The BRugs Package

September 21, 2007

Title OpenBUGS and its R / S-PLUS interface BRugs

Version 0.4-1

Date 2007-09-18

Author The Chief Software Bug is Andrew Thomas, with web assistance from Real Bug Bob O'Hara. Other members of the BUGS team are statisticians David Spiegelhalter, Nicky Best, Dave Lunn and Ken Rice. Dave Lunn has also made major contributions to the software development. R Code modified, extended and packaged for R by Uwe Ligges and Sibylle Sturtz. Some ideas taken from the R2WinBUGS package based on code by Andrew Gelman.

Description An R / S-PLUS package containing OpenBUGS and its R / S-PLUS interface BRugs.

Maintainer Uwe Ligges <ligges@statistik.uni-dortmund.de>

Depends R (>= 2.5.0), coda

SystemRequirements currently the only supported OS is Windows, we expect to support Linux in future releases

License GPL version 2

URL http://mathstat.helsinki.fi/openbugs/

R topics documented:

`BRugs` *Introduction to BRugs*

Description

This manual describes how to use the BRugs software

Usage

```
help.BRugs(browser = getOption("browser"))
```

Arguments

`browser`	the name of the program to be used as hypertext browser. It should be in the PATH, or a full path specified.

Details

BRugs is a collection of R functions that allow users to analyze graphical models using MCMC techniques. Most of the R functions in BRugs provide a interface to the BRugs dynamic link library (shared object file). The BRugs dynamic link library is able to make use of many of the WinBUGS components, in particular those components concerned with graphical models and MCMC simulation. BRugs lacks the GUI interface of WinBUGS but is able to use R to create graphical displays of the MCMC simulation. BRugs uses the same model specification language as WinBUGS and the same format for data and initial values. However BRugs always uses plain text files for input inplace of WinBUGS compound documents. The BRugs functions can be split into two groups: those associated with setting up and simulating the graphical model and those associated with making statistical inference. In general the R functions in BRugs correspond to the command buttons and and text entry fields in the menus of WinBUGS. Each WinBUGS text entry field splits into two R functions, one to set the quantity and the other to get the value of the quantity.

Andrew Gelman suggests to use the function `bugs` in the **R2WinBUGS** package with argument `program="openbugs"` as a wrapper.

Permission and Disclaimer

BRugs is released under the GNU GENERAL PUBLIC LICENSE. For details see `http://mathstat.helsinki.fi/openbugs/` or type `help.BRugs()`.

More informally, potential users are reminded to be extremely careful if using this program for serious statistical analysis. We have tested the program on quite a wide set of examples, but be particularly careful with types of model that are currently not featured. If there is a problem, BRugs might just crash, which is not very good, but it might well carry on and produce answers that are wrong, which is even worse. Please let us know of any successes or failures.

See Also

`help.WinBUGS` (which currently is called from `help.BRugs`) and the meta function `BRugsFit`

Examples

```
###    Step by step example:    ###
library(BRugs) # loading BRugs

## Now setting the working directory to the examples' one:
oldwd <- getwd()
setwd(system.file("OpenBUGS", "Examples", package="BRugs"))

## some usual steps (like clicking in WinBUGS):
modelCheck("ratsmodel.txt")          # check model file
modelData("ratsdata.txt")            # read data file
modelCompile(numChains=2)            # compile model with 2 chains
modelInits(rep("ratsinits.txt", 2))  # read init data file
modelUpdate(1000)                    # burn in
samplesSet(c("alpha0", "alpha"))     # alpha0 and alpha should be monitored
modelUpdate(1000)                    # 1000 more iterations ....

samplesStats("*")                    # the summarized results

## some plots
samplesHistory("*", mfrow = c(4, 2)) # plot the chain,
samplesDensity("alpha")              # plot the densities,
samplesBgr("alpha[1:6]")             # plot the bgr statistics, and
samplesAutoC("alpha[1:6]", 1)        # plot autocorrelations of 1st chain

## switch back to the previous working directory:
setwd(oldwd)
## Not run:
# Getting more (online-)help:
if (is.R())
  help.BRugs()
## End(Not run)
```

`BRugsFit` *BRugs' meta function*

Description

This function takes model, data and starting values as input and automatically runs a simulation in BRugs.

Usage

```
BRugsFit(modelFile, data, inits, numChains = 3, parametersToSave,
    nBurnin = 1000, nIter = 1000, nThin = 1,
    DIC = TRUE, working.directory = NULL, digits = 5,
    BRugsVerbose = getOption("BRugsVerbose"))
```

Arguments

`modelFile`	File containing the model written in OpenBUGS code, an R function that contains a BUGS model that is written to a temporary model file (see `tempfile`) using `writeModel`.
`data`	Either a named list (names corresponding to variable names in the `modelFile`) of the data for the OpenBUGS model, *or* a vector or list of the names of the data objects used by the model. In these cases data are written into a file 'data.txt' into the working directory. If a filename of an existing file is given, data are read from that file.
`inits`	A list with `numChains` elements; each element of the list is itself a list of starting values for the OpenBUGS model, *or* a function creating (possibly random) initial values. In these cases inits are written into files 'inits1.txt', ..., 'initsN.txt' into the working directory. If a vector of filenames of existing files is given, inits are read from those files. Alternatively, if `inits` is not specified, initial values are generated by OpenBUGS.
`numChains`	Number of Markov chains (default: 3).
`parametersToSave`	Character vector of the names of the parameters to save which should be monitored.
`nBurnin`	Length of burn in (before `nIter` iterations start).
`nIter`	Number of iterations (without burn in).
`nThin`	Every `nThin`-th iteration of each chain is stored.
`DIC`	Logical, whether to calculate and return the DIC.
`working.directory`	Sets working directory during execution of this function; `data`, `inits` and other files are written to / read from this directory if no other directory is explicitly given in those arguments. If `NULL`, the current working directory is chosen.
`digits`	Number of significant digits used for OpenBUGS input, see `formatC`.
`BRugsVerbose`	Logical, whether BRugs is supposed to be verbose. This can be controlled for the whole BRugs package by by the option 'BRugsVerbose' (see `options`) which is set to `TRUE` by default.

