Package 'eiwild'

July 2, 2014

Type Package

Title Ecological Inference with individual and aggregate data

Description This package allows to use the hybrid Multinomial-Dirichlet-Model of Ecological Inference for estimating inner Cells of RxC-Tables. This was already implemented in the eiPack-package. eiwild-package now has the possibility to use individual level data to support the aggregate level data and using different Hyperpriori-Distributions.

Version 0.6.7

Date 2014-03-04

Author Thomas Schlesinger <tho.schlesinger@gmail.com>

Maintainer Thomas Schlesinger <tho.schlesinger@gmail.com>

Depends R (>= 2.15)

Imports gtools, coda, lattice

License GPL-3

URL http://CRAN.R-project.org/package=eiwild

Repository CRAN

Repository/R-Forge/Project eiwild

Repository/R-Forge/Revision 22

Repository/R-Forge/DateTimeStamp 2014-03-04 17:36:42

Date/Publication 2014-03-06 21:31:14

NeedsCompilation yes

R topics documented:

`alphaCheck` *alphaCheck function*

Description

Given starting values for the alpha values of the dirichlet distribution of the second level of the ecological inference model `alphaCheck` checks if parameters are correctly specified

Usage

```
alphaCheck(mat, r, c)
```

Arguments

`mat`	matrix with alpha values
`r`	number of rows in the RxC-table
`c`	number of cols in the RxC-table

Value

alphamatrix if everything is okay. `Error`-message if something fails.

Examples

```
## Not run:

# right alpha matrix
alphaRight <- matrix(1:9, 3,3)
alphaCheck(alphaRight, 3, 3)
# return is right alpha matrix

# wrong alpha matrix
alphaWrong <- matrix(0:8, 3,3)
alphaCheck(alphaWrong, 3,3)
alphaCheck(alphaRight, 3,4)

## End(Not run)
```

`betaCheck` *betaCheck function*

Description

Given starting values for the beta values on the first level of the ecological inference model, `betaCheck` checks if parameters are correctly specified

Usage

```
betaCheck(arr, r, c, prec)
```

Arguments

`arr`	array with beta values. 1st dimension: rows, 2nd dimension columns, 3rd dimension precincts
`r`	number of rows in the RxC-table
`c`	number of cols in the RxC-table
`prec`	number of precincts

Value

betaarry if everything is okay. `Error`-message if something fails.

Examples

```
## Not run:
# right beta array
beta1 <- rep(c(0,0.25,0.75), each=3)
beta2 <- rep(beta1, 3)
betaRight <- array(beta2, dim=c(3,3,3))
betaCheck(betaRight, 3,3,3)
```

```
# wrong beta array
betaWrong <- array(1:27, dim=c(3,3,3))
betaCheck(betaWrong)

## End(Not run)
```

`calcmfrow` *calculate mfrow*

Description

calculate mfrow

Usage

```
calcmfrow(x)
```

Arguments

`x`	length of certain vector

`calcWhichCell` *calculating cell number*

Description

Calculates Cell numbers for whichCell-parameters in eiwild-plot function

Usage

```
calcWhichCell(rc, whichRow = NULL, whichCol = NULL)
```

Arguments

`rc`	integer vector with 2 elements giving the dimension of the ecological inference table
`whichRow`	integer naming the specified row number
`whichCol`	integer naming the specified column number

Value

returns vector of innerCells

comPlot *Plot Diagnostics to compare eiwild objects*

Description

This function uses plot diagnostics (see plot.eiwild) to compare different eiwild objects using functions of coda package

Usage

```
comPlot(eiList, whichCell, whichPlot, whichParam = "cellCounts",
  rollLim = NULL, rollCol = NULL, ...)
```

Arguments

eiList	list of eiwild objects
whichCell	which cell to plot
whichPlot	which type of plot. see Details for more information
whichParam	which parameter should be plotted "alphaDraws" or "cellCounts"
rollLim	specifying other ylim-values for rolling mean plot (Default=NULL)
rollCol	specifying other col-values for rolling mean plot (Default=NULL)
...	arguments given to corresponding coda function

Details

whichPlot controls the plot diagnostic to run:

- 1 passes arguments to traceplot
- 2 passes arguments to densityplot
- 3 calculates Running Mean with eiwild:::rollMean
- 4 passes arguments to gelman.plot. Output of gelman.diag will be title of this plot.

