = 0,
  dirichletAlpha = 1
)
```

Arguments

`normalMean` Sets the prior distribution mean for all parameters with normal distributions not named in priorsList. Defaults to `0`.

`normalVariance` Sets the prior distribution variance for all parameters with normal distributions not named in priorsList. Defaults to `1000`.

`normalCovariance` Sets the prior distribution covariance for all parameters with multivariate normal distributions not named in priorsList. Defaults to `0`.

`dirichletAlpha` Sets the prior distribution parameter values when variable distributions are Dirichlet. Defaults to `1`.

Value

A list containing named values for each argument in the function.

setDefaultSimulatedParameters

Creates simulation specifications for simulating data in blatent

Description

Sets the specifications for the generation of the types of parameters in the simulation. Currently comprised of a list of unevaluated expressions (encapsulated in quotation marks; not calls for ease of user input) that will be evaluated by simulation function to generate parameters. Input must be in the form of a random number generation function to be called, surrounded by quotation marks.

Usage

```
setDefaultSimulatedParameters(
  observedIntercepts = "runif(n = 1, min = -2, max = 2)",
  observedMainEffects = "runif(n = 1, min = 0, max = 2)",
  observedInteractions = "runif(n = 1, min = -2, max = 2)",
  latentIntercepts = "runif(n = 1, min = -1, max = 1)",
  latentMainEffects = "runif(n = 1, min = -1, max = 1)",
  latentInteractions = "runif(n = 1, min = -0.5, max = 0.5)",
  latentJointMultinomial = "rdirichlet(n = 1, alpha = rep(1,nCategories))"
)
```

Arguments

`observedIntercepts`
: The data generating function for all intercepts for observed variables. Defaults to `"runif(n = 1, min = -2, max = 2)"`.

`observedMainEffects`
: The data generating function for the main effects for observed variables. Defaults to `"runif(n = 1, min = 0, max = 2)"`.

`observedInteractions`
: The data generating function for all interactions for observed variables. Defaults to `"runif(n = 1, min = -2, max = 2)"`.

`latentIntercepts`
: The data generating function for all intercepts for Bernoulli latent variables modeled with univariate structural models. Defaults to `"runif(n = 1, min = -1, max = 1)"`.

`latentMainEffects`
: The data generating function for the main effects for Bernoulli latent variables modeled with univariate structural models. Defaults to `"runif(n = 1, min = -1, max = 1)"`.

`latentInteractions`
: The data generating function for all interactions for Bernoulli latent variables modeled with univariate structural models. Defaults to `"runif(n = 1, min = -0.5, max = 0.5)"`.

`latentJointMultinomial`
: The data generating function for all interactions for multivariate Bernoulli latent variables modeled with joint structural models. Defaults to `"rdirichlet(n = 1, alpha = rep(1,nCategories))"`. Can use variable `nCategories` to replicate a value or provide a numeric atomic vector as input. Will return an error if size of resulting parameter vector is not correct.

`setPosteriorPredictiveCheckOptions`
Posterior Predictive Model Checking Options

Description

Provides a list of posterior predictive model checks to be run following estimation of a blatent model. Currently six types of posterior predictive model checks (PPMCs) are available: univarate: mean and univariate Chi-square statistic, bivariate: covariance, tetrachoric correlation, pearson correlation, and bivariate Chi-square statistic.

Usage

```
setPosteriorPredictiveCheckOptions(
  estimatePPMC = TRUE,
  PPMCsamples = 1000,
  PPMCtypes = c("mean", "covariance", "univariate", "bivariate", "tetrachoric",
    "pearson"),
  lowPPMCpercentile = c(0.025, 0.025, 0, 0, 0.025, 0.025),
  highPPMCpercentile = c(0.975, 0.975, 1, 1, 0.975, 0.975)
)
```

Arguments

`estimatePPMC`
: If `TRUE`, runs all PPMCs listed in `PPMCtypes`. Defaults to `TRUE`.

`PPMCsamples`
: The number of samples from the posterior distribution and simulated PPMC data sets.

`PPMCtypes`
: The type of PPMC tests to conduct. For each test, the statistic listed is calculated on each PPMC-based simulated data set. Comparisons are made with the values of the statistics calculated on the original data set. Currently six PPMC statistics are available:

  - `mean` Calculates the mean of each variable
  - `univariate` Calculates the Pearson Chi-Square for each variable using simulated data as frequency expected and original data as frequency observed
  - `covariance` Calculates the covariance of every pair of variables
  - `pearson` Calculates the Pearson correlation of every pair of variables
  - `tetrachoric` Calculates the tetrachoric correlation of every pair of variables

- `bivariate` Calculates the Pearson Chi-Square for each pair of variables using simulated data as frequency expected and original data as frequency observed

`lowPPMCpercentile`
: A vector of length equal to the length and number of `PPMCtypes` listing the lower percentile limit for the statistic in the observed data to be considered extreme. Defaults to .025 for non-Chi-Square based statistics and 0 for the Chi-Square statistics

`highPPMCpercentile`
: A vector of length equal to the length and number of `PPMCtypes` listing the upper percentile limit for the statistic in the observed data to be considered extreme. Defaults to .975 for non-Chi-Square based statistics and 1 for the Chi-Square statistics

Value

A list of named values containing a logical value for each parameter above.

Index