Value

A list containg components

`Stats`	A data frame containing sample statistics. See `samplesStats`.
`DIC`	The DIC statistics, if `DIC=TRUE`, else `NULL`. See `dicStats`.

See Also

`BRugs`, `help.WinBUGS`. Andrew Gelman proposes some `print` and `plot` methods that can be accessed by the `openbugs` (and `bugs`) and `as.bugs.array` functions in the CRAN package **R2WinBUGS**.

Examples

```
BRugsFit(data = "ratsdata.txt", inits = "ratsinits.txt",
    para = c("alpha", "beta"), modelFile = "ratsmodel.txt",
    numChains = 1,
    working.directory = system.file("OpenBUGS", "Examples",
                                    package = "BRugs"))
```

---

bgrPoint *Internal functions (to support plotting the Gelman-Rubin convergence statistic)*

Description

These functions are for internal use only. They support `samplesBgr` and `plotBgr`.

Usage

```
bgrGrid(node, bins = 50)
bgrPoint(node, iteration)
```

Arguments

`node`	Character vector of length 1, name of a variable in the model.
`bins`	Blocksize
`iteration`	Calculated by `bgrGrid`

Note

Intended for internal use only.

See Also

`samplesBgr`, `BRugs`, `help.WinBUGS`

---

buffer *Reading OpenBUGS buffer file*

Description

Reads OpenBUGS buffer file, internally used for intefacing to OpenBUGS.

Usage

```
buffer()
```

Value

Prints the buffer, returns nothing.

See Also

BRugs, help.WinBUGS

---

bugsData *Writing input for OpenBUGS*

Description

Write data file for OpenBUGS.

Usage

```
bugsData(data, fileName = file.path(getwd(), "data.txt"), digits = 5)
```

Arguments

data	either a named list (names corresponding to variable names in the model file) of the data for the OpenBUGS model, *or* a vector or list of the names of the data objects used by the model
fileName	the filename, defaults to 'data.txt' in the current working directory
digits	number of significant digits used for OpenBUGS input, see formatC

Value

Invisibly returns the fileName.

Note

bugsData uses format="E" internally, i.e. you need to pay attention when writing integers containg many significant digits to the data file.

See Also

BRugs

bugsInits *Writing input for OpenBUGS*

Description

Write files containing inits.

Usage

```
bugsInits(inits, numChains = 1, fileName, digits = 5)
```

Arguments

inits	a list with n.chains elements; each element of the list is itself a list of starting values for the OpenBUGS model, *or* a function creating (possibly random) initial values
numChains	number of Markov chains
fileName	the filename(s), one for each chain. Defaults to 'inits1.txt', ..., 'initsN.txt' in the current working directory.
digits	number of significant digits used for OpenBUGS input, see formatC

Value

Invisibly returns the fileName(s).

See Also

BRugs

buildMCMC *Generating mcmc.list objects for package coda*

Description

This functions reads samples from OpenBUGS and converts the results into an object of class mcmc.list that can directly be used by package coda for further analysis.

Usage

```
buildMCMC(node, beg = samplesGetBeg(), end = samplesGetEnd(),
    firstChain = samplesGetFirstChain(),
    lastChain = samplesGetLastChain(), thin = samplesGetThin())
```

Arguments

`node`	Character vector of length 1, name of a variable in the model.
`beg, end`	Arguments to select a slice of monitored values corresponding to iterations `beg:end`.
`firstChain, lastChain`	Arguments to select a sub group of chains.
`thin`	To only use every `thin`-th value of the stored sample.

Details

If the variable of interest is an array, slices of the array can be selected using the notation `variable[lower0:upper0, lower1:upper1, ...]`. A star '`*`' can be entered as shorthand for all the stored samples.

If the arguments are left at their defaults the whole sample for all chains will be used for calculation.

Value

An object of class `mcmc.list` which is a list containing `mcmc` objects.

See Also

`mcmc.list`, `mcmc`, `BRugs`, `help.WinBUGS`

`currentValues` *Last sampled values*

Description

This function returns the current (last sampled) values of a variable.

Usage

```
currentValues(nodeLabel)
```

Arguments

`nodeLabel`	Character vector of length 1, name of a variable in the model.

Value

Vector of the current (last sampled) values of a variable.

See Also

`setValues`, `BRugs`, `help.WinBUGS`

dic *DIC*

Description

These functions are used to evaluate the Deviance Information Criterion.

Usage

```
dicSet()
dicStats()
dicClear()
```

Details

These functions are used to evaluate the Deviance Information Criterion (DIC; Spiegelhalter et al., 2002) and related statistics - these can be used to assess model complexity and compare different models. Most of the examples packaged with OpenBUGS contain an example of their usage.

It is important to note that DIC assumes the posterior mean to be a good estimate of the stochastic parameters. If this is not so, say because of extreme skewness or even bimodality, then DIC may not be appropriate. There are also circumstances, such as with mixture models, in which OpenBUGS will not permit the calculation of DIC and so the menu option is greyed out. Please see help.WinBUGS for restrictions.

Value

dicStats returns a data frame with columns:

Dbar	The posterior mean of the deviance, which is exactly the same as if the node 'deviance' had been monitored. This deviance is defined as -2 * log(likelihood): 'likelihood' is defined as p(y | theta), where y comprises all stochastic nodes given values (i.e. data), and theta comprises the stochastic parents of y - 'stochastic parents' are the stochastic nodes upon which the distribution of y depends, when collapsing over all logical relationships.
Dhat	A point estimate of the deviance (-2 * log(likelihood)) obtained by substituting in the posterior means theta.bar of theta: thus Dhat = -2 * log(p(y | theta.bar)).
pD	The effective number of parameters is given by pD = Dbar - Dhat. Thus pD is the posterior mean of the deviance minus the deviance of the posterior means.
DIC	The Deviance Information Criterion is given by DIC = Dbar + pD = Dhat + 2 * pD. The model with the smallest DIC is estimated to be the model that would best predict a replicate dataset of the same structure as that currently observed.

Note

Users should ensure their simulation has converged before using these functions. If the MCMC simulation has an adaptive phase it will not be possible to make inference using values sampled before the end of this phase.