See Also

mcmc plot.eiwild indAggEi

Examples

```
## Not run:
# loading some fake election data
data(topleveldat)
form <- cbind(CSU_2, SPD_2, LINK_2, GRUN_2) ~ cbind(CSU_1, SPD_1, Link_1)
set.seed(1234)
out1 <- indAggEi(form=form, aggr=aggr, indi=indi, IDCols=c("ID","ID"),
                 sample=1000, thinning=2, burnin=100, verbose=100)
out2 <- indAggEi(form=form, aggr=aggr, indi=indi, IDCols=c("ID","ID"),
```

```
                sample=1000, thinning=2, burnin=100, verbose=100)
out3 <- indAggEi(form=form, aggr=aggr, indi=indi, IDCols=c("ID","ID"),
                sample=1000, thinning=2, burnin=100, verbose=100)
out4 <- indAggEi(form=form, aggr=aggr, indi=indi, IDCols=c("ID","ID"),
                sample=1000, thinning=2, burnin=100, verbose=100)

eiList <- list(out1, out2, out3, out4)

comPlot(eiList, whichCell=1, whichPlot=1)
comPlot(eiList, whichCell="counts.CSU_1.CSU_2", whichPlot=1)
comPlot(eiList, whichCell=1, whichPlot=1, smooth=TRUE)

comPlot(eiList, whichCell=1, whichPlot=2)

comPlot(eiList, whichCell=1, whichPlot=3)

comPlot(eiList, whichCell=1, whichPlot=4)
comPlot(eiList, whichCell=1, whichPlot=4)
comPlot(eiList, 1, 3, whichParam="alphaDraws")

comPlot(eiList, "alpha.CSU_1.CSU_2", 3, whichParam="alphaDraws")

## End(Not run)
```

`convertEiData` *Extracting important values for ecological Inference*

Description

Extracting important values for calculation of the ecological Inference with the `runMBayes`-function

Usage

```
convertEiData(form, aggr, indi = NULL, IDCols = c("ID"))
```

Arguments

`form`	`formula` in this format `cbind(column_1,column_2, ...,column_c)~cbind(row_1,row_2,...,row_`
`aggr`	`data.frame` with aggregate data. One district per line and one column giving one ID per district. (see Details)
`indi`	`data.frame` with individual data. One district per line and one column giving one ID per district. (see Details) If no individual data are present it defaults to `NULL`
`IDCols`	vector of length 2 giving the column-names or numbers of ID column

Details

`indi` is a *districts x [(r*c)+1]* `data.frame` containing one district per line. One column gives the ID of the districts which will be connection to the ID column in the `aggr`-data.frame.

For example a 2x3 ecological Inference problem with `formula` `cbind(col1,col2,col3) ~ cbind(row1,row2)` will have the row format : `[ID,row1.col1, row1.col2, row1.col3, row2.col1, row2.col2,row2.col3]`

It is important that the `formula` names correspond to the exact column number in the *indi*-data.frame.

Value

`list` with components needed for the Metropolis algorithm in `runMBayes`

See Also

`runMBayes`, `mcmc` `tuneVars`

Examples

```
## Not run:
# loading some fake election data
data(topleveldat)
form <- cbind(CSU_2, SPD_2, LINK_2, GRUN_2) ~ cbind(CSU_1, SPD_1, Link_1)

conv <- convertEiData(form=form, aggr=aggr, indi=indi, IDCols=c("ID", "ID"))

str(conv)

## End(Not run)
```

`eiwild` *eiwild*

Description

eiwild means **E***cological* **I***nference* **W***ith* **I***ndividual* **L***evel* **D***ata*.

It enables to estimate the inner cells of an RxC-table with aggregate data with a number of precincts and/or individual level data for a few precincts.

The assumptions are based on *Rosen et al's* (2001) on the Multinomial-Dirichlet- Model and the *Wakefield's* (2004) paper on combining individual level data for 2x2-tables. In the master thesis of Thomas Schlesinger (2013) he expanded the 2x2-case to the RxC-case and implemented it in this package. Some of the functions are based on the equivalent functions of the *eiPack*-package.

