Package 'runjags'

March 9, 2022

Title Interface Utilities, Model Templates, Parallel Computing Methods and Additional Distributions for MCMC Models in JAGS

Version 2.2.1-5

Date 2022-03-08

Description User-friendly interface utilities for MCMC models via Just Another Gibbs Sampler (JAGS), facilitating the use of parallel (or distributed) processors for multiple chains, automated control of convergence and sample length diagnostics, and evaluation of the performance of a model using drop-k validation or against simulated data. Template model specifications can be generated using a standard lme4-style formula interface to assist users less familiar with the BUGS syntax. A JAGS extension module provides additional distributions including the Pareto family of distributions, the DuMouchel prior and the half-Cauchy prior.

URL https://github.com/ku-awdc/runjags

BugReports https://github.com/ku-awdc/runjags/issues

License GPL-2

Encoding UTF-8

RoxygenNote 7.1.2

SystemRequirements JAGS >= 4.3.0 (https://mcmc-jags.sourceforge.io/)

Depends R (>= 2.14.0)

Imports parallel, lattice (>= 0.20-10), coda (>= 0.17-1), stats, utils

Suggests rjags (>= 4-7), modeest, testthat (>= 3.0.0), knitr, markdown, spelling

Config/testthat/edition 3

VignetteBuilder knitr

Language en-GB

NeedsCompilation yes

Author Matthew Denwood [aut, cre],
Martyn Plummer [cph] (Copyright holder of the code in
src/distributions/DPar1.*, configure.ac, R/rjags_functions.R, and
original copyright holder of some modified code where indicated)

Maintainer Matthew Denwood <md@sund.ku.dk>

Repository CRAN

Date/Publication 2022-03-09 17:10:02 UTC

R topics documented:

`add.summary` *Summary statistics and plot methods for runjags class objects*

Description

Objects of class `runjags-class` have specialised options available for print, plot and summary. These allow various options for controlling how the output is presented, including sub-selection of variables of interest (using partial matching).

Usage

```
add.summary(
  runjags.object,
  vars = NA,
  mutate = NA,
  psrf.target = 1.05,
  normalise.mcmc = TRUE,
  modeest.opts = list(),
  confidence = c(0.95),
  autocorr.lags = c(10),
  custom = NULL,
  silent.jags = runjags.getOption("silent.jags"),
  plots = runjags.getOption("predraw.plots"),
  plot.type = c("trace", "ecdf", "histogram", "autocorr", "key", "crosscorr"),
  col = NA,
  summary.iters = 20000,
  trace.iters = 1000,
  separate.chains = FALSE,
  trace.options = list(),
  density.options = list(),
  histogram.options = list(),
  ecdfplot.options = list(),
  acplot.options = list()
)

## S3 method for class 'runjags'
summary(object, ...)

## S3 method for class 'runjags'
plot(
  x,
  plot.type = c("trace", "ecdf", "histogram", "autocorr", "crosscorr"),
  vars = NA,
  layout = runjags.getOption("plot.layout"),
  new.windows = runjags.getOption("new.windows"),
  file = "",
  mutate = NULL,
  col = NA,
  trace.iters = NA,
  separate.chains = NA,
  trace.options = NA,
  density.options = NA,
  histogram.options = NA,
  ecdfplot.options = NA,
  acplot.options = NA,
  ...
)
```

```
## S3 method for class 'runjags'
print(x, vars = NA, digits = 5, ...)

## S3 method for class 'runjagsplots'
print(
  x,
  layout = runjags.getOption("plot.layout"),
  new.windows = runjags.getOption("new.windows"),
  file = "",
  ...
)

## S3 method for class 'runjagsplots'
plot(
  x,
  layout = runjags.getOption("plot.layout"),
  new.windows = runjags.getOption("new.windows"),
  file = "",
  ...
)
```

Arguments

runjags.object	an object of class runjags-class.
vars	an optional character vector of variable names. If supplied, only variable names in the object supplied with a partial match to anything in 'vars' will be used. Note that regular expressions are not allowed, but the caret (^) token can be used to specify the match at the start of a variable name, and a quoted vars will be matched exactly. Default NA meaning all variables available are returned.
mutate	either a function or a list with first element a function and remaining elements arguments to this function. This can be used to add new variables to the posterior chains that are derived from the directly monitored variables in JAGS. This allows the variables to be summarised or extracted as part of the MCMC objects as if they had been calculated in JAGS, but without the computational or storage overheads associated with calculating them in JAGS directly. The plot, summary and as.mcmc methods for runjags objects will automatically extract the mutated variables along with the directly monitored variables.
psrf.target	the desired cutoff for 'convergence' as determined Gelman and Rubin's convergence diagnostic (see gelman.diag). This is somewhat arbitrary, but 1.05 is a commonly used figure.
normalise.mcmc	an option test transformations of the monitored variable for improved normality, which is an assumption of the Gelman and Rubin statistic. Setting this option to FALSE will likely cause problems with calculating the psrf for highly skewed variables.
modeest.opts	arguments to be passed to the mlv function to calculate the mode of continuous variables. Ignored if the mode.continuous option in runjags.options is set to FALSE.

`confidence` a numeric vector of probabilities (between 0 and 1) on which to base confidence interval calculations.

`autocorr.lags` a numeric vector of integers on which to base the autocorrelation diagnostic. See also the autocorr plot type.

`custom` a custom function which takes a numeric object as input and outputs a single summary statistic. This statistic will be included with the others in the print and summary method outputs.

`silent.jags` option to suppress feedback text produced by the summary function when summary statistics must be recalculated.

`plots` option to pre-draw the plots given by plot.type to facilitate more convinient assessment of convergence after the model has finished running, at the expense of requiring a larger object to stored. The default value uses the option given in `runjags.options`

`plot.type` a character vector of plots to produce, from 'trace', 'density', 'ecdf', 'histogram', 'autocorr', 'crosscorr', 'key' or 'all'. These are all based on the equivalent plots from the `lattice` package with some modifications.

`col` a vector of colours to use for the different chains. This will be used for all plot types (where relevant), including the 'key' plot which functions to label the chain numbers of the various colours. The default uses the standard lattice colour palatte for up to 7 chains, with a rainbow palette used for larger numbers of chains, and combined chains shown in dark grey.

`summary.iters` the number of iterations to thin the chains to before calculating summary statistics (including all plots except the trace plot). Setting too high a value will cause a long delay while calculating these statistics.

`trace.iters` the number of iterations to thin the chains to before producing traceplots. Setting too high a value will cause large file sizes and delays displaying the trace plots.

`separate.chains` option to display each plot separately for different chains (except crosscorr and key). If FALSE, either the separate chains will be shown on the same plot (for trace, density, and ecdf) or as a single plot with combined chains (for histogram and autocorr).

`trace.options` a list of arguments to be passed to the underlying plot function that creates the trace plots. A colour specification should be specified using the 'col' argument above to ensure that this is the same across plot types.

`density.options` a list of arguments to be passed to the underlying plot function that creates the density plots. A colour specification should be specified using the 'col' argument above to ensure that this is the same across plot types.

`histogram.options` a list of arguments to be passed to the underlying plot function that creates the histogram plots. A colour specification should be specified using the 'col' argument above to ensure that this is the same across plot types.

`ecdfplot.options` a list of arguments to be passed to the underlying plot function that creates the ecdf plots. A colour specification should be specified using the 'col' argument above to ensure that this is the same across plot types.

acplot.options	a list of arguments to be passed to the underlying plot function that creates the autocorr plots. A colour specification should be specified using the 'col' argument above to ensure that this is the same across plot types.
object	an object of class runjags-class.
...	additional arguments to be passed to pdf for the plot.runjags method, or the default print method for the print.runjags method.
x	an object of class runjags-class.
layout	the layout of the runjags plots to print, a numeric vector of length 2 stating the number of rows and columns of plots. The default value is taken from runjags.options.
new.windows	option to produce each plot (or matrix of plots) on a new graphics window rather than over-writing the previous plots. For R interfaces where plots can be cycled through (e.g. the OS X GUI and RStudio), it is likely to be preferable to produce all plots to the same device. The default value is taken from runjags.options, which depends on the system.
file	an optional filename to which plots can be saved using pdf. The default "" means produce plots in the active graphics device.
digits	the number of digits to display for printed numerical output.

Details

The print method is designed to display option prettily, wheras the summary method is designed to return the central table (summary statistics for each variable) as a numeric matrix that can be assigned to another variable and manipulated by the user. If summary statistics have been pre-calculated these will be returned without re-calculation by both methods, wheras only the summary method will re-calculate summary statistics if they are not already available.

The add.summary function returns an object of class runjags, with the new summary statistics (and plots if selected) stored internally for future use. Note that many of the summary method options can be passed to run.jags when the model is run and will be remembered for future output, although they can be modified explicitly by subsequent calls to summary or add.summary. If the summary statistics or plots requested are identical to those stored inside the runjags object, they will not be re-calculated. Calculation of the mode of continuous variables is possible, but requires the suggested modeest package.

Value

The summary method returns a numeric matrix of summary statistics for each variable (invisibly for the print method), wheras the add.summary function returns an object of class runjags-class with the new sumamry statistics (and plots if selected) stored for future use. Some summary statistics are only calculated for stochastic variables, but all monitored variables are shown in the output. The information returned as part of the summary is as follows:

- LowerXX The lower confidence limit for the highest posterior density (HPD) credible interval, as calculated by HPDinterval. One or more confidence limits can be selected using the confidence argument - the default of 0.95 corresponds to 95% credible intervals.
- Median The median value, as calculated by median.