References

Spiegelhalter, D.J., Best, N.G., Carlin B.P., and van der Linde, A. (2002): Bayesian measures of model complexity and fit (with discussion). *J. Roy. Statist. Soc. B.* 64, 583-640.

See Also

BRugs, help.WinBUGS

---

dimensions *Dimension of BUGS variables*

Description

This function is intended for internal use only.

Usage

```
dimensions(node)
```

Arguments

node	Character vector of length 1, name of a variable in the model.

Value

Dimension of BUGS variable specified by node, if it is a non-scalar one, else NULL.

See Also

BRugs, help.WinBUGS

---

getChain *Current chain to be initialized*

Description

This function is intended for internal use only.

Usage

```
getChain()
```

Value

Number of the chain to be initialized next.

See Also

BRugs, help.WinBUGS

getNumChains *Number of chains*

Description

This function returns the number of chains being simulated.

Usage

```
getNumChains()
```

Value

Returns the number of chains from the current simulation.

See Also

BRugs, help.WinBUGS

help.WinBUGS *WinBUGS documentation*

Description

Function that open the html version of the OpenBUGS manual

Usage

```
help.WinBUGS(browser = getOption("browser"))
```

Arguments

browser	the name of the program to be used as hypertext browser. It should be in the PATH, or a full path specified.

Details

Not yet available in S-PLUS.

See Also

help.BRugs

Examples

```
## Not run:
help.WinBUGS()
## End(Not run)
```

modelAdaptivePhase *Getting length of adaptive phase*

Description

This function returns the length of the adaptive phase of the simulation.

Usage

```
modelAdaptivePhase()
```

Value

This function returns the length of the adaptive phase of the simulation. This is only known after the simulation has finished adapting. If this function is called while the simulation is still adapting `MAX(INTEGER)` is returned. If the simulation does not have an adaptive phase then zero is returned.

Note

This function can be executed once the model has been compiled and initialized.

If an attempt is made to execute this function in an inappropriate context the generic error message 'command is not allowed (greyed out)' is displayed.

See Also

BRugs, help.WinBUGS

modelCheck *Checking the model file*

Description

This function parses a BUGS language description of the statistical model.

Usage

```
modelCheck(fileName)
```

Arguments

fileName	file containing the BUGS language description of the statistical model.

Value

If a syntax error is detected the position of the error and a description of the error is printed, otherwise the 'model is syntaxicaly correct' message is displayed.

Note

If an attempt is made to execute this function in an inappropriate context the generic error message 'command is not allowed (greyed out)' is displayed.

See Also

BRugs, help.WinBUGS

---

modelCompile *Compiling the model*

Description

This function builds the data structures needed to carry out MCMC sampling.

Usage

```
modelCompile(numChains = 1)
```

Arguments

numChains	Simulation is carried out for numChains chains.

Details

The model is checked for completeness and consistency with the data. A node called 'deviance' is automatically created which calculates minus twice the log-likelihood at each iteration, up to a constant. This node can be used like any other node in the graphical model.

Value

When the model has been successfully compiled, 'model compiled' message should be printed.

Note

This command becomes active once the model has been successfully checked (see modelCheck).

If an attempt is made to execute this function in an inappropriate context the generic error message 'command is not allowed (greyed out)' is displayed.

See Also

BRugs, help.WinBUGS

modelData *Loading the data*

Description

This function loads data into the statistical model.

Usage

```
modelData(fileName = "data.txt")
```

Arguments

fileName	Filename(s) of file(s) containing the data in OpenBUGS format.

Value

If any syntax errors or data inconsistencies are detected an error message is displayed. Corrections can be made to the data without returning to the 'check model' stage. When the data have been loaded successfully the message 'data loaded' should appear.

Note

This function can be executed once a model has been successfully checked (see modelCheck), it can no longer be executed once the model has been successfully compiled.

If an attempt is made to execute this function in an inappropriate context the generic error message 'command is not allowed (greyed out)' is displayed.

See Also

BRugs, help.WinBUGS

modelDynamic *Controlling Dynamic Compilation*

Description

These functions are used to enable and disable the feature for dynamic compilation.

Usage

```
modelEnableDynamic()
modelDisableDynamic()
```

See Also

BRugs, help.WinBUGS

modelFactory *Enable and disable factories to create updaters*

Description

These functions enable and disable factories that create updaters.

Usage

```
modelEnable(factory)
modelDisable(factory)
```

Arguments

factory	Character (length 1) name of the factory to be disabled/enabled.

See Also

BRugs, help.WinBUGS

modelGenInits *Generating initial values*

Description

This function attempts to generate initial values by sampling either from the prior or from an approximation to the prior.

Usage

```
modelGenInits()
```

Details

In the case of discrete variables a check is made that a configuration of zero probability is not generated. This function will generate extreme values if any of the priors are very vague.

Value

If the function is successful the message 'initial values generated: model initialized' is displayed otherwise the message 'could not generate initial values' is displayed.

Note

This function can be executed once the model has been successfully compiled (modelCompile), and can no longer be executed once the model has been initialized.

If an attempt is made to execute this function in an inappropriate context the generic error message 'command is not allowed (greyed out)' is displayed.

See Also

BRugs, help.WinBUGS

---

modelInits *Loading initial values*

Description

This function loads initial values for the MCMC simulation.

Usage

```
modelInits(fileName, chainNum = NULL)
```

Arguments

fileName	Character vector of filenames containing the initial values in OpenBUGS format.
chainNum	The initial values will be loaded for the chain number chainNum. By default chainNum is one the first time modelInits is executed and incremented by one after each call modula the number of chains numChains being simulated (and restarts at 1 after that). If fileName is a vector, chainNum is increased automatically by default after processing each file. If there is more than one file containing initial values for one chain, either set chainNum explicitly, or wait until cycle restarts at chain 1.

Details

This function checks that initial values are in the form of an appropriate R object or rectangular array and that they are consistent with any previously loaded data. If some of the elements in an array are known (say because they are constraints in a parameterisation), those elements should be specified as missing (NA) in the initial values file.