It is a hierarchical Bayesian model which uses MCMC-Algorithms to calculate the estimations. Therefore it has all the pros and cons of these methods.

The typical workflow consists of

1. Formatting your data in the accepted way. Loading `topleveldat` and inspecting the `data.frame`'s `aggr` and `indi` will give you a general idea.

2. Running one or multiple times:
   (a) function tuneVars
   (b) function indAggEi
3. Analyzing results with plot.eiwild, comPlot, summary.eiwild, plotResult

For more examples look at the appropriate functions

Author(s)

Thomas Schlesinger

References

«to be added»

`getBalance` *get profit and loss of partys*

Description

get profit and loss of partys

Usage

```
getBalance(x, rnd = 1, zero = TRUE, which = NULL)
```

Arguments

x	table of ecological inference (see summary.eiwild)
rnd	rounding of values (default is 1)
zero	replace negative values with 0 (default=TRUE)
which	if table isn't square it has to be a vector giving 2 arguments (see Details)

Details

if table isn't square the row or column not to be calculated has to be given in wich.

First element has to be "r" for row or "c" for column.

2nd element has to give name of row or column.

Value

table with balance BUT (!!!) order of rows or cols maybe changed

See Also

summary.eiwild

indAggEi *Ecological Inference including Individual Data*

Description

indAggEi (Individual and Aggregate Ecological Inference) Calculating ecological Inference including Individual data using the two functions convertEiData and runMBayes

Usage

```
indAggEi(form, aggr, indi = NULL, IDCols = c("ID"), whichPriori = "gamma",
  prioriPars = list(shape = 4, rate = 2), startValsAlpha = NULL,
  startValsBeta = NULL, betaVars = NULL, alphaVars = NULL, sample,
  burnin = 0, thinning = 1, verbose = 1, retBeta = FALSE, seed = NULL)
```

Arguments

form	formula in this format cbind(column_1,column_2, ...,column_c)~cbind(row_1,row_2,...,row_
aggr	data.frame with aggregate data. One district per line and one column giving one ID per district. (see Details)
indi	data.frame with individual data. One district per line and one column giving one ID per district. (see Details) If no individual data are present it defaults to NULL
IDCols	vector of length 2 giving the column-names or numbers of ID column
whichPriori	character string specifying the chosen hyperpriori. Options are "gamma" or "expo" (see Details)
prioriPars	vector or matrix of parameters for the specified hyperpriori in *whichPriori*
startValsAlpha	matrix with dimension c(rows,columns) giving the starting values for alpha. If NULL random numbers of rdirichlet with chosen hyperpriori will be chosen.
startValsBeta	array with dimension c(rows,columns,districts) giving the starting values of beta If NULL random multinomial numbers will be chosen.
sample	the sample size to be saved in output. Total length of chain will be *burnin* + *sample* $*$ *thinning*
burnin	number of draws to be cut away from the beginning of the Markov-Chain. default=0
thinning	number specifying the thinning interval. default=1
verbose	an integer specifying whether the progress of the sampler is printed to the screen (defaults to 0). If verbose is greater than 0, the iteration number is printed to the screen every verboseth iteration
betaVars	array-object with dimensions (rows, columns-1, districts) giving variance of proposal density for β-values
alphaVars	matrix of dimensions (rows, columns) giving variance of proposal density for α-values.

retBeta	logical TRUE if estimated β-parameters should be returned. With large number of precincts there can be problems with memory
seed	Default is NULL. Can be given the "seed"-attribute of an eiwild-object to reproduce an eiwild-object

Details

indi is a *districts x [(r*c)+1]* data.frame containing one district per line. One column gives the ID of the districts which will be connection to the ID column in the aggr-data.frame.

For example a 2x3 ecological Inference problem with formula cbind(col1,col2,col3) ~ cbind(row1,row2) will have the row format : [ID,row1.col1, row1.col2, row1.col3, row2.col1, row2.col2,row2.col3]

It is important that the formula names correspond to the exact column number in the *indi*-data.frame.