- UpperXX The upper confidence limit for the highest posterior density (HPD) credible interval, as calculated by `HPDinterval`. One or more confidence limits can be selected using the confidence argument - the default of 0.95 corresponds to 95% credible intervals.
- Mean The mean value, as calculated by `mean`.
- SD The sample standard deviation, derived from `var`.
- Mode The mode of the variable. For discrete variables this is calculated using `table`, and for continuous variables by `mlv` if this package is installed - see the modeest.opts argument for more details.
- MCerr The Monte Carlo standard error associated with this variable, which is the standard error divided by the square root of the effective sample size as caulculated by `effectiveSize`.
- MC%ofSD The Monte Carlo standard error expressed as a percentage of the standard deviation of the variable - a rule of thumb is that this should be less than approximately 5%.
- SSeff The effective sample size as caulculated by `effectiveSize`. An effective sample size of over 400 should correspond to an MCerr of less than 5% of the sample standard deviation.
- AC.XX The autocorrelation of the sample, as calculated by `autocorr.diag`. One or more lag values can be specified using the autocorr.lags argument - the default is 10 iterations.
- psrf The potential scale reduction factor of the Gelman-Rubin statistic autocorrelation of the sample, as calculated by `autocorr.diag`. This is sometimes referred to as Rhat (or R-hat). Note that any variables marked with a $ sign were stochastic in some chains but not in others - this usually indicates a problem with the model or sampler.

References

Matthew J. Denwood (2016). runjags: An R Package Providing Interface Utilities, Model Templates, Parallel Computing Methods and Additional Distributions for MCMC Models in JAGS. Journal of Statistical Software, 71(9), 1-25. doi:10.18637/jss.v071.i09

See Also

`runjags-class` for details on other methods available for runjags class objects

---

ask *Obtain Input from User With Error Handling*

Description

A simple function to detect input from the user, and keep prompting until a response matching the class of input required is given.

Usage

```
ask(prompt = "?", type = "logical", bounds = c(-Inf, Inf), na.allow = FALSE)
```

Arguments

prompt	what text string should be used to prompt the user? (character string)
type	the class of object expected to be returned - "logical", "numeric", "integer", "character". If the user input does not match this return, the prompt is repeated
bounds	the lower and upper bounds of number to be returned. Ignored if type is "logical" or "character".
na.allow	if TRUE, allows the user to input "NA" for any type, which is returned as NA

See Also

readline and menu

---

autorun.jags — *Run or extend a user-specified Bayesian MCMC model in JAGS with automatically calculated run-length and convergence diagnostics*

Description

Runs or extends a user specified JAGS model from within R, returning an object of class runjags-class. The model is automatically assessed for convergence and adequate sample size before being returned.

Usage

```
autorun.jags(
  model,
  monitor = NA,
  data = NA,
  n.chains = NA,
  inits = NA,
  startburnin = 4000,
  startsample = 10000,
  adapt = 1000,
  datalist = NA,
  initlist = NA,
  jags = runjags.getOption("jagspath"),
  silent.jags = runjags.getOption("silent.jags"),
  modules = runjags.getOption("modules"),
  factories = runjags.getOption("factories"),
  summarise = TRUE,
  mutate = NA,
  thin = 1,
  thin.sample = FALSE,
  raftery.options = list(),
  crash.retry = 1,
  interactive = FALSE,
```

```
  max.time = Inf,
  tempdir = runjags.getOption("tempdir"),
  jags.refresh = 0.1,
  batch.jags = silent.jags,
  method = runjags.getOption("method"),
  method.options = list(),
  ...
)

autoextend.jags(
  runjags.object,
  add.monitor = character(0),
  drop.monitor = character(0),
  drop.chain = numeric(0),
  combine = length(c(add.monitor, drop.monitor, drop.chain)) == 0,
  startburnin = 0,
  startsample = 10000,
  adapt = 1000,
  jags = NA,
  silent.jags = NA,
  summarise = TRUE,
  thin = NA,
  thin.sample = FALSE,
  raftery.options = list(),
  crash.retry = 1,
  interactive = FALSE,
  max.time = Inf,
  tempdir = runjags.getOption("tempdir"),
  jags.refresh = NA,
  batch.jags = NA,
  method = NA,
  method.options = NA,
  ...
)
```

Arguments

model	either a relative or absolute path to a textfile (including the file extension) containing a model in the JAGS language and possibly monitored variable names, data and/or initial values, or a character string of the same. No default. See `read.jagsfile` for more details.
monitor	a character vector of the names of variables to monitor. No default. The special node names 'deviance', 'pd', 'popt', 'dic', 'ped' and 'full.pd' are used to monitor the deviance, mean pD, mean pOpt, DIC, PED and full distribution of sum(pD) respectively. Note that these monitored nodes (with the exception of 'deviance') require multiple chains within the same simulation, and won't appear as variables in the summary statistics or plots (but see `extract` for a way of extracting these from the returned object).

`data` a named list, data frame, environment, character string in the R dump format (see `dump.format`), or a function (with no arguments) returning one of these types. If the model text contains inline #data# comments, then this argument specifies the list, data frame or environment in which to search first for these variables (the global environment is always searched last). If the model text does not contain #data# comments, then the full list or data frame (but not environment) is included as data. If the data argument is a character string, then any #data# comments in the model are ignored (with a warning). The default specifies the parent environment of the function call.

`n.chains` the number of chains to use with the simulation. More chains will improve the sensitivity of the convergence diagnostic, but will cause the simulation to run more slowly (although this may be improved by using a method such as 'parallel', 'rjparallel' or 'snow'). The minimum (and default) number of chains is 2.

`inits` either a character vector with length equal to the number of chains the model will be run using, or a list of named lists representing names and corresponding values of inits for each chain, or a function with either 1 argument representing the chain or no arguments. If a vector, each element of the vector must be a character string in the R dump format representing the initial values for that chain, or NA. If not all initialising variables are specified, the unspecified variables are taken deterministically from the mean or mode of the prior distribution by JAGS. Values left as NA result in all initial values for that chain being taken from the prior distribution. The special variables '.RNG.seed', '.RNG.name', and '.RNG.state' are allowed for explicit control over random number generators in JAGS. If a function is provided, the data is available inside the function as a named list 'data' - this may be useful for setting initial values that depend on the data. Default NA.

`startburnin` the number of burnin iterations, NOT including the adaptive iterations to use for the initial pilot run of the chains.

`startsample` the total number of samples (including the chains supplied in runjags.object for autoextend.jags) on which to assess convergence, with a minimum of 4000. If the runjags.object already contains this number of samples then convergence will be assessed on this object, otherwise the required number of additional samples will be obtained before combining the chains with the old chains. More samples will give a better chance of allowing the chain to converge, but will take longer to achieve. Default 10000 iterations.

`adapt` the number of adaptive iterations to use at the start of each simulation. For the rjags method this adaptation is only performed once and the model remains compiled, unless the repeatable.methods option is activated in `runjags.options`. For all other methods adaptation is done every time the simulation is extended. Default 1000 iterations.

`datalist` deprecated argument.

`initlist` deprecated argument.

`jags` the system call or path for activating JAGS. Default uses the option given in `runjags.options`.

`silent.jags` option to suppress output of the JAGS simulations. Default uses the option given in `runjags.options`.

`modules` a character vector of external modules to be loaded into JAGS, either as the module name on its own or as the module name and status separated by a space, for example 'glm on'.

`factories` a character vector of factory modules to be loaded into JAGS. Factories should be provided in the format '\<facname\> \<factype\> \<status\>' (where status is optional), for example: factories='mix::TemperedMix sampler on'. You must also ensure that any required modules are also specified (in this case 'mix').

`summarise` should summary statistics be automatically calculated for the output chains? Default TRUE (but see also ?runjags.options -> force.summary).

`mutate` either a function or a list with first element a function and remaining elements arguments to this function. This can be used to add new variables to the posterior chains that are derived from the directly monitored variables in JAGS. This allows the variables to be summarised or extracted as part of the MCMC objects as if they had been calculated in JAGS, but without the computational or storage overheads associated with calculating them in JAGS directly. The plot, summary and as.mcmc methods for runjags objects will automatically extract the mutated variables along with the directly monitored variables. For an application to pairwise comparisons of different levels within fixed effects see `contrasts.mcmc`.

`thin` the thinning interval to be used in JAGS. Increasing the thinning interval may reduce autocorrelation, and therefore reduce the number of samples required, but will increase the time required to run the simulation. Using this option thinning is performed directly in JAGS, rather than on an existing MCMC object as with thin.sample. Default 1.

`thin.sample` option to thin the final MCMC chain(s) before calculating summary statistics and returning the chains. Thinning very long chains reduces the size of the returned object. If TRUE, the chain is thinned to as close to a minimum of startsample iterations as possible to ensure the chain length matches thin.sample. A positive integer can also be specified as the desired chain length after thinning; the chains will be thinned to as close to this minimum value as possible. Default TRUE (thinned chains of length startsample returned). This option does NOT carry out thinning in JAGS, therefore R must have enough available memory to hold the chains BEFORE thinning. To avoid this problem use the 'thin' option instead.