Generally it is recommended to load initial values for all fixed effect nodes (founder nodes with no parents) for all chains, initial values for random effects can be generated using the modelGenInits function.

Value

Any syntax errors or inconsistencies in the initial value are displayed. If, after loading the initial values, the model is fully initialized this will be reported by displaying the message 'model initialized'. Otherwise the message 'initial values loaded but this or another chain contain uninitialized variables' will be displayed. The second message can have several meanings:

a) If only one chain is simulated it means that the chain contains some nodes that have not been initialized yet.

b) If several chains are to be simulated it could mean (a) or that no initial values have been loaded for one of the chains.

In either case further initial values can be loaded, or `modelGenInits` can be executed to try and generate initial values for all the uninitialized nodes in all the simulated chains.

Note

This function can be executed once the model has been successfully compiled. It can still be executed once MCMC sampling has been started having the effect of starting the sampler out on a new trajectory.

If an attempt is made to execute this function in an inappropriate context the generic error message 'command is not allowed (greyed out)' is displayed.

See Also

`BRugs`, `help.WinBUGS`

`modelIteration` *Returns number of iterations*

Description

This function returns the total number of iterations carried out divided by thin.

Usage

```
modelIteration()
```

Value

This function returns the total number of iterations carried out divided by thin.

Note

This function can be executed once the model has been compiled and initialized.

If an attempt is made to execute this function in an inappropriate context the generic error message 'command is not allowed (greyed out)' is displayed.

See Also

BRugs, help.WinBUGS

---

modelModules *Loaded modules*

Description

Displays all the modules (dynamic link libraries) in use.

Usage

```
modelModules()
```

Value

Dataframe containing information on all the modules (dynamic link libraries) in use.

See Also

BRugs, help.WinBUGS

---

modelNames *Get variable names in model*

Description

This function returns the names of variables contained in the current model.

Usage

```
modelNames()
```

Value

Character vector of names of variables contained in the current model.

See Also

BRugs, help.WinBUGS

modelPrecision *Setting precision for prec figures*

Description

This function sets the precision to which results are displayed to prec figures.

Usage

```
modelPrecision(prec)
```

Arguments

```
prec
```

Details

It does not affect the precision of any calculations!

See Also

BRugs, help.WinBUGS

modelSaveState *Save the model's current state*

Description

This function saves the sate of each chain in OpenBUGS model

Usage

```
modelSaveState(stem)
```

Arguments

stem	?????.

Value

Note

This function can be executed once a model has been successfully checked ????? (see modelCheck).

If an attempt is made to execute this function in an inappropriate context the generic error message 'command is not allowed (greyed out)' is displayed. ?????

See Also

BRugs, help.WinBUGS

`modelSeed` *Seed of Random Number Generator*

Description

These functions set/return the seed of the random number generator.

Usage

```
modelSetSeed(newSeed)
modelGetSeed(i = 1)
```

Arguments

`newSeed`	a positive, non zero (vector of) integer(s). More than one integer if the chosen random number generator requires more seed components.
`i`	indicates which component of the seed should be returned.

See Also

BRugs, help.WinBUGS

`modelSetAP` *Changing settings of updating algorithms*

Description

These functions change adaptivePhase, iterations, and overRelaxation settings.

Usage

```
modelSetAP(factoryName, adaptivePhase)
modelSetIts(factoryName, iterations)
modelSetOR(factoryName, overRelaxation)
```

Arguments

factoryName	String defining which particular MCMC updating algorithm is to be tuned. Technically this string is the type name of the factory object used to create the updater, for example 'UpdaterMetnormal.Factory' for the random walk metropolis sampler.
adaptivePhase	length of the updater's adaptive phase
iterations	number of times an iterative algorithm is run before a failure is reported
overRelaxation	amount of over relaxation the updater uses

Details

Once a model has been compiled, the various updating algorithms required in order to perform the MCMC simulation may be 'tuned' somewhat via these three functions.

See Also

BRugs, help.WinBUGS

---

modelUpdate *Updating the model*

Description

This function updates the model.

Usage

```
modelUpdate(numUpdates, thin = 1, overRelax = FALSE)
```

Arguments

numUpdates	This function updates the model by carrying out `thin * numUpdates` MCMC iterations for each chain.
thin	The samples from every k*th* iteration will be used for inference, where k is the value of `thin`. Setting `thin > 1` can help to reduce the autocorrelation in the sample, but there is no real advantage in thinning except to reduce storage requirements.
overRelax	If `overRelax` is `TRUE` an over-relaxed form of MCMC (Neal, 1998) which will be executed where possible. This generates multiple samples at each iteration and then selects one that is negatively correlated with the current value. The time per iteration will be increased, but the within-chain correlations should be reduced and hence fewer iterations may be necessary. However, this method is not always effective and should be used with caution. The auto-correlation function may be used to check whether the mixing of the chain is improved.

Note

This function can be executed once the model has been compiled and initialized.

If an attempt is made to execute this function in an inappropriate context the generic error message 'command is not allowed (greyed out)' is displayed.

References

Neal, R. (1998): Suppressing random walks in Markov chain Monte Carlo using ordered overrelaxation. In M.I. Jordan (Ed.): *Learning in Graphical Models*, Kluwer Academic Publishers, Dordrecht, 205-230. http://www.cs.utoronto.ca/~radford/publications.html

See Also

BRugs, help.WinBUGS

---

plotAutoC *Plot autocorrelation function for a scalar variable*

Description

This function plots the autocorrelation function of a scalar variable.

Usage

```
plotAutoC(node, plot = TRUE,
    colour = c("red", "blue", "green", "yellow", "black"),
    lwd = 5, main = NULL, ...)
```

Arguments

node	Character, name of a scalar variable in the model.
plot	Logical, whether to plot the ACF or only return the values. If TRUE, values are returned invisibly.
colour	Colours used to represent different chains.
lwd, main	graphical parameters, see plot.default
...	Further graphical parameters as in par.

Details

Acts on a scalar variable. See the wrapper function samplesAutoC for more details.

Value

An acf object. See acf for details.