The *aggr* data.frame can have more columns than the names given in *formula* as long as the colnames exist

The whichPriori-parameter has the options "gamma" or "expo" and corresponding prioriPars-parameters in a "list":

- "expo" and numeric list-element called "lam" corresponding to: $\alpha_{rc} \sim Exp(\lambda)$
- "expo" and matrix list-element called "lam" corresponding to: $\alpha_{rc} \sim Exp(\lambda_{rc})$
- "gamma" and two numeric list-element called "shape" and "rate" corresponding to: $\alpha_{rc} \sim Gamma(\lambda_1, \lambda_2)$
- "gamma" and two matrix list-element called "shape" and "rate" corresponding to: $\alpha_{rc} \sim Gamma(\lambda_1^{rc}, \lambda_2^{rc})$

The "seed" attribute is generated by the .Random.seed-function.

Value

object of class "eiwild" which is a nested list with elements:

- draws
  - alphaDraws mcmc-object
  - cellCounts mcmc-object
  - betaDraws mcmc-object
  - betaAcc "numeric" with Acceptance ratios
  - alphaAcc "numeric" with Acceptance ratios
  - alphaVars matrix with variances for proposal density
  - betaVars array with variances for proposal density
- rowdf original aggregate data
- coldf original aggregate data

An Attribute called "seed" is also saved to reproduce the eiwild-object

See Also

convertEiData, runMBayes, mcmc tuneVars

Examples

```
## Not run:
# loading some fake election data
data(topleveldat)
form <- cbind(CSU_2, SPD_2, LINK_2, GRUN_2) ~ cbind(CSU_1, SPD_1, Link_1)
set.seed(1234)
res <- indAggEi(form=form, aggr=aggr, indi=indi, IDCols=c("ID","ID"),
               sample=1000, thinning=2, burnin=100,verbose=100)
res
summary(res)

# with individual alpha-hyperpriori-parameters
hypMat <- list(shape = matrix(c(30,4,4,4,
                                4,30,4,4,
                                4,4,30,4), nrow=3, ncol=4, byrow=TRUE),
               rate = matrix(c(1,2,2,2,
                               2,1,2,2,
                               2,2,1,2), nrow=3, ncol=4, byrow=TRUE))
set.seed(12345)
res2 <- indAggEi(form=form, aggr=aggr, indi=indi, IDCols=c("ID","ID"),
                 sample=1000, thinning=2, burnin=100, verbose=100,
                 prioriPars=hypMat, whichPriori="gamma")

## End(Not run)
```

`plot.eiwild` *Plot diagnostics for an eiwild Object*

Description

plot diagnostics for an eiwild Object containing autocorrelation plots, density plots and trace using functions of coda package

Usage

```
## S3 method for class 'eiwild'
plot(x, whichPlot = NULL, whichParam = "cellCounts",
  whichCell = NULL, layout = TRUE, ...)
```

Arguments

`x`	an object of class `eiwild`
`whichPlot`	which type of plot. see Details for more information
`whichParam`	which parameter should be plotted `"alphaDraws"` or `"cellCounts"`
`whichCell`	which cell to plot
`layout`	logical if automatic layout of plot should be made with help of `par=mfrow()`
`...`	further graphical parameters given to correspondent coda function

Details

`plot.eiwild` uses the plot diagnostic functions of the coda package. The default is `NULL` which passes arguments to `plot.mcmc`:

- 1 passes arguments to `traceplot`
- 2 passes arguments to `autocorr.plot`
- 3 passes arguments to `densplot`
- 4 calculates and plots rolling mean

See Also

`plot.mcmc` `mcmc`

Examples

```
## Not run:
# loading some fake election data
data(topleveldat)
form <- cbind(CSU_2, SPD_2, LINK_2, GRUN_2) ~ cbind(CSU_1, SPD_1, Link_1)
set.seed(1234)
res <- indAggEi(form=form, aggr=aggr, indi=indi, IDCols=c("ID","ID"),
                sample=1000, thinning=2, burnin=100,verbose=100)

plot(res, whichPlot=1)
plot(res, whichPlot=2)
plot(res, whichPlot=3)
plot(res, whichPlot=4)

plot(res, whichPlot=1, whichCell=c(1,4,5))
plot(res, whichPlot=2, whichCell=c(1,4,5))
plot(res, whichPlot=3, whichCell=c(1,4,5))
plot(res, whichPlot=4, whichCell=c(1,4,5))

plot(res, whichPlot=1, whichCell=c(1))
plot(res, whichPlot=2, whichCell=c(1))
plot(res, whichPlot=3, whichCell=c(1))
plot(res, whichPlot=4, whichCell=c(1))

plot(res, whichPlot=1, whichParam="alphaDraws")
plot(res, whichPlot=2, whichParam="alphaDraws")
plot(res, whichPlot=3, whichParam="alphaDraws")
plot(res, whichPlot=4, whichParam="alphaDraws")

par(mfrow=c(2,2))
plot(res, whichPlot=1, whichCell=1, layout=FALSE)
plot(res, whichPlot=2, whichCell=1, layout=FALSE)
plot(res, whichPlot=3, whichCell=1, layout=FALSE)
plot(res, whichPlot=4, whichCell=1, layout=FALSE)

## End(Not run)
```