`raftery.options`

a named list which is passed as additional arguments to `raftery.diag`, or the logical FALSE to deactivate automatic run length calculation. Default none (default arguments to raftery.diag are used).

`crash.retry` the number of times to re-attempt a simulation if the model returns an error. Default 1 retry (simulation will be aborted after the second crash).

`interactive` option to allow the simulation to be interactive, in which case the user is asked if the simulation should be extended when run length and convergence calculations are performed and the extended simulation will take more than 1 minute. The function will wait for a response before extending the simulations. If FALSE, the simulation will be run until the chains have converged or until the next extension would extend the simulation beyond 'max.time'. Default FALSE.

`max.time` the maximum time for which the function is allowed to extend the chains to improve convergence, as a character string including units or as an integer in which case units are taken as seconds. Ignored if interactive=TRUE. If the function thinks that the next simulation extension to improve convergence will result in a total time of greater than max.time, the extension is aborted. The time per iteration is estimated from the first simulation. Acceptable units include 'seconds', 'minutes', 'hours', 'days', 'weeks', or the first letter(s) of each.

`tempdir` option to use the temporary directory as specified by the system rather than creating files in the working directory. Any files created in the temporary directory are removed when the function exits for any reason. Default TRUE.

`jags.refresh` the refresh interval (in seconds) for monitoring JAGS output using the 'interactive' and 'parallel' methods (see the 'method' argument). Longer refresh intervals will use slightly less processor time, but will make the simulation updates to be shown on the screen less frequently. Reducing the refresh rate to every 10 or 30 seconds may be worthwhile for simulations taking several days to run. Note that this will have no effect on the processor use of the simulations themselves. Default 0.1 seconds.

`batch.jags` option to call JAGS in batch mode, rather than using input redirection. On JAGS >= 3.0.0, this suppresses output of the status which may be useful in some situations. Default TRUE if silent.jags is TRUE, or FALSE otherwise.

`method` the method with which to call JAGS; probably a character vector specifying one of 'rjags', 'simple', 'interruptible', 'parallel', 'rjparallel', or 'snow'. The 'rjags' and 'rjparallel' methods run JAGS using the rjags package, whereas other options do not require the rjags package and call JAGS as an external executable. The advantage of the 'rjags' method is that the model will not need to be recompiled between successive calls to extend.jags, all other methods require a re-compilation (and adaptation if necessary) every time the model is extended. Note that the 'rjparallel' and 'snow' methods may leave behind zombie JAGS processes if the user interrupts the R session used to start the simulations - for this reason the 'parallel' method is recommended for interactive use with parallel chains. The 'parallel' and 'interruptible' methods for Windows require XP Professional, Vista or later (or any Unix-alike). For more information refer to the userguide vignette.

`method.options` a deprecated argument currently permitted for backwards compatibility, but this will be removed from a future version of runjags. Pass these arguments directly to autorun.jags or autoextend.jags.

`...` summary parameters to be passed to `add.summary`, and/or additional options to control some methods including n.sims for parallel methods, cl for rjparallel and snow methods, remote.jags for snow, and by and progress.bar for the rjags method.

`runjags.object` the model to be extended - the output of a run.jags (or autorun.jags or extend.jags etc) function, with class 'runjags'. No default.

`add.monitor` a character vector of variables to add to the monitored variable list. All previously monitored variables are automatically included - although see the 'drop.monitor' argument. Default no additional monitors.

`drop.monitor` a character vector of previously monitored variables to remove from the monitored variable list for the extended model. Default none.

drop.chain	a numeric vector of chains to remove from the extended model. Default none.
combine	a logical flag indicating if results from the new JAGS run should be combined with the previous chains. Default TRUE if not adding or removing variables or chains, and FALSE otherwise.

Details

The autorun.jags function reads, compiles, and updates a JAGS model based on a model representation (plus data, monitors and initial values) input by the user. The autoextend.jags function takes an existing runjags-class object and extends the simulation as required. Chain convergence over the first run of the simulation is assessed using Gelman and Rubin's convergence diagnostic. If necessary, the simulation is extended to improve chain convergence (up to a user-specified maximum time limit), before the required sample size of the Markov chain is calculated using Raftery and Lewis's diagnostic. The simulation is extended to the required sample size dependant on autocorrelation and the number of chains. Note that automated convergence diagnostics are not perfect, and should not be considered as a replacement for manually assessing convergence and Monte Carlo error using the results returned. For more complex models, the use of run.jags directly with manual assessment of necessary run length may be preferable.

For autoextend.jags, any arguments with a default of NA are taken from the runjags object passed to the function.

Value

An object of class 'runjags' (see runjags-class for available methods).

References

Matthew J. Denwood (2016). runjags: An R Package Providing Interface Utilities, Model Templates, Parallel Computing Methods and Additional Distributions for MCMC Models in JAGS. Journal of Statistical Software, 71(9), 1-25. doi:10.18637/jss.v071.i09

See Also

run.jags for fixed run length models, read.winbugs for details of model specification options, read.jagsfile and summary.runjags for details on available methods for the returned models, and run.jags.study for examples of simulation studies using automated model control provided by autorun.jags

Examples

```
# Run a model to calculate the intercept and slope of the expression
# y = m x + c, assuming normal observation errors for y:

# Simulate the data
N <- 100
X <- 1:N
Y <- rnorm(N, 2*X + 10, 1)

# Model in the JAGS format
model <- "model {
```

```
for(i in 1 : N){
Y[i] ~ dnorm(true.y[i], precision)
true.y[i] <- m * X[i] + c
}
m ~ dunif(-1000,1000)
c ~ dunif(-1000,1000)
precision ~ dexp(1)

#data# N, X, Y
#inits# m, c, precision
}"

# Initial values to be used:
m <- list(-10, 10)
c <- list(-10, 10)
precision <- list(0.1, 10)
## Not run:
# Run the model using rjags with a 5 minute timeout:
results <- autorun.jags(model=model, max.time="5m",
monitor=c("m", "c", "precision"), n.chains=2,
method="rjags")

# Analyse standard plots of the results to assess convergence:
plot(results)

# Summary of the monitored variables:
results

# For more details about possible methods see:
vignette('userguide', package='runjags')

## End(Not run)
```

`combine.mcmc` *Combining and dividing runjags and MCMC objects*

Description

Utility functions for combining separate MCMC or runjags objects into a single object, or the reverse operation

Usage

```
combine.mcmc(
  mcmc.objects = list(),
  thin = 1,
  return.samples = NA,
  collapse.chains = if (length(mcmc.objects) == 1) TRUE else FALSE,
  vars = NA,
```

```
  add.mutate = TRUE
)

combine.jags(runjags.objects = list(), summarise = TRUE, ...)

divide.jags(
  runjags.object,
  which.chains = 1:nchain(as.mcmc.list(runjags.object)),
  summarise = TRUE,
  ...
)
```

Arguments

mcmc.objects	a list of MCMC or runjags objects, all with the same number of chains and matching variable names, or a single MCMC object/list or runjags object. No default.
thin	an integer to use to thin the (final) MCMC object by, in addition to any thinning already applied to the objects before being passed to combine.mcmc. Ignored if return.samples is specified (!is.na). Default 1 (no additional thinning is performed).
return.samples	the number of samples to return after thinning. The chains will be thinned to as close to this minimum value as possible, and any excess iterations discarded. Supersedes thin if both are specified. Ignored if niter(mcmc.objects) < return.samples. Default NA.
collapse.chains	option to combine all MCMC chains into a single MCMC chain with more iterations. Can be used for combining chains prior to calculating results in order to reduce the Monte Carlo error of estimates. Default TRUE if a single mcmc.object is provided, or FALSE otherwise.
vars	an optional character vector of variable names to extract. If supplied, only variable names in the object supplied with a partial match to anything in 'vars' will be used. Note that regular expressions are not allowed, but the caret (^) token can be used to specify the match at the start of a variable name, and a quoted vars will be matched exactly. Default NA meaning all variables available are returned.
add.mutate	should any mutate function associated with the runjags objects be run to collect the additional variables before returning MCMC chains?
runjags.objects	a list of runjags class objects to be combined
summarise	option to add a new set of summary statistics to the newly created runjags object
...	other arguments to be passed to add.summary
runjags.object	a single runjags class object to be divided
which.chains	the chains to extract from the runjags object

Details

The combine.mcmc function allows an MCMC object (with 1 or more chains) to be combined with object(s) representing extensions of the same simulation, to produce one MCMC object that contains the continuous combined Markov chains. Alternatively, a single MCMC list object can be converted into a single chain by combining all chains sequentially. An object of class runjags-class can also be used, in which case the MCMC objects will be extracted from this. The combine.jags function does a similar operation, but returning the entire runjags object as a single object that can be extended using extend.jags. The divide.jags extracts one or more chains from a given runjags object.

Value

For combine.mcmc: an MCMC object if collapse.chains=TRUE, or an mcmc.list object if collapse.chains=FALSE

For combine.jags and divide.jags: a runjags-class object

See Also

run.jags and runjags-class

---

dump.list.format *Conversion Between a Named List and a Character String in the R Dump Format*

Description

Convert a named list of numeric vector(s) or array(s) of data or initial values to a character string in the correct format to be read directly by JAGS as either data or initial values.