See Also

samplesAutoC, acf, BRugs, help.WinBUGS

`plotBgr` *Plot the Gelman-Rubin convergence statistic for a scalar variable*

Description

This function calculates and plots the Gelman-Rubin convergence statistic for a scalar variable, as modified by Brooks and Gelman (1998).

Usage

```
plotBgr(node, plot = TRUE, main = NULL, xlab = "iteration",
    ylab = "bgr", col = c("red", "blue", "green"), bins = 50,
    ...)
```

Arguments

`node`	Character, name of a scalar variable in the model.
`plot`	Logical, whether to plot the BGR statistics or only return the values. If `TRUE`, values are returned invisibly.
`main, xlab, ylab`	annotation, see `plot.default`
`col`	Colours, see Details Section in `samplesBgr`.
`bins`	Number of blocks
`...`	Further graphical parameters as in `par`.

Details

Acts on a scalar variable. See the wrapper function `samplesBgr` for more details.

Value

Data frame with elements

`Iteration`	end iteration of corresponding bin
`pooledChain80pct)`	80pct interval (normalized) of pooled chains
`withinChain80pct`	80pct interval (normalized) of mean within chain
`bgrRatio`	BGR ratio

See Also

`samplesBgr`, `BRugs`, `help.WinBUGS`

plotDensity *Plot density estimate or histogram of a scalar variable*

Description

This function plots a smoothed kernel density estimate for a scalar variable if it is continuous or a histogram if it is discrete.

Usage

```
plotDensity(node, main = NULL, xlab = "" , ylab = "", col = "red", ...)
```

Arguments

node	Character, name of a scalar variable in the model.
main, xlab, ylab, col	graphical parameters, see plot.default
...	Further graphical parameters as in par.

Details

Acts on a scalar variable. See the wrapper function samplesDensity for more details.

See Also

samplesDensity, BRugs, help.WinBUGS

plotHistory *Trace of a scalar variable*

Description

This function returns and plots a complete trace for a scalar variable.

Usage

```
plotHistory(node, plot = TRUE,
    colour = c("red", "blue", "green", "yellow", "black"),
    main = NULL, xlab = "iteration", ylab = "", ...)
```

Arguments

node	Character, name of a scalar variable in the model.
plot	Logical, whether to plot the trace or only return the values. If TRUE, values are returned invisibly.
colour	Colours used to represent different chains.
main, xlab, ylab	graphical parameters, see plot.default
...	Further graphical parameters as in par.

Details

Acts on a scalar variable. See the wrapper function samplesHistory for more details.

Value

A matrix containing samples of node, each row corresponds to one chain.

See Also

samplesHistory, BRugs, help.WinBUGS

---

ranks *Calculation of ranks*

Description

These functions are used to calculate ranks of vector valued quantities in the model.

Usage

```
ranksSet(node)
ranksStats(node)
ranksClear(node)
```

Arguments

node	Character, name of a vector (one dimensional array) variable in the model.

Details

ranksSet creates a monitor that starts building running histograms to represent the rank of each component of node. An amount of storage proportional to the square of the number of components of node is allocated. Even for large numbers of components this can require less storage than calculating the ranks explicitly in the model specification and storing their samples, and it is also much quicker.

ranksStats displays summarises of the distribution of the ranks of each component of node.

ranksClear removes the monitor calculating running histograms for node.

Value

`ranksStats` returns a data frame with columns:

`val2.5pc`	0.025 quantiles
`median`	medians
`val97.5pc`	0.975 quantiles

Note

Users should ensure their simulation has converged before using these functions. Note that if the MCMC simulation has an adaptive phase it will not be possible to make inference using values sampled before the end of this phase.

See Also

`BRugs`, `help.WinBUGS`

`rats` *ratsdata example*

Description

ratsdata example

Usage

```
data(ratsdata)
data(ratsinits)
```

Format

The list `ratsdata` contains data originally taken from section 6 of Gelfand and Smith (1990).

Source

A. Gelfand and A. Smith (1990): Sampling-based Approaches to Calculating Marginal Densities. *Journal of the American Statistical Association*, 85, 398-409.

`samplesAutoC` *Plot autocorrelation function*

Description

This function calculates and plots the autocorrelation function of a variable.

Usage

```
samplesAutoC(node, chain, beg = samplesGetBeg(),
    end = samplesGetEnd(), thin = samplesGetThin(), plot = TRUE,
    mfrow = c(3, 2), ask = NULL, ann = TRUE, ...)
```

Arguments

`node`	Character vector of length 1, name of a variable in the model.
`chain`	Selects a chain to plot autocorrelation function for.
`beg, end`	Arguments to select a slice of monitored values corresponding to iterations `beg:end`.
`thin`	To only use every `thin`-th value of the stored sample for statistics.
`plot`	Logical, whether to plot the ACF or only return the values. If `TRUE`, values are returned invisibly.
`mfrow, ask, ann`	Graphical parameters, see `par` for details. `ask` defaults to `TRUE` unless it is plotting into an already opened non-interactive device. The `ann` parameter is not available in S-PLUS, and will be ignored if it is set.
`...`	Further graphical parameters as in `par` may also be passed as arguments to `plotAutoC`.

Details

If the variable of interest is an array, slices of the array can be selected using the notation `variable[lower0:upper0, lower1:upper1, ...]`. A star '`*`' can be entered as shorthand for all the stored samples.

If the arguments are left at their defaults the whole sample for all chains will be used for calculation.

Value

A list containing `acf` objects - one for each scalar variable contained in argument `node`. See `acf` for details on the list elements.

Note

If the MCMC simulation has an adaptive phase it will not be possible to make inference using values sampled before the end of this phase.

See Also

plotAutoC, acf, BRugs, help.WinBUGS

`samplesBgr` *Plot the Gelman-Rubin convergence statistic*

Description

This function calculates and plots the Gelman-Rubin convergence statistic, as modified by Brooks and Gelman (1998).