plotResult *plots result of the voter transition table*

Description

plots results of the voter transition table. Supports absolute values, relative values and negative values

Usage

```
plotResult(x, abs = FALSE, bgColors = c("white", "steelblue", 10),
  rectOpts = list(border = grey(0.7)), cellOpts = list(cex = 1),
  dimNameOpts = list(col = grey(0.2), cex = 0.8))
```

Arguments

x	matrix with results
abs	TRUE if values are not between [0,1] (Default is FALSE)
bgColors	vector with 3 elements: starting colour of new colour palette, ending colour of new colour palette, length of colour palette
rectOpts	named list of options for rectangle
cellOpts	named list of options for inner cell text
dimNameOpts	named list of options for dimnames

See Also

getBalance

Examples

```
## Not run:
# loading some fake election data
data(topleveldat)
form <- cbind(CSU_2, SPD_2, LINK_2, GRUN_2) ~ cbind(CSU_1, SPD_1, Link_1)
set.seed(1234)
res <- indAggEi(form=form, aggr=aggr, indi=indi, IDCols=c("ID","ID"),
                 sample=1000, thinning=2, burnin=100,verbose=100)
res2 <- indAggEi(form=form, aggr=aggr, indi=indi, IDCols=c("ID","ID"),
                sample=1000, thinning=2, burnin=100,verbose=100)


tabs <- summary(res)
tabs2 <- summary(res2)
plotResult(round(tabs$relative,3))
plotResult(tabs$absolut, abs=TRUE)
bal <- getBalance(tabs$absolut, which=c("c","GRUN_2"))
plotResult(bal, abs=TRUE)
```

```
plotResult(round(tabs$relative,3), bgColors=c("white", "darkorange", 9))
plotResult(round(tabs$relative,3), bgColors=c("white", "darkorange", 5))

plotResult(round(tabs$relative,3) - round(tabs2$relative, 3), abs=TRUE,
           bgColors=c("white", "darkorange", 9))

# ugly ;)
plotResult(round(tabs$relative,3), bgColors=c("blue", "red", 5))
#'
## End(Not run)
```

`prioriPlot` *plots $\beta\hat{}rc$ given hyperpriori parameters*

Description

prioriPlot simulates the $beta_i^r c$ values at first level given specific parameters at hyperpriori level

Usage

```
prioriPlot(pars, which, cols, alphaSample = 10000, betaSample = 300,
  plot = TRUE, ...)
```

Arguments

`pars`	list of parameters for hyperpriori. if which="gamma" then parameter has to be a list with shape and rate as parameters if which="expo" then parameter has to be a list with only lam
`which`	specified priori. "gamma" or "expo"
`cols`	integer specifying how many columns the RxC-Table should have
`alphaSample`	integer specifying the number of times new alpha-values are drawn
`betaSample`	integer specifying the number of times betas will be drawn for each alpha-value
`plot`	logical TRUE/FALSE if histogram should be plotted
`...`	additional arguments for "hist" function

Details

Calculation is made via the marginal beta distribution

function structure:

- `"gamma"` choose one parameter for every alpha_rc-parameter or a two matrices of parameters specifying lambda's for every alpha_rc-parameter
- `"expo"` choose one parameter for every alpha_rc-parameter or a one matrix of parameters specifying lambda's for every alpha_rc-parameter

Value

nested `list` with each element containing another `list`. First level are rows and second level are columns per row.