Usage

```
dump.format(namedlist = list(), checkvalid = TRUE, convertfactors = TRUE)

list.format(data = character(), checkvalid = TRUE)
```

Arguments

namedlist	a named list of numeric or integer (or something that can be coerced to numeric) vectors, matrices or arrays. The name of each list item will be used as the name of the resulting dump.format variables.
checkvalid	option to ensure that the object returned from the function does not contain any values that would be invalid for import into JAGS, such as Inf, -Inf or character values etc.
convertfactors	option to automatically convert any factor variables to numeric (otherwise the presence of factors will create an error if checkvalid==TRUE).
data	a character string in the R dump format, such as that produced by dump.format.

Details

The 'dump.format' function creates a character string of the supplied variables in the same way that dump() would, except that the result is returned as a character string rather than written to file. Additionally, dump.format() will look for any variable with the name '.RNG.name' and double quote the value if not already double quoted (to ensure compatibility with JAGS).

Value

Either a character string in the R dump format (for dump.format), or a named list (for list.format).

See Also

run.jags and dump

Examples

```
# A named list:
namedlist1 <- list(N=10, Count=c(4,2,7,0,6,9,1,4,12,1))
# Convert to a character vector:
chardata <- dump.format(namedlist1)
# And back to a named list:
namedlist2 <- list.format(chardata)
# These should be the same:
stopifnot(identical(namedlist1, namedlist2))
```

---

extract.runjags *Extract peripheral information from runjags objects*

Description

Objects of class 'runjags' are produced by run.jags, results.jags and autorun.jags, and contain the MCMC chains as well as all information required to extend the simulation. This function allows specific information to be extracted from these functions. For other utility methods for the runjags class, see runjags-class.

Usage

```
## S3 method for class 'runjags'
extract(x, what, force.resample = FALSE, ...)
```

Arguments

x	an object of class runjags.
what	the information contained in the runjags object to be extracted. See the details section for the available options.
force.resample	option to re-draw new deviance/DIC/PED etc samples from the model (using dic.samples) rather than using any statistics that may already be available from the saved runjags object
...	additional options to be passed to dic.samples

Details

The supported options for the 'what' argument are as follows:

- crosscorr - the cross-correlation matrix
- summary - the same as the summary method for runjags object
- model - the model
- data - the data
- end.state - the model state at the last iteration (or initial values for non-updated models) which will be used to start an extended simulation
- samplers - a matrix giving the sampler used for stochastic nodes (not available for all models)
- stochastic - a logical vector of length equal to the number of variables indicating which variables are stochastic, with NA values for variables that are stochastic in one chain but not others - the return value of this can be passed to the 'vars' argument for combine.mcmc etc functions
- dic - the DIC, as returned by `dic.samples`
- dic - the PED, as returned by `dic.samples` with type="popt"
- sum.deviance - the sum of the mean estimated deviance for each stochastic variable
- sum.pd - the sum of the mean estimated pD for each stochastic variable
- sum.popt - the sum of the mean estimated pOpt for each stochastic variable
- mean.deviance - the mean estimated pD for each stochastic variable
- mean.pd - the mean estimated pD for each stochastic variable
- mean.popt - the mean estimated pOpt for each stochastic variable
- full.deviance - the sum of the model deviance at each iteration (for each chain)
- full.pd - the sum of the estimated pD at each iteration

Note that for the deviance/DIC related parameters, these will be extracted from the available information if possible, or otherwise re-sampled.

References

Matthew J. Denwood (2016). runjags: An R Package Providing Interface Utilities, Model Templates, Parallel Computing Methods and Additional Distributions for MCMC Models in JAGS. Journal of Statistical Software, 71(9), 1-25. doi:10.18637/jss.v071.i09

See Also

`runjags-class` for additional methods for runjags objects, `add.summary` for details on plot, print and summary methods for runjags class objects, `runjags.options` for general options available, and `run.jags` and `autorun.jags` for the functions that create objects of this class.

findjags *Attempt to Locate a JAGS Install*

Description

Search the most likely locations for JAGS to be installed on the users system, based on the operating system, and return the most likely path to try. Where multiple installs exist, findjags will attempt to return the path to the install with the highest version number. For Unix systems, calling jags using 'jags' requires the jags binary to be in the search path, which may be specified in your user '.Profile' if necessary (the JAGS executable is also looked for in the default install location of /usr/local/bin/jags if popen support is enabled).

Usage

```
findjags(ostype = .Platform$OS.type, look_in = NA, ...)
```

Arguments

ostype	the operating system type. There is probably no reason to want to change this...
look_in	for Windows only, the path to a folder (or vector of folders) which contains another folder with name containing 'JAGS', where the JAGS executable(s) are to be found. findjags() will attempt to find the highest version, assuming that the version number is somewhere in the file path to the executable (as per default installation).
...	provided for compatibility with deprecated arguments.

Value

A path or command for the most likely location of the desired JAGS executable on the system. On unix this will always be 'jags', on Windows for example "C:/Program Files/JAGS/bin/jags-terminal.exe" or "C:/Program Files/JAGS/JAGS-1.0.0/bin/jags-terminal.exe"

See Also

testjags. runjags.options and run.jags

load.runjagsmodule *Load the internal JAGS module provided by runjags*

Description

The runjags package contains a JAGS extension module that provides several additional distributions for use within JAGS (see details below). This function is a simple wrapper to load this module. The version of the module supplied within the runjags package can only be used with the rjags package, or with the rjags or rjparallel methods within runjags. For a standalone JAGS module for use with any JAGS method (or independent JAGS runs) please see: https://sourceforge.net/projects/runjags/files/paretoprior/

Usage

```
load.runjagsmodule(fail = TRUE, silent = FALSE)

unload.runjagsmodule()

load.runJAGSmodule(fail = TRUE, silent = FALSE)

unload.runJAGSmodule()
```

Arguments

fail	should the function fail using stop() if the module cannot be loaded?
silent	if !fail, the function will by default print a diagnostic message if the module cannot be loaded - the silent option suppresses this message.

Details

This module provides the following distributions for JAGS:

PARETO TYPE I: dpar1(alpha, sigma)

$$p(x) = \alpha\sigma^{\alpha}x^{-(\alpha+1)}$$

$$\alpha > 0, \sigma > 0, x > \sigma$$

PARETO TYPE II: dpar2(alpha, sigma, mu)

$$p(x) = \frac{\alpha}{\sigma}\left(\frac{\alpha + x - \mu}{\sigma}\right)^{-(\alpha+1)}$$

$$\alpha > 0, \sigma > 0, x > \mu$$

PARETO TYPE III: dpar3(sigma, mu, gamma)

$$p(x) = \frac{\frac{x-\mu}{\sigma}^{\frac{1}{\gamma}-1}\left(\frac{x-\mu}{\sigma}^{\frac{1}{\gamma}} + 1\right)^{-2}}{\gamma\sigma}$$

$$\sigma > 0, \gamma > 0, x > \mu$$

PARETO TYPE IV: dpar4(alpha, sigma, mu, gamma)

$$p(x) = \frac{\alpha\frac{x-\mu}{\sigma}^{\frac{1}{\gamma}-1}\left(\frac{x-\mu}{\sigma}^{\frac{1}{\gamma}} + 1\right)^{-(\alpha+1)}}{\gamma\sigma}$$

$$\alpha > 0, \sigma > 0, \gamma > 0, x > \mu$$

LOMAX: dlomax(alpha, sigma)

$$p(x) = \frac{\alpha}{\sigma}\left(1 + \frac{x}{\sigma}\right)^{-(\alpha+1)}$$

$$\alpha > 0, \sigma > 0, x > 0$$

GENERALISED PARETO: dgenpar(sigma, mu, xi)

$$p(x) = \frac{1}{\sigma}\left(1 + \xi\left(\frac{x-\mu}{\sigma}\right)\right)^{-\left(\frac{1}{\xi}+1\right)}$$

For $\xi = 0$:

$$p(x) = \frac{1}{\sigma} e^{\frac{-(x-\mu)}{\sigma}}$$

$$\sigma > 0, x > \mu$$

DUMOUCHEL: dmouch(sigma)

$$p(x) = \frac{\sigma}{(x+\sigma)^2}$$

$$\sigma > 0, x > 0$$

HALF CAUCHY: dhalfcauchy(sigma)

$$p(x) = \frac{2\sigma}{\pi\left(x^2 + \sigma^2\right)}$$

$$\sigma > 0, x > 0$$

For an easier to read version of these PDF equations, see the userguide vignette.

Value

Invisibly returns TRUE if able to (un)load the module, or FALSE otherwise

References

Denwood, M.J. 2016. runjags: An R Package Providing Interface Utilities, Model Templates, Parallel Computing Methods and Additional Distributions for MCMC Models in JAGS. J. Stat. Softw. 71. doi:10.18637/jss.v071.i09.

See Also

runjags-class, load.module

Examples

```
# Load the module for use with any rjags model:
available <- load.runjagsmodule(fail=FALSE)
if(available){
# A simple model to sample from a Lomax distribution.
# (Requires the rjags or rjparallel methods)
m <- "model{
 L ~ dlomax(1,1)
}"
## Not run:
results <- run.jags(m, monitor="L", method="rjags", modules="runjags")

## End(Not run)
}
```

---

mutate.functions *Mutate functions to be used with runjags summary methods*

Description

Objects of class runjags-class have specialised options available for print, plot and summary. These methods allow a mutate function to be specified which produces additional variables based on the monitor variables. These functions are examples of valid syntax, and may be useful in their own right.