Usage

```
samplesBgr(node, beg = samplesGetBeg(), end = samplesGetEnd(),
    firstChain = samplesGetFirstChain(),
    lastChain = samplesGetLastChain(), thin = samplesGetThin(),
    bins = 50, plot = TRUE, mfrow = c(3, 2), ask = NULL,
    ann = TRUE, ...)
```

Arguments

`node`	Character vector of length 1, name of a variable in the model.
`beg, end`	Arguments to select a slice of monitored values corresponding to iterations `beg:end`.
`firstChain, lastChain`	Arguments to select a sub group of chains to calculate the Gelman-Rubin convergence statistics for. Number of chains must be larger than one.
`thin`	Only use every `thin`-th value of the stored sample for statistics.
`bins`	Number of blocks
`plot`	Logical, whether to plot the BGR statistics or only return the values. If `TRUE`, values are returned invisibly.
`mfrow, ask, ann`	Graphical parameters, see `par` for details. `ask` defaults to `TRUE` unless it is plotting into an already opened non-interactive device. The `ann` parameter is not available in S-PLUS, and will be ignored if it is set.
`...`	Further graphical parameters as in `par` may also be passed as arguments to `plotBgr`.

Details

The width of the central 80% interval of the pooled runs is green, the average width of the 80% intervals within the individual runs is blue, and their ratio $R(= pooled/within)$ is red. For plotting purposes the pooled and within interval widths are normalised to have an overall maximum of one. The statistics are calculated in bins of length 50: R would generally be expected to be greater than 1 if the starting values are suitably over-dispersed. Brooks and Gelman (1998) emphasise that one

should be concerned both with convergence of R to 1, and with convergence of both the pooled and within interval widths to stability.

If the variable of interest is an array, slices of the array can be selected using the notation `variable[lower0:upper0, lower1:upper1, ...]`. A star '`*`' can be entered as shorthand for all the stored samples.

If the arguments are left at their defaults the whole sample for all chains will be used for calculation.

Value

A list containing data frames - one for each scalar variable contained in argument `node`. Each data frames contains elements

`Iteration`	end iteration of corresponding bin
`pooledChain80pct)`	80pct interval (normalized) of pooled chains
`withinChain80pct`	80pct interval (normalized) of mean within chain
`bgrRatio`	BGR ratio

Note

If the MCMC simulation has an adaptive phase it will not be possible to make inference using values sampled before the end of this phase.

References

Brooks, S.P. and Gelman A. (1998): Alternative Methods for Monitoring Convergence of Iterative Simulations. *Journal of Computational and Graphical Statistics*, 7, 434-455.

See Also

`plotBgr`, `BRugs`, `help.WinBUGS`

---

samplesClear *Clear recorded values*

Description

This function is used to remove the stored values of a variable.

Usage

```
samplesClear(node)
```

Arguments

`node`	Character vector of length 1, name of a variable in the model.

Details

If the variable of interest is an array, slices of the array can be selected using the notation `variable[lower0:upper0, lower1:upper1, ...]`. A star '`*`' can be entered as shorthand for all the stored samples.

See Also

`BRugs`, `help.WinBUGS`

---

samplesCoda *Writing files in CODA format*

Description

This function writes files in CODA format to be processed or imported, e.g, by some other software.

Usage

```
samplesCoda(node, stem, beg = samplesGetBeg(),
    end = samplesGetEnd(), firstChain = samplesGetFirstChain(),
    lastChain = samplesGetLastChain(), thin = samplesGetThin())
```

Arguments

`node`	Character vector of length 1, name of a variable in the model.
`stem`	The filestem of the CODA files to be generated. See details.
`beg, end`	Arguments to select a slice of monitored values corresponding to iterations `beg:end`.
`firstChain, lastChain`	Arguments to select a sub group of chains.
`thin`	to only use every `thin`-th value of the stored sample.

Details

Example for argument `stem`: If `stem = "c:/myFolder/foo"`, the resulting files are called 'fooCODAchain1.txt', ..., 'fooCODAchainN.txt', and 'fooCODAindex.txt'. They are written into the `tempdir()` and copied to the path '"c:/myFolder"'.

If the variable of interest is an array, slices of the array can be selected using the notation `variable[lower0:upper0, lower1:upper1, ...]`.

If the arguments are left at their defaults the whole sample for all chains will be used for output.

Value

Prints 'CODA files written'.

Note

If the MCMC simulation has an adaptive phase it will not be possible to make inference using values sampled before the end of this phase.

See Also

BRugs, help.WinBUGS

---

samplesCorrel *Correlation*

Description

This function calculates the correlation matrix between two vectors of variables.

Usage

```
samplesCorrel(node0, node1, beg = samplesGetBeg(),
    end = samplesGetEnd(), firstChain = samplesGetFirstChain(),
    lastChain = samplesGetLastChain(), thin = samplesGetThin())
```

Arguments

node0, node1	Character vectors of length 1, name of variables in the model.
beg, end	Arguments to select a slice of monitored values corresponding to iterations beg:end.
firstChain, lastChain	Arguments to select a sub group of chains to calculate correlation(s) for.
thin	to only use every thin-th value of the stored sample for statistics.

Details

If the variable of interest is an array, slices of the array can be selected using the notation variable[lower0:upper0, lower1:upper1, ...].

If the arguments are left at their defaults the whole sample for all chains will be used for calculation.

Value

Correlation matrix.

Note

If the MCMC simulation has an adaptive phase it will not be possible to make inference using values sampled before the end of this phase.

See Also

BRugs, help.WinBUGS

`samplesDensity` *Plot density estimate or histogram*

Description

This function plots a smoothed kernel density estimate for a variable if it is continuous or a histogram if it is discrete.

Usage

```
samplesDensity(node, beg = samplesGetBeg(), end = samplesGetEnd(),
    firstChain = samplesGetFirstChain(),
    lastChain = samplesGetLastChain(), thin = samplesGetThin(),
    mfrow = c(3, 2), ask = NULL, ann = TRUE, ...)
```

Arguments

`node`	Character vector of length 1, name of a variable in the model.
`beg, end`	Arguments to select a slice of monitored values corresponding to iterations `beg:end`.
`firstChain, lastChain`	Arguments to select a sub group of chains to plot density estimate or histogram for.
`thin`	to only use every `thin`-th value of the stored sample for statistics.
`mfrow, ask, ann`	Graphical parameters, see `par` for details. `ask` defaults to `TRUE` unless it is plotting into an already opened non-interactive device. The `ann` parameter is not available in S-PLUS, and will be ignored if it is set.
`...`	Further graphical parameters as in `par` may also be passed as arguments to `plotDensity`.