Examples

```
## Not run:
test1 <- prioriPlot(list(shape=4,rate=2), "gamma",cols=4)
str(test1)

pars <- list(shape=matrix(1:9,3,3),rate=matrix(9:1,3,3))
test2 <- prioriPlot(pars, "gamma",breaks=100)
test3 <- prioriPlot(list(shape=8,rate=2),"gamma",breaks=100,cols=3)

pars4 <- list(shape=matrix(c(6,6,6),1,3), rate=matrix(c(4,4,4),1,3))
test4 <- prioriPlot(pars4, "gamma",breaks=100)

pars5 <- list(lam=2)
test5 <- prioriPlot(pars5, "expo",cols=4, breaks=100)

pars6 <- list(lam=matrix(1:9,3,3)/100)
test6 <- prioriPlot(pars6, "expo", breaks=25, col=grey(0.8))

# example for 3x4-table
set.seed(568)
pars7 <- list(shape=matrix(sample(1:20,12), 3,4), rate=matrix(sample(1:20,12),3,4))
test7 <- prioriPlot(pars7, "gamma",breaks=50)

## End(Not run)
```

`rollMean` *calculate rolling mean up to each iteration*

Description

calculate rolling mean up to each iteration

Usage

```
rollMean(x)
```

Arguments

x	vector

runMBayes *Metropolis Algorithm for ecological Inference*

Description

This function is a wrapper function which calls the appropriate Metropolis algorithm depending on the hyperpriori which is chosen.

Usage

```
runMBayes(convList, whichPriori = "gamma", prioriPars = list(shape = 4, rate
  = 2), startValsAlpha = NULL, startValsBeta = NULL, betaVars = NULL,
  alphaVars = NULL, sample, burnin = 0, thinning = 1, verbose = 1,
  retBeta = FALSE, seed = NULL)
```

Arguments

convList	output of convertEiData-function.
whichPriori	character string specifying the chosen hyperpriori. Options are "gamma" or "expo" (see Details)
prioriPars	vector or matrix of parameters for the specified hyperpriori in *whichPriori*
startValsAlpha	matrix with dimension c(rows,columns) giving the starting values for alpha. If NULL random numbers of rdirichlet with chosen hyperpriori will be chosen.
startValsBeta	array with dimension c(rows,columns,districts) giving the starting values of beta If NULL random multinomial numbers will be chosen.
sample	the sample size to be saved in output. Total length of chain will be *burnin* + *sample* $*$ *thinning*
burnin	number of draws to be cut away from the beginning of the Markov-Chain. default=0
thinning	number specifying the thinning interval. default=1
verbose	an integer specifying whether the progress of the sampler is printed to the screen (defaults to 0). If verbose is greater than 0, the iteration number is printed to the screen every verboseth iteration
betaVars	array-object with dimensions (rows, columns-1, districts) giving variance of proposal density for β-values
alphaVars	matrix of dimensions (rows, columns) giving variance of proposal density for α-values.
retBeta	logical TRUE if estimated β-parameters should be returned. With large number of precincts there can be problems with memory
seed	Default is NULL. Can be given the "seed"-attribute of an eiwild-object to reproduce an eiwild-object

Details

The whichPriori-parameter has the options "gamma" or "expo" and corresponding prioriPars-parameters in a "list":

- "expo" and numeric list-element called "lam" corresponding to: $\alpha_{rc} \sim Exp(\lambda)$
- "expo" and matrix list-element called "lam" corresponding to: $\alpha_{rc} \sim Exp(\lambda_{rc})$
- "gamma" and two numeric list-element called "shape" and "rate" corresponding to: $\alpha_{rc} \sim Gamma(\lambda_1, \lambda_2)$
- "gamma" and two matrix list-element called "shape" and "rate" corresponding to: $\alpha_{rc} \sim Gamma(\lambda_1^{rc}, \lambda_2^{rc})$

The "seed" attribute is generated by the .Random.seed-function.