Usage

```
contrasts.mcmc(x, vars)

prec2sd(x, vars)
```

Arguments

x	an object of class MCMC.
vars	an optional character vector of variable names. If supplied, only variable names in the object supplied with a partial match to anything in 'vars' will be used. Note that regular expressions are not allowed, but the caret (^) token can be used to specify the match at the start of a variable name, and a quoted vars will be matched exactly. Default NA meaning all variables available are returned.

Details

The contrasts.mcmc and prec2sd functions are two common applications of the mutate argument to add.summary and `run.jags` and can be used as examples of the expected inputs and permitted return values. They must take an MCMC object as input, and return an MCMC object or named list with the same length. This can be used to add new variables to the posterior chains that are derived from the directly monitored variables in JAGS. This allows the variables to be summarised or extracted as part of the MCMC objects as if they had been calculated in JAGS, but without the computational or storage overheads associated with calculating them in JAGS directly. The contrasts.mcmc and prec2sd functions are examples of valid objects (but both require an argument, so will have to be passed as e.g. mutate=list('contrasts.mcmc', 'variabletocontrast')). See the mutate argument to `add.summary`.

Value

An MCMC object.

References

Matthew J. Denwood (2016). runjags: An R Package Providing Interface Utilities, Model Templates, Parallel Computing Methods and Additional Distributions for MCMC Models in JAGS. Journal of Statistical Software, 71(9), 1-25. doi:10.18637/jss.v071.i09

See Also

`add.summary` for an applciation of these functions.

`new_unique` *Create a Unique Filename*

Description

Search the current working directory for a file or directory matching the input name, and if it exists suggest a new name by appending a counter to the input name. Alternatively, the function can ask the user if the existing file should be overwritten, in which case the existing file will be erased if the answer is 'yes'. The function also checks for write access permissions at the current working directory.

Usage

```
new_unique(
  name = NA,
  suffix = "",
  ask = FALSE,
  prompt = "A file or directory with this name already exists.  Overwrite?",
  touch = FALSE,
  type = "file"
)
```

Arguments

name	the filename to be used (character string). A vector of character strings is also permissible, in which case they will be pasted together. One or more missing (NA) values can also be used, which will be replaced with a randomly generated 9 character alphanumeric string. Default NA.
suffix	the file extension (including '.') to use (character string). If this does not start with a '.', one will be prepended automatically. Default none.
ask	if a file exists with the input name, should the function ask to overwrite the file? (logical) If FALSE, a new filename is used instead and no files will be over-written. Default FALSE.
prompt	what text string should be used to prompt the user? (character string) Ignored is ask==FALSE. A generic default is supplied.
touch	option to create (touch) the file/folder after generating the unique name, which prevents other processes from sneaking in and creating a file with the same name before the returned filename has had chance to be used. Default FALSE.
type	if touch==TRUE, then type controls if a file or directory is created. One of 'file', 'f', 'directory', or 'd'. Default 'file'.

Value

A unique filename that is safe to use without fear of destroying existing files

See Also

ask

Examples

```
#  Create a unique file name with a .R extension.
new_unique(c("new_file", NA), ".R", ask=FALSE)
```

---

read.jagsfile *Extract Any Models, Data, Monitored Variables or Initial Values As Character Vectors from a JAGS or WinBUGS Format Textfile*

Description

Read a user specified BUGS or JAGS textfile or character variable and extract any models, data, monitored variables or initial values as character vectors. Used by (auto)run.jags to interpret the input file(s) or strings. This function is more likely to be used via run.jags where the model specified to run.jags is the path used by this function. The read.winbugs function is an alias to read.jagsfile.

Usage

```
read.jagsfile(file)

read.JAGSfile(file)

read.winbugs(file)

read.WinBUGS(file)
```

Arguments

file	either a relative or absolute path to a textfile (including the file extension) containing a model in the JAGS language and possibly monitored variable names, data and/or initial values, or a character string of the same. May also be a vector of paths to different text files, possibly separately containing the model, data and intitial values. No default. The model must be started with the string 'model{' and ended with '}' on new lines. Data must be similarly started with 'data{', monitored variables with 'monitor{', and initial values as 'inits{', and all ended with '}'. Seperate variables in such blocks must be separated by a line break. If multiple models are found, all but the first one are ignored with a warning. Multiple data blocks and monitor blocks are combined, multiple inits blocks are used for different chains. Monitors may also be given using the phrase '#monitor# variable' within the model block, in which case 'variable' is added to the list of monitored variables found in the monitor block(s). The use of automatically generated data and initial values is also supported using similar syntax, with '#data# variable' for automatically generated data variables or '#inits# variable' for automatically generated initial value variables in which case 'variable' is used as data or initial values with a value taken by run.jags from datalist, initlist or R objects as appropriate. '#inits#', '#data#' and '#monitor#' statements can appear on the same line as model code, but no more than one of these statements should be used on the same line. Examples of acceptable model syntax are given below.

Details

There are a number of special strings permitted inside the model specification as follows:

#data# variables to be retrieved from a list or environment

#inits# variables to be retrieved from a list or environment

#monitors# monitored variables to use

#modules# JAGS extension modules optionally also specifying the status (e.g. #modules# glm on, dic on)

#factories# JAGS factories and types required, optionally also specifying the status (e.g. #factories# mix::TemperedMix sampler on)

#response# - a single variable name specifying the response variable (optional)

#residual# - a single variable name specifying a variable that represents the residuals (optional)

#fitted# - a single variable name specifying a variable that represents the fitted value (optional)

#Rdata# when placed inside a data or inits block, this signifies that any arrays indside are in column major order. This is the default for any blocks that are not specified as a list( ).

#BUGSdata# when placed inside a data or inits block, this signifies that any arrays indside are in row major order. This is the default for any blocks that are specified as a list( ), such as those that have been created for use with WinBUGS.

#modeldata# when placed inside a data block, this signifies that the code is to be passed to JAGS along with the model block

Value

A named list of elements required to compile a model. These can be used to create a call to run.jags, but it would be more usual to call this function directly on the model file.

References

Lunn D, Jackson C, Best N, Thomas A, Spiegelhalter D (2012). The BUGS book: A practical introduction to Bayesian analysis. CRC press; and Matthew J. Denwood (2016). runjags: An R Package Providing Interface Utilities, Model Templates, Parallel Computing Methods and Additional Distributions for MCMC Models in JAGS. Journal of Statistical Software, 71(9), 1-25. doi:10.18637/jss.v071.i09

See Also

run.jags and write.jagsfile for the reverse operation, and possibly an example of the formatting allowed

Examples

```
# ALL SYNTAX GIVEN BELOW IS EQUIVALENT

# Use a modified WinBUGS text file with manual inits and manual data and
# a seperate monitor block (requires least modification from a WinBUGS
# file).  For compatibility with WinBUGS, the use of list() to enclose
# data and initial values is allowed and ignored, however all seperate
# variables in the data and inits blocks must be seperated with a line
# break (commas or semicolons before linebreaks are ignored).  data{ ... }
# and inits{ ... } must also be added to WinBUGS textfiles so that the
# function can seperate data from initial values.  Iterative loops are
# allowed in data blocks but not in init blocks.  See also the differences
# in JAGS versus WinBUGS syntax in the JAGS help file.

# The examples below are given as character strings for portability,
# but these could also be contained in a separate model file with the
# arguments to read.jagsfile and run.jags specified as the file path


# A model that could be used with WinBUGS, incorporating data and inits.
# A note will be produced that the data and inits are being converted
# from WinBUGS format:

string <- "
```

```
model{

for(i in 1:N){
Count[i] ~ dpois(mean)
}
mean ~ dgamma(0.01, 100)
}

data{
list(
 Count = c(1,2,3,4,5,6,7,8,9,10),
 N = 10
 )
}

inits{
list(
mean = 1
)
}

inits{
list(
mean = 100
)
}

"

model <- read.winbugs(string)
## Not run:
results <- run.jags(string, monitor='mean')

## End(Not run)


# The same model but specified in JAGS format.  This syntax also defines
# monitors in the model, and uses data retrieved from the R environment:

string <- "
model{

for(i in 1:N){
Count[i] ~ dpois(mean) #data# Count, N
}
mean ~ dgamma(0.01, 100)
#monitor# mean
}

inits{
mean <- 1
}
```

```
inits{
mean <- 100
}
"

model <- read.jagsfile(string)
Count <- 1:10
N <- length(Count)
## Not run:
results <- run.jags(string)

## End(Not run)


# The same model using autoinits and a mixture of manual and autodata:
string <- "
model{

for(i in 1:N){
Count[i] ~ dpois(mean) #data# Count
}
mean ~ dgamma(0.01, 100)
#monitor# mean
#inits# mean
}

data{

N <- 10

}
"

model <- read.jagsfile(string)
Count <- 1:10
mean <- list(1, 100)
## Not run:
results <- run.jags(string, n.chains=2)

## End(Not run)
```

`results.jags` *Importing of saved JAGS simulations with partial error recovery*

Description

Imports a completed JAGS simulation from a folder created by `run.jags` using the background or bgparallel methods, or any other method where the keep.jags.files=TRUE option was used. Partial recovery simulations is possible for parallel methods where one or more simulation failed to complete. Additional chain thinning and parameter import selection is also supported.