Details

If the variable of interest is an array, slices of the array can be selected using the notation `variable[lower0:upper0, lower1:upper1, ...]`. A star '`*`' can be entered as shorthand for all the stored samples.

If the arguments are left at their defaults the whole sample for all chains will be used for calculation.

Note

If the MCMC simulation has an adaptive phase it will not be possible to make inference using values sampled before the end of this phase.

See Also

`BRugs`, `help.WinBUGS`

samplesGet *Get settings used for calculations*

Description

These low level functions can be used to get information on settings of begin, end, and thinning of chains, as well as the number of the first/last chain of the stored sample.

Usage

```
samplesGetBeg()
samplesGetEnd()
samplesGetThin()
samplesGetFirstChain()
samplesGetLastChain()
```

Value

`samplesGetBeg` returns the first iteration of the stored sample used for calculating statistics.

`samplesGetEnd` returns the last iteration of the stored sample used for calculating statistics to end.

`samplesGetThin` returns the thin parameter, see `samplesSetThin`.

`samplesGetFirstChain` returns the number of the first chain of the stored sample used for calculating statistics.

`samplesGetLastChain` returns the number of the last chain of the stored sample used for calculating statistics.

See Also

`samplesSetBeg`, `BRugs`, `help.WinBUGS`

samplesHistory *Trace of a variable*

Description

This function returns and plots a complete trace for a variable.

Usage

```
samplesHistory(node, beg = samplesGetBeg(), end = samplesGetEnd(),
    firstChain = samplesGetFirstChain(),
    lastChain = samplesGetLastChain(), thin = samplesGetThin(),
    plot = TRUE, mfrow = c(3, 1), ask = NULL, ann = TRUE, ...)
```

Arguments

node	Character vector of length 1, name of a variable in the model.
beg, end	Arguments to select a slice of monitored values corresponding to iterations beg:end.
firstChain, lastChain	Arguments to select a sub group of chains to plot the trace for.
thin	to only use every thin-th value of the stored sample for statistics.
plot	Logical, whether to plot the trace or only return the values. If TRUE, values are returned invisibly.
mfrow, ask, ann	Graphical parameters, see par for details. ask defaults to TRUE unless it is plotting into an already opened non-interactive device. The ann parameter is not available in S-PLUS, and will be ignored if it is set.
...	Further graphical parameters as in par may also be passed as arguments to plotHistory.

Details

If the variable of interest is an array, slices of the array can be selected using the notation variable[lower0:upper0, lower1:upper1, ...]. A star '*' can be entered as shorthand for all the stored samples.

If the arguments are left at their defaults the whole sample for all chains will be used for calculation.

Value

A list containing matrices - one for each scalar variable contained in argument node. Each row of a matrix corresponds to one chain.

See Also

plotHistory, BRugs, help.WinBUGS

---

samplesMonitors *Names of monitored scalar variables*

Description

This function returns names of monitored scalar variables.

Usage

```
samplesMonitors(node)
```

Arguments

node	Character vector of length 1, name of a variable in the model, or simply '*'. node can be a vector quantity with sub ranges given to indices (e.g. samplesMonitors("node[3:5

Value

A list of names that are monitored. If sampling a vector of parameters of `node`, all elements are printed, e.g.: `"node[beg]", ..., "node[end]"`.

See Also

BRugs, help.WinBUGS

---

samplesSample *Stored values*

Description

This function returns an array of stored values.

Usage

```
samplesSample(node)
```

Arguments

`node`	Character vector of length 1, name of a variable in the model.

Value

Values of the stored sample.

Note

If sampling a vector of parameters, the function must be called for each parameter separately such as `samplesSample(node[1])`.

See Also

BRugs, help.WinBUGS

samplesSet *Start recording*

Description

This function is used to start recording a chain of values for particular variables.

Usage

```
samplesSet(node)
```

Arguments

node	Character vector of names of variables in the model.

Details

WinBUGS generally automatically sets up a logical node to measure a quantity known as deviance; this may be accessed, in the same way as any other variable of interest, by typing its name, i.e. "deviance"

See Also

BRugs, help.WinBUGS

samplesSetting *Change settings used for calculations*

Description

These low level functions can be used to set begin, end, and thinning of chains as well as the first/last chain of the stored sample.

Usage

```
samplesSetBeg(begIt)
samplesSetEnd(endIt)
samplesSetThin(thin)
samplesSetFirstChain(first)
samplesSetLastChain(last)
```

Arguments

`begIt`	First iteration of the stored sample used for calculating statistics.
`endIt`	Last iteration of the stored sample used for calculating statistics.
`thin`	Every `thin`-th iteration of each chain is used to contribute to the statistics being calculated.
`first, last`	First/last chain of the stored sample used for calculating statistics.

Details

`samplesSetBeg` sets the first iteration of the stored sample used for calculating statistics to `begIt`.

`samplesSetEnd` sets the last iteration of the stored sample used for calculating statistics to `endIt`.

`samplesSetThin` sets the numerical field used to select every `thin`-th iteration of each chain to contribute to the statistics being calculated.

`samplesSetFirstChain` is used to set the first chain of the stored sample used for calculating statistics to be `first`.

`samplesSetLastChain` is used to set the last chain of the stored sample used for calculating statistics to be `last`.

Note

Note the difference between this and the thinning facility of the update function: when thinning via the update function we are permanently discarding samples as the MCMC simulation runs, whereas here we have already generated (and stored) a suitable number of (posterior) samples and may wish to discard some of them only temporarily. Thus, setting `thin > 1` here will not have any impact on the storage (memory) requirements; if you wish to reduce the number of samples actually stored (to free-up memory) you should thin via the update function.

See Also

`BRugs`, `help.WinBUGS`

`samplesSize` *Size of the stored sample*

Description

This function returns the size of the stored sample.

Usage

```
samplesSize(node)
```

Arguments

node	Character vector of length 1, name of a variable in the model.