Value

list-Object with elements:

- alphaDraws mcmc-object
- cellCounts mcmc-object
- betaDraws mcmc-object
- betaAcc "numeric" with Acceptance ratios
- alphaAcc "numeric" with Acceptance ratios
- alphaVars matrix with variances for proposal density
- betaVars array with variances for proposal density

See Also

convertEiData, runMBayes, mcmc tuneVars, indAggEi

Examples

```
## Not run:
# loading some fake election data
data(topleveldat)
form <- cbind(CSU_2, SPD_2, LINK_2, GRUN_2) ~ cbind(CSU_1, SPD_1, Link_1)
conv <- convertEiData(form=form, aggr=aggr, indi=indi, IDCols=c("ID","ID"))
set.seed(1234)
res <- runMBayes(conv, sample=1000, thinning=2, burnin=100,verbose=100)

## !!! not an eiwild object !!!
class(res)

# better to use indAggEi
set.seed(12345)
res2 <- indAggEi(form=form, aggr=aggr, indi=indi, IDCols=c("ID","ID"),
                 sample=1000, thinning=2, burnin=100,verbose=100)
class(res2)
summary(res2)
```

```
# with individual alpha-hyperpriori-parameters
hypMat <- list(shape = matrix(c(30,4,4,4,
                                4,30,4,4,
                                4,4,30,4), nrow=3, ncol=4, byrow=TRUE),
               rate = matrix(c(1,2,2,2,
                               2,1,2,2,
                               2,2,1,2), nrow=3, ncol=4, byrow=TRUE))
set.seed(12345)
res2 <- indAggEi(form=form, aggr=aggr, indi=indi, IDCols=c("ID","ID"),
                 sample=1000, thinning=2, burnin=100, verbose=100,
                 prioriPars=hypMat, whichPriori="gamma")

## End(Not run)
```

---

summary.eiwild *summary method for* eiwild-*object*

Description

summary method for eiwild-object

Usage

```
## S3 method for class 'eiwild'
summary(object, cred = 0.95, ...)
```

Arguments

object	eiwild-object
cred	percentage for credibility interval of alphas and cellCounts
...	no function. included for S3 generic/method consistency

Value

tables and matrices

- relative: global beta values calculated with cellCounts
- absolut: cellCounts mean
- alphaMeans: Means of alphaDraws
- relativeCol: proportions with colSum=1
- countsCred: Credibility Interval of length cred for cellCounts
- alphaCred: Credibility Interval of length cred for alphaDraws
- realtiveCred: Credibility Interval of length cred for global beta values calculated with cellCounts

Examples

```
## Not run:
# loading some fake election data
data(topleveldat)
form <- cbind(CSU_2, SPD_2, LINK_2, GRUN_2) ~ cbind(CSU_1, SPD_1, Link_1)
set.seed(1234)
res <- indAggEi(form=form, aggr=aggr, indi=indi, IDCols=c("ID","ID"),
                sample=1000, thinning=2, burnin=100,verbose=100)

res
sumRes <- summary(res)
sumRes
str(sumRes)

## End(Not run)
```

`topleveldat` *aggregate and individual data of election 3x4-table*

Description

loads 2 `data.frame`-objects called `aggr` and `indi`. `aggr` is a `data.frame` with election results for 3 party's in election one and 4 party's in election 2 for 25 precincts. `indi` is a `data.frame` with voter transitions for 10 precincts for the party's in `aggr`.

Author(s)

Thomas Schlesinger

`trimEiwild` *Trims eiwild-object*

Description

Trims eiwild-object with burn-in and thinning

Usage

```
trimEiwild(obj, burnin = 0, thinning = 1, sample = NULL)
```

Arguments

`obj`	object of type eiwild
`burnin`	number of draws to be cut away from the beginning of the chain. default=0
`thinning`	number specifying the thinning interval. default=1
`sample`	specifies sample size after burn-in and thinning (default is `NULL`)

Value

eiwild-object

Examples

```
## Not run:
# loading some fake election data
data(topleveldat)
form <- cbind(CSU_2, SPD_2, LINK_2, GRUN_2) ~ cbind(CSU_1, SPD_1, Link_1)
set.seed(1234)
res <- indAggEi(form=form, aggr=aggr, indi=indi, IDCols=c("ID","ID"),
                sample=1000, thinning=2, burnin=100,verbose=100)

res
res2<- trimEiwild(res, burnin=100, thinning=3)
res2

## End(Not run)
```

---

tuneVars *Generate Variance parameters for proposal density*

Description

tuneVars generates variance parameters dependent on the acceptance ratio of drawn parameters in the Metropolis algorithm used in indAggEi. The target acceptance rate will be given by parameter accRat