Usage

```
results.jags(
  foldername,
  echo = NA,
  combine = NA,
  summarise = NA,
  keep.jags.files = NA,
  read.monitor = NA,
  return.samples = NA,
  recover.chains = NA,
  ...
)

results.JAGS(
  foldername,
  echo = NA,
  combine = NA,
  summarise = NA,
  keep.jags.files = NA,
  read.monitor = NA,
  return.samples = NA,
  recover.chains = NA,
  ...
)
```

Arguments

foldername	the absolute or relative path to the folder containing the JAGS simulation to be imported. May also be the return value of a call to run.jags with method = 'background' or method = 'bgparallel', which will avoid having to re-load some information from disk and therefore may be slightly faster.
echo	option to display the output of the simulations to screen. If the simulations have not finished, the progress-to-date will be displayed.
combine	a logical flag indicating if results from the new JAGS run should be combined with the previous chains. Default value respects the setting chosen during the initial run.jags function call, changing the option to TRUE or FALSE overrides the original setting.
summarise	should summary statistics be automatically calculated for the output chains? Default value respects the setting chosen during the initial run.jags function call, changing the option to TRUE or FALSE overrides the original setting.
keep.jags.files	option to keep the folder with files needed to call JAGS, rather than deleting it after importing. Default value respects the setting chosen during the initial run.jags function call, changing the option to TRUE or FALSE overrides the original setting. See also the cleanup.jags function.
read.monitor	an optional character vector of variables to import from the simulation folder. This may be useful for models with large numbers of variables that would oth-

erwise not be able to be loaded into R. Default value loads all variables given by the monitor argument (but NOT the noread.monitor argument) to the original run.jags call.

`return.samples` option to thin the final MCMC chain(s) before calculating summary statistics and returning the chains. Note that this option does NOT currently carry out thinning in JAGS, therefore R must have enough available memory to hold the chains BEFORE thinning (for very large chains, it may be necessary to specify a subset of the variables at a time using read.monitor='...' and keep.jags.files=TRUE). Default value returns all available iterations.

`recover.chains` option to try to recover successful simulations if some simulations failed (this is only relevant for parallel methods with more than 1 simulation). A value of TRUE returns only successful simulations, FALSE will cause an error if any simulation has failed. A numeric vector of specific chain(s) to be read is also permitted, but an error will be returned if any of the simulations containing these chains was unsuccessful. The default version reads the option set in `runjags.options`.

`...` additional summary parameters to be passed to `add.summary`

Value

An object of class 'runjags' (see `runjags-class`).

See Also

`runjags-class` for details of available methods for the returned object, `run.jags` for details of how to start simulations, and `runjags.options` for user options regarding warning messages etc.

Examples

```
# Run a model using parallel background JAGS calls:

# Simulate the data:
N <- 100
X <- 1:N
Y <- rnorm(N, 2*X + 10, 1)
# Initial values for 2 chains:
m <- list(-10, 10)
c <- list(-10, 10)
precision <- list(0.01, 10)

# Model in the JAGS format
model <- "model {
for(i in 1 : N){
Y[i] ~ dnorm(true.y[i], precision);
true.y[i] <- (m * X[i]) + c
}
m ~ dunif(-1000,1000)
c ~ dunif(-1000,1000)
precision ~ dexp(1)
```

```
#data# X, Y, N
#monitor# m, c, precision
#inits# m, c, precision
}"

## Not run:
# Run the model and produce plots
fileinfo <- run.jags(model=model, n.chains=2, method="bgparallel")
# Wait for the simulations to complete:
Sys.sleep(10)
# Import only variable m from the first chain:
results <- results.jags(fileinfo, read.monitor='m', recover.chains=1)
# Look at the summary statistics:
print(results)

## End(Not run)
```

---

run.jags *Run or extend a user-specified Bayesian MCMC model in JAGS from within R*

Description

Runs or extends a user specified JAGS model from within R, returning an object of class runjags-class.

Usage

```
run.jags(
  model,
  monitor = NA,
  data = NA,
  n.chains = NA,
  inits = NA,
  burnin = 4000,
  sample = 10000,
  adapt = 1000,
  noread.monitor = NULL,
  datalist = NA,
  initlist = NA,
  jags = runjags.getOption("jagspath"),
  silent.jags = runjags.getOption("silent.jags"),
  modules = runjags.getOption("modules"),
  factories = runjags.getOption("factories"),
  summarise = TRUE,
  mutate = NA,
  thin = 1,
  keep.jags.files = FALSE,
  tempdir = runjags.getOption("tempdir"),
```

```
  jags.refresh = 0.1,
  batch.jags = silent.jags,
  method = runjags.getOption("method"),
  method.options = list(),
  ...
)

extend.jags(
  runjags.object,
  add.monitor = character(0),
  drop.monitor = character(0),
  drop.chain = numeric(0),
  combine = length(c(add.monitor, drop.monitor, drop.chain)) == 0,
  burnin = 0,
  sample = 10000,
  adapt = 1000,
  noread.monitor = NA,
  jags = NA,
  silent.jags = NA,
  summarise = sample >= 100,
  thin = NA,
  keep.jags.files = FALSE,
  tempdir = runjags.getOption("tempdir"),
  jags.refresh = NA,
  batch.jags = silent.jags,
  method = NA,
  method.options = NA,
  ...
)

run.JAGS(
  model,
  monitor = NA,
  data = NA,
  n.chains = NA,
  inits = NA,
  burnin = 4000,
  sample = 10000,
  adapt = 1000,
  noread.monitor = NULL,
  datalist = NA,
  initlist = NA,
  jags = runjags.getOption("jagspath"),
  silent.jags = runjags.getOption("silent.jags"),
  modules = runjags.getOption("modules"),
  factories = runjags.getOption("factories"),
  summarise = TRUE,
  mutate = NA,
```

```
  thin = 1,
  keep.jags.files = FALSE,
  tempdir = runjags.getOption("tempdir"),
  jags.refresh = 0.1,
  batch.jags = silent.jags,
  method = runjags.getOption("method"),
  method.options = list(),
  ...
)

extend.JAGS(
  runjags.object,
  add.monitor = character(0),
  drop.monitor = character(0),
  drop.chain = numeric(0),
  combine = length(c(add.monitor, drop.monitor, drop.chain)) == 0,
  burnin = 0,
  sample = 10000,
  adapt = 1000,
  noread.monitor = NA,
  jags = NA,
  silent.jags = NA,
  summarise = sample >= 100,
  thin = NA,
  keep.jags.files = FALSE,
  tempdir = runjags.getOption("tempdir"),
  jags.refresh = NA,
  batch.jags = silent.jags,
  method = NA,
  method.options = NA,
  ...
)
```

Arguments

model	either a relative or absolute path to a textfile (including the file extension) containing a model in the JAGS language and possibly monitored variable names, data and/or initial values, or a character string of the same. No default. See read.jagsfile for more details.
monitor	a character vector of the names of variables to monitor. No default. The special node names 'deviance', 'pd', 'popt', 'dic', 'ped' and 'full.pd' are used to monitor the deviance, mean pD, mean pOpt, DIC, PED and full distribution of sum(pD) respectively. Note that these monitored nodes (with the exception of 'deviance') require multiple chains within the same simulation, and won't appear as variables in the summary statistics or plots (but see extract for a way of extracting these from the returned object).
data	a named list, data frame, environment, character string in the R dump format (see dump.format), or a function (with no arguments) returning one of these types.

If the model text contains inline #data# comments, then this argument specifies the list, data frame or environment in which to search first for these variables (the global environment is always searched last). If the model text does not contain #data# comments, then the full list or data frame (but not environment) is included as data. If the data argument is a character string, then any #data# comments in the model are ignored (with a warning). The default specifies the parent environment of the function call.

`n.chains` the number of chains to use with the simulation. More chains will improve the sensitivity of the convergence diagnostic, but will cause the simulation to run more slowly (although this may be improved by using a method such as 'parallel', 'rjparallel' or 'snow'). The minimum (and default) number of chains is 2.

`inits` either a character vector with length equal to the number of chains the model will be run using, or a list of named lists representing names and corresponding values of inits for each chain, or a function with either 1 argument representing the chain or no arguments. If a vector, each element of the vector must be a character string in the R dump format representing the initial values for that chain, or NA. If not all initialising variables are specified, the unspecified variables are taken deterministically from the mean or mode of the prior distribution by JAGS. Values left as NA result in all initial values for that chain being taken from the prior distribution. The special variables '.RNG.seed', '.RNG.name', and '.RNG.state' are allowed for explicit control over random number generators in JAGS. If a function is provided, the data is available inside the function as a named list 'data' - this may be useful for setting initial values that depend on the data. Default NA. Note that the dimensions of any variables used for initial values must match the dimensions of the same parameter in the model - recycling is not performed. If any elements of the initial values have deterministic values in the model, the corresponding elements must be defined as NA in the initial values.

`burnin` the number of burnin iterations, NOT including the adaptive iterations to use for the simulation. Note that the default is 4000 plus 1000 adaptive iterations, with a total of 5000.

`sample` the total number of (additional) samples to take. Default 10000 iterations. If specified as 0, then the model will be created and returned without any MCMC samples (burnin and adapt will be ignored). Note that a minimum of 100 samples is required to generate summary statistics.