Value

Size of the stored sample. If no samples exist, -1 will be returned.

Note

If sampling a vector of parameters, the function must be called for each parameter separately such as `samplesSize(node[1])`.

See Also

BRugs, help.WinBUGS

---

samplesStats *Calculate summary statistics*

Description

This function produces summary statistics for a variable, pooling over the chains selected.

Usage

```
samplesStats(node, beg = samplesGetBeg(), end = samplesGetEnd(),
    firstChain = samplesGetFirstChain(),
    lastChain = samplesGetLastChain(), thin = samplesGetThin())
```

Arguments

node	Character vector containing names of variables in the model.
beg, end	Arguments to select a slice of monitored values corresponding to iterations beg:end.
firstChain, lastChain	Arguments to select a sub group of chains to calculate summary statistics for.
thin	to only use every thin-th value of the stored sample for statistics.

Details

If the variable of interest is an array, slices of the array can be selected using the notation `variable[lower0:upper0, lower1:upper1, ...]`. A star '*' can be entered as shorthand for all the stored samples.

If the arguments are left at their defaults the whole sample for all chains will be used for calculation.

Value

`samples.stats` returns a data frame with columns:

`mean`	means
`sd`	standard deviations
`MC_error`	Estimate of $s/\sqrt{(N)}$, the Monte Carlo standard error of the mean. The batch means method outlined by Roberts (1996; p.50) is used to estimate s.
`val2.5pc`	0.025 quantiles
`median`	medians
`val97.5pc`	0.975 quantiles
`start`	`beg` + 1
`sample`	sample sizes

Note

If the MCMC simulation has an adaptive phase it will not be possible to make inference using values sampled before the end of this phase.

References

Roberts, G.O. (1996): Markov Chain Concepts Related to Sampling Algorithms. In: W.R. Gilks, S. Richardson and D.J. Spiegelhalter (Eds.): *Markov Chain Monte Carlo in Practice*. Chapman and Hall, London, UK.

See Also

`BRugs`, `help.WinBUGS`

`setValues` *Setting current values*

Description

This function sets current values for a variable for future iterations.

Usage

```
setValues(nodeLabel, values)
```

Arguments

`nodeLabel`	Character vector of length 1, name of a variable in the model.
`values`	The values to be set, generated, e.g., by `currentValues`.

Details

currentValues of a model can be stored in order to be used as initial values.

Value

The number of values set.

See Also

currentValues, BRugs, help.WinBUGS

---

summary *Summary of MCMC simulation*

Description

These functions are used to calculate running means, standard deviations and quantiles.

Usage

```
summarySet(node)
summaryStats(node)
summaryClear(node)
```

Arguments

node	Character vector containing names of a variables in the model.

Details

summarySet creates monitor(s) that starts recording the running totals for node.

summaryStats displays the running means, standard deviations, and 2.5%, 50% (median) and 97.5% quantiles for node. Note that these running quantiles are calculated via an approximate algorithm and should therefore be used with caution.

summaryClear removes the monitor(s) calculating running totals for node.

These functions are less powerful and general than the samples functions (e.g., see samplesSet), but they also require much less storage (an important consideration when many variables and/or long runs are of interest).

Value

summaryStats returns a data frame with columns:

mean	means
sd	standard deviations
val2.5pc	0.025 quantiles

median	medians
val97.5pc	0.975 quantiles
sample	sample sizes

Note

Users should ensure their simulation has converged before using these functions. Note that if the MCMC simulation has an adaptive phase it will not be possible to make inference using values sampled before the end of this phase.

See Also

BRugs, help.WinBUGS

---

write.datafile *Write data for OpenBUGS - intended for internal use only*

Description

Write data in files that can be read by OpenBUGS - intended for internal use only

Usage

```
write.datafile(datalist, towhere, fill = TRUE)
formatdata(datalist)
```

Arguments

datalist	a *list* to be written into an appropriate structure
towhere	the name of the file which the data are to be written to
fill	see cat, defaults to TRUE

Value

datalist.tofile
: A structure appropriate to be read in by OpenBUGS.

See Also

The main functions to be called by the user are bugsData and bugsInits.

`writeModel` *Creating an OpenBUGS model file*

Description

Convert R function to an OpenBUGS model file

Usage

```
writeModel(model, con = "model.txt")
```

Arguments

`model`	R function containg the BUGS model in the BUGS model language, for minor differences see Section Details.
`con`	passed to `link{writeLines}` which actually writes the model file

Details

The fact that bugs models follow closely to S (R) syntax is used. It should be possible to write most BUGS models as R functions.

As a difference, BUGS syntax allows truncation specification like this: `dnorm(...) I(...)` but this is illegal in R. To overcome this incompatibility, use `%_%` before `I(...)`: `dnorm(...) %_% I(...)`. The dummy operator `%_%` will be removed before the BUGS code is saved.

Value

Nothing, but as a side effect, the model file is written.

Author(s)

original idea by Jouni Kerman, modified by Uwe Ligges

See Also

`modelCheck`, `BRugs`

Examples

```
## Same "ratsmodel" that is used in the examples in ?BRugs and ?BRugsFit:
ratsmodel <- function(){
    for(i in 1:N){
        for(j in 1:T){
            Y[i, j] ~ dnorm(mu[i, j],tau.c)
            mu[i, j] <- alpha[i] + beta[i] * (x[j] - xbar)
        }
        alpha[i] ~ dnorm(alpha.c, alpha.tau)
        beta[i] ~ dnorm(beta.c, beta.tau)
```

```
    }
    tau.c ~ dgamma(0.001, 0.001)
    sigma <- 1 / sqrt(tau.c)
    alpha.c ~ dnorm(0.0, 1.0E-6)
    alpha.tau ~ dgamma(0.001, 0.001)
    beta.c ~ dnorm(0.0, 1.0E-6)
    beta.tau ~ dgamma(0.001, 0.001)
    alpha0 <- alpha.c - xbar * beta.c
}

## some temporary filename:
filename <- file.path(tempdir(), "ratsmodel.txt")
## write model file:
writeModel(ratsmodel, filename)
## and let's take a look:
file.show(filename)
```

Index