Usage

```
tuneVars(form, aggr, indi = NULL, IDCols = c("ID"), whichPriori = "gamma",
  prioriPars = list(shape = 4, rate = 2), accRat = c(0.4, 0.6),
  minProp = 0.7, maxiter = 20, sample = 10000, verbose = 10000,
  verboseTune = TRUE, improv = NULL, betaVars = NULL, alphaVars = NULL,
  startValsAlpha = NULL, startValsBeta = NULL, seed = NULL)
```

Arguments

form	formula in this format cbind(column_1,column_2, ...,column_c)~cbind(row_1,row_2,...,row_r))
aggr	data.frame with aggregate data. One district per line and one column giving one ID per district. (see Details)
indi	data.frame with individual data. One district per line and one column giving one ID per district. (see Details) If no individual data are present it defaults to NULL
IDCols	vector of length 2 (or 1) giving the columnnames or numbers of ID column
whichPriori	character string defining the hyperpriori. default="gamma"
prioriPars	vector giving the parameters of the hyperpriori

accRat	vector with two elements describing the wished range of the acceptance ratios
minProp	numeric between 0 and 1 describing the percentage of parameters to have the wished acceptance ratios (accRat). maxiter will be the maximum iteration
maxiter	numeric how many times the algorithm should run maximum. If NULL tuning will run until minProp is reached
sample	the sample size to be drawn each tuning run.
verbose	an integer specifying whether the progress of the sampler is printed to the screen (defaults to 0). If verbose is greater than 0, the iteration number is printed to the screen every verboseth iteration
verboseTune	logical if tuning iteration should be printed (default=TRUE)
improv	numeric vector with 2 elements c(a,b). standard deviation will be calculated with the last b percentages of parameters to have the wished acceptance ratio. If standard deviation is lower than a, than tuning is finished. Default is NULL
betaVars	array of dimensions (rows, columns, districts) giving variance of proposal density for betavalues
alphaVars	matrix of dimensions (rows, columns) giving variance of proposal density for alphavalues.
startValsAlpha	matrix with dimension=c(rows,columns) giving the starting values for alpha. If NULL random numbers of rdirichlet with prioriPars will be drawn
startValsBeta	array with dimension=c(rows,columns,districts) giving the starting values of beta If NULL random multinomial numbers with startValsAlpha or prioriPars will be draws
seed	Default is NULL. Can be given the "seed" attribute of an eiwild-object to reproduce an eiwild-object

Details

indi is a districts x [(r*c)+1] data.frame containing one district per line. One column gives the ID of the districts. This will we connected to the ID column in the *aggr*-data.frame. The rest of one line in *indi* is every row beside the nex row. For example a 2x3 ecological Inference problem with *formula* `cbind(col1,col2,col3) ~ cbind(row1,row2)` will have the row format : `[ID, row1.col1, row1.col2,row1.col3, row2.col1, row2.col2, row2.col3]`

It is important that the formula names correspond to the exact column number in the *indi*-data.frame.

The *aggr* data.frame can have more columns than the names given in *formula* as long as the colnames exist

Priorities for finishing of tuning are as follows: If *improv* isn't specified: *minProp* and *maxiter* are checked. If *improv* is specified: 1) *improv* is checked, 2) *minProp* and *maxiter* are checked.

Value

A list containing matrices of variance parameters for the proposal densities

See Also

convertEiData, runMBayes, mcmc tuneVars, indAggEi

Examples

```
## Not run:
data(topleveldat)
out1 <- tuneVars(form=form, aggr=aggr, indi=indi, IDCols=c("ID","ID"),sample=10000, verbose=11000)
out2 <- tuneVars(form=form, aggr=aggr, indi=indi, IDCols=c("ID","ID"), sample=10000, verbose=11000,
                 maxiter=NULL, improv=c(0.01,5))
out3 <- tuneVars(form=form, aggr=aggr, indi=indi, IDCols=c("ID","ID"), sample=10000, verbose=11000,
                 maxiter=NULL, accRat=c(0.45,0.55), improv=c(0.01,5))
str(out3)
out4 <- indAggEi(form=form, aggr=aggr, indi=indi, IDCols=c("ID","ID"),
                 betaVars=out1$betaVars, alphaVars=out1$alphaVars,
                 sample=10000,thinning=1,burnin=100, verbose=1000)
out4

## End(Not run)
```

Index