`adapt` the number of adaptive iterations to use at the start of the simulation. If the adaptive phase is not long enough, the sampling efficiency of the MCMC chains will be compromised. If the model does not require adaptation (either because a compiled rjags model is already available or because the model contains no data) then this will be ignored, with a warning that the model is not in adaptive mode. Default 1000 iterations.

`noread.monitor` an optional character vector of variables to monitor in JAGS and output to coda files, but that should not be read back into R. This may be useful (in conjunction with keep.jags.files=TRUE) for looking at large numbers of variables a few at a time using the read.monitor argument to results.jags. This argument is ignored for the rjags and rjparallel methods, and if keep.jags.files=FALSE.

`datalist` deprecated argument.

`initlist` deprecated argument.

`jags` the system call or path for activating JAGS. Default uses the option given in `runjags.options`.

`silent.jags` option to suppress output of the JAGS simulations. Default uses the option given in `runjags.options`.

`modules` a character vector of external modules to be loaded into JAGS, either as the module name on its own or as the module name and status separated by a space, for example 'glm on'.

`factories` a character vector of factory modules to be loaded into JAGS. Factories should be provided in the format '\<facname\> \<factype\> \<status\>' (where status is optional), for example: factories='mix::TemperedMix sampler on'. You must also ensure that any required modules are also specified (in this case 'mix').

`summarise` should summary statistics be automatically calculated for the output chains? Default TRUE (but see also ?runjags.options -> force.summary).

`mutate` either a function or a list with first element a function and remaining elements arguments to this function. This can be used to add new variables to the posterior chains that are derived from the directly monitored variables in JAGS. This allows the variables to be summarised or extracted as part of the MCMC objects as if they had been calculated in JAGS, but without the computational or storage overheads associated with calculating them in JAGS directly. The plot, summary and as.mcmc methods for runjags objects will automatically extract the mutated variables along with the directly monitored variables. For an application to pairwise comparisons of different levels within fixed effects see `contrasts.mcmc`.

`thin` the thinning interval to be used in JAGS. Increasing the thinning interval may reduce autocorrelation, and therefore reduce the number of samples required, but will increase the time required to run the simulation. Using this option thinning is performed directly in JAGS, rather than on an existing MCMC object as with thin.sample. Default 1.

`keep.jags.files` option to keep the folder with files needed to call JAGS, rather than deleting it. This allows the simulation results to be re-read using results.jags(path-to-folder), even from another R session, and may also be useful for attempting to bug fix models. A character string can also provided, in which case this folder name will be used instead of the default (existing folders will NOT be over-written). Default FALSE. See also the `cleanup.jags` function.

`tempdir` option to use the temporary directory as specified by the system rather than creating files in the working directory. If keep.jags.files=TRUE then the folder is copied to the working directory after the job has finished (with a unique folder name based on 'runjagsfiles'). Any files created in the temporary directory are removed when the function exits for any reason. It is not possible to use a temporary directory with the background methods, so tempdir will be set to FALSE if not done so by the user (possibly with a warning depending on the settings in `runjags.options`). Default TRUE.

jags.refresh	the refresh interval (in seconds) for monitoring JAGS output using the 'interactive' and 'parallel' methods (see the 'method' argument). Longer refresh intervals will use slightly less processor time, but will make the simulation updates to be shown on the screen less frequently. Reducing the refresh rate to every 10 or 30 seconds may be worthwhile for simulations taking several days to run. Note that this will have no effect on the processor use of the simulations themselves. Default 0.1 seconds.
batch.jags	option to call JAGS in batch mode, rather than using input redirection. On JAGS >= 3.0.0, this suppresses output of the status which may be useful in some situations. Default TRUE if silent.jags is TRUE, or FALSE otherwise.
method	the method with which to call JAGS; probably a character vector specifying one of 'rjags', 'simple', 'interruptible', 'parallel', 'rjparallel', 'background', 'bgparallel' or 'snow'. The 'rjags' and 'rjparallel' methods run JAGS using the rjags package, whereas other options do not require the rjags package and call JAGS as an external executable. The advantage of the 'rjags' method is that the model will not need to be recompiled between successive calls to extend.jags, all other methods require a re-compilation (and adaptation if necessary) step at every call to extend.jags. Note that the 'rjparallel' and 'snow' methods may leave behind zombie JAGS processes if the user interrupts the R session used to start the simulations - for this reason the 'parallel' method is recommended for interactive use with parallel chains. The 'background' and 'bgparallel' return a filename for the started simulation, which can be read using results.jags. The 'parallel' and 'interruptible' methods for Windows require XP Professional, Vista or later (or any Unix-alike). For more information refer to the userguide vignette.
method.options	a deprecated argument currently permitted for backwards compatibility, but this will be removed from a future version of runjags. Pass these arguments directly to run.jags or extend.jags.
...	summary parameters to be passed to add.summary, and/or additional options to control some methods including n.sims for parallel methods, cl for rjparallel and snow methods, remote.jags for snow, and by and progress.bar for the rjags method.
runjags.object	the model to be extended - the output of a run.jags (or autorun.jags or extend.jags etc) function, with class 'runjags'. No default.
add.monitor	a character vector of variables to add to the monitored variable list. All previously monitored variables are automatically included - although see the 'drop.monitor' argument. Default no additional monitors.
drop.monitor	a character vector of previously monitored variables to remove from the monitored variable list for the extended model. Default none.
drop.chain	a numeric vector of chains to remove from the extended model. Default none.
combine	a logical flag indicating if results from the new JAGS run should be combined with the previous chains. Default TRUE if not adding or removing variables or chains, and FALSE otherwise.

Details

The run.jags function reads, compiles, and updates a JAGS model based on a model representation (plus data, monitors and initial values) input by the user. The model can be contained in an external

text file, or a character vector within R. The autorun.jags function takes an existing runjags-class object and extends the simulation. Running a JAGS model using these functions has two main advantages:

1. The method used to call or extend the simulation can be changed simply using the method option. The methods most likely to be used are 'interruptible' and 'rjags' which use one simulation per model, or 'parallel', 'bgparallel' and 'rjparallel' which run a separate simulation for each chain to speed up the model run. For more details see below under the 'method' argument.
2. All information required to re-run the simulations is stored within the runjags-class object returned. This complete representation can be written to a text file using write.jagsfile, then modified as necessary and re-run using only the file path as input.
3. Summary statistics for the returned simulations are automatically calculated and displayed using associated S3 methods intended to facilitate checking model convergence and run length. Additional methods are available for plot functions, as well as conversion to and from MCMC and rjags objects. See the help file for runjags-class for more details.

Value

Usually an object of class 'runjags', or an object of class 'runjagsbginfo' for background methods (see runjags-class).

References

Matthew J. Denwood (2016). runjags: An R Package Providing Interface Utilities, Model Templates, Parallel Computing Methods and Additional Distributions for MCMC Models in JAGS. Journal of Statistical Software, 71(9), 1-25. doi:10.18637/jss.v071.i09

See Also

results.jags to import completed simulations (or partially successful simulations) from saved JAGS files, runjags-class for details of available methods for the returned object, read.jagsfile for more details on the permitted format of the model file, write.jagsfile for a way to write an existing runjags object to file, and runjags.options for user options regarding warning messages etc.

Examples

```
# run a model to calculate the intercept and slope of the expression
# y = m x + c, assuming normal observation errors for y:

# Simulate the data
X <- 1:100
Y <- rnorm(length(X), 2*X + 10, 1)

# Model in the JAGS format
model <- "model {
for(i in 1 : N){
Y[i] ~ dnorm(true.y[i], precision);
```

```
true.y[i] <- (m * X[i]) + c
}
m ~ dunif(-1000,1000)
c ~ dunif(-1000,1000)
precision ~ dexp(1)
}"

# Data and initial values in a named list format,
# with explicit control over the random number
# generator used for each chain (optional):
data <- list(X=X, Y=Y, N=length(X))
inits1 <- list(m=1, c=1, precision=1,
.RNG.name="base::Super-Duper", .RNG.seed=1)
inits2 <- list(m=0.1, c=10, precision=1,
.RNG.name="base::Wichmann-Hill", .RNG.seed=2)

## Not run:
# Run the model and produce plots
results <- run.jags(model=model, monitor=c("m", "c", "precision"),
data=data, n.chains=2, method="rjags", inits=list(inits1,inits2))

# Standard plots of the monitored variables:
plot(results)

# Look at the summary statistics:
print(results)

# Extract only the coefficient as an mcmc.list object:
library('coda')
coeff <- as.mcmc.list(results, vars="m")

## End(Not run)


# The same model but using embedded shortcuts to specify data, inits and monitors,
# and using parallel chains:

# Model in the JAGS format

model <- "model {
for(i in 1 : N){ #data# N
Y[i] ~ dnorm(true.y[i], precision) #data# Y
true.y[i] <- (m * X[i]) + c #data# X
}
m ~ dunif(-1000,1000) #inits# m
c ~ dunif(-1000,1000)
precision ~ dexp(1)
#monitor# m, c, precision
}"

# Simulate the data
X <- 1:100
Y <- rnorm(length(X), 2*X + 10, 1)
```

```
N <- length(X)

initfunction <- function(chain) return(switch(chain,
"1"=list(m=-10), "2"=list(m=10)))

## Not run:
# Run the 2 chains in parallel (allowing the run.jags function
# to control the number of parallel chains). We also use a
# mutate function to convert the precision to standard deviation:
results <- run.jags(model, n.chains=2, inits=initfunction,
method="parallel", mutate=list("prec2sd", vars="precision"))

# View the results using the standard print method:
results

# Look at some plots of the intercept and slope on a 3x3 grid:
plot(results, c('trace','histogram','ecdf','crosscorr','key'),
vars=c("m","^c"),layout=c(3,3))

# Write the current model representation to file:
write.jagsfile(results, file='mymod.txt')
# And re-run the simulation from this point:
newresults <- run.jags('mymod.txt')

## End(Not run)
# Run the same model using 8 chains in parallel:
# distributed computing cluster:
## Not run:

# A list of 8 randomly generated starting values for m:
initlist <- replicate(8,list(m=runif(1,-20,20)),simplify=FALSE)

# Run the chains in parallel using JAGS (2 models
# with 4 chains each):
results <- run.jags(model, n.chains=8, inits=initlist,
method="parallel", n.sims=2)

# Set up a distributed computing cluster with 2 nodes:
library(parallel)
cl <- makeCluster(4)

# Run the chains in parallel rjags models (4 models
# with 2 chains each) on this cluster:
results <- run.jags(model, n.chains=8, inits=initlist,
method="rjparallel", cl=cl)

stopCluster(cl)

# For more examples see the quick-start vignette:
vignette('quickjags', package='runjags')

# And for more details about possible methods see:
vignette('userguide', package='runjags')
```

```
## End(Not run)
```

`run.jags.study` *Drop-k and simulated dataset studies using JAGS*

Description

These functions can be used to fit a user specified JAGS model to multiple datasets with automatic control of run length and convergence, over a distributed computing cluster such as that provided by snow. The results for monitored variables are compared to the target values provided and a summary of the model performance is returned. This may be used to facilitate model validation using simulated data, or to assess model fit using a 'drop-k' type cross validation study where one or more data points are removed in turn and the model's ability to predict that datapoint is assessed.

Usage

```
drop.k(runjags.object, dropvars, k = 1, simulations = NA, ...)

drop.k.jags(runjags.object, dropvars, k = 1, simulations = NA, ...)

drop.k.JAGS(runjags.object, dropvars, k = 1, simulations = NA, ...)

run.jags.study(
  simulations,
  model,
  datafunction,
  targets = list(),
  confidence = 0.95,
  record.chains = FALSE,
  max.time = "15m",
  silent.jags = TRUE,
  parallel.method = parLapply,
  n.cores = NA,
  export.cluster = character(0),
  inits = list(),
  ...
)
```

Arguments

`runjags.object`	an object of class `runjagsstudy-class` on which to perform the drop-k analysis
`dropvars`	the variable(s) to be eliminated from the data so that the ability of the model to predict these datapoints can be assessed. The variable can be specified as a vector, or as a single character for which partial matching will be done. Array indices can be used, but must be specified as a complete range e.g. variable[2:5,2] is permitted, but variable[,2] is not because the first index is empty

k	the number of datapoints to be dropped from each individual simulation. The default of 1 is a drop-1 study (also called a leave-one-out cross validation study).
simulations	the number of datasets to run the model on. For drop.k the default is to use the number of unique datapoints, resulting in a drop-1 study. If the specified number of simulations is different to the number of unique datapoints, the datapoints are dropped randomly between simulations.
...	optional arguments to be passed to autorun.jags, or to the parallel method function (such as 'cl').
model	the JAGS model to use, in the same format as would be specified to run.jags.
datafunction	a function that will be used to specify the data. This must take either zero arguments, or one argument representing the simulation number, and return either a named list or character vector in the R dump format containing the data specific to that simulation. It is possible to specify any data that does not change for each simulation using a #data# \<variable\> tag in the model code.
targets	a named list of variables (which can include vectors/arrays) with values to which the model outputs are compared (if stochastic). The target variable names are also automatically included as monitored variables.
confidence	a probability (or vector of probabilities) to use when calculating the proportion of credible intervals containing the true target value. Default 95% CI.
record.chains	option to return the full runjags objects returned from each simulation as a list item named 'runjags'.
max.time	the maximum time for which each individual simulation is allowed to run by the underling autorun.jags function. Acceptable units include 'seconds', 'minutes', 'hours', 'days', 'weeks', or the first letter(s) of each. Default is 15 minutes.
silent.jags	option to suppress all JAGS output, even for simulations run locally. If set to FALSE, there is no guarantee that the output will be displayed in sequential order between the parallel simulations. Default TRUE.
parallel.method	a function that will be used to call the repeated simulations. This must take the first two arguments 'X' and 'FUN' as for lapply, with other optional arguments passed through from the parent function call. Default uses parLapply, but lapply or mclapply could also be used.
n.cores	the maximum number of cores to use for parallel simulations. Default value uses detectCores, or a minumum of 2. Ignored if cl is supplied, or if parallel.method does not take a cl argument.
export.cluster	a character vector naming objects to be retrieved from the parent frame of the function call and made available to the cluster nodes. This may be useful if the initial values specified for the model are required to be extracted from the working environment, however it may be preferable to specify a function for inits instead.
inits	as for run.jags, except that it is not permitted to be an environment. It is recommended to a function to return appropriate initial values (which may depend on the data visible when the function is evaluated).

Details

The drop.k function is a wrapper to run.jags.study for the common application of drop-k cross validation studies on fitted JAGS models. The run.jags.study function is more flexible, and can be used for validating the performance of a model against simulated data with known parameters. For the latter, a user-specified function to generate suitable datasets to analyse is required.

Value

An object of class runjagsstudy-class, containing a summary of the performance of the model with regards to the target variables specified. If record.chains=TRUE, an element named 'runjags' containing a list of all the runjags objects returned will also be present. Any error messages given by individual simulations will be contained in the \$errors element of the returned list.

See Also

autorun.jags for the underlying methods used to run simulations to convergence, and runjagsstudy-class for details of the returned object

Examples

```
# For examples of usage see the following vignette:
## Not run:
vignette('userguide', package='runjags')

## End(Not run)
```

---

runjags-class *The runjags class and available S3 methods*

Description

Objects of class 'runjags' are produced by run.jags, results.jags and autorun.jags, and contain the MCMC chains as well as all information required to extend the simulation. These are a number of utility functions associated with these objects.

Usage

```
## S3 method for class 'runjags'
as.mcmc(x, vars = NA, add.mutate = TRUE, ...)

## S3 method for class 'runjags'
as.mcmc.list(x, vars = NA, add.mutate = TRUE, ...)

## S3 method for class 'runjags'
as.jags(x, adapt = 1000, quiet = FALSE, ...)

## S3 method for class 'jags'
```

```
as.runjags(
  jags.model,
  monitor = stop("No monitored variables supplied"),
  modules = runjags.getOption("modules"),
  factories = runjags.getOption("factories"),
  jags = runjags.getOption("jagspath"),
  mutate = NA,
  check = TRUE,
  ...
)

is.runjags(x)

cleanup.jags(all.folders = FALSE, silent = FALSE)

cleanup.JAGS(all.folders = FALSE, silent = FALSE)

failed.jags(show = c("model", "output"))

as.jags(x, adapt = 1000, quiet = FALSE, ...)

## Default S3 method:
as.jags(x, ...)

as.runjags(jags.model, ...)

## Default S3 method:
as.runjags(jags.model, ...)

## S3 method for class 'runjags'
residuals(
  object,
  variable = object$residual,
  show.summary = FALSE,
  output = "mean",
  ...
)

## S3 method for class 'runjags'
fitted(
  object,
  variable = object$fitted,
  show.summary = FALSE,
  output = "mean",
  ...
)
```

Arguments

x	an object of class runjags.
vars	an optional character vector of variable names to extract. If supplied, only variable names in the object supplied with a partial match to anything in 'vars' will be summarised/plotted/extracted. Note that regular expressions are not allowed, but the caret (^) token can be used to specify the match at the start of a variable name, and a quoted vars will be matched exactly. Default NA meaning all variables available are returned.
add.mutate	option to use the inbuild mutate function to produce additional MCMC variables before returning the MCMC object.
...	additional options to be passed to default methods or additional functions.
adapt	as for jags.model
quiet	as for jags.model
jags.model	a model produced by jags.model
monitor	a character vector of the names of variables to monitor, as for run.jags
modules	a character vector of external modules to be loaded into JAGS, either as the module name on its own or as the module name and status separated by a space, for example 'glm on'.
factories	a character vector of factory modules to be loaded into JAGS. Factories should be provided in the format '\<facname\> \<factype\> \<status\>' (where status is optional), for example: factories='mix::TemperedMix sampler on'. You must also ensure that any required modules are also specified (in this case 'mix').
jags	the system call or path for activating JAGS. Default uses the option given in runjags.options.
mutate	either a function or a list with first element a function and remaining elements arguments to this function that can be used to add variables to the model output. See add.summary for more details.
check	option to check that the model can be (re)-compiled.
all.folders	option to remove ALL simulation folders created using keep.jags.files=TRUE and not just unsuccessful simulations.
silent	option to suppress feedback when deleting simulation folders.
show	which parts of the failed JAGS simulation to display - options are: 'model', 'data', 'inits', 'output', 'end.state', 'all'
object	an object of class runjags.
variable	the name of the variable within the JAGS simulation that denotes the residual/fitted variable. This must be specified to be able to use the residuals and fitted methods.
show.summary	option to show the full summary statistics of the returned models before extracting just the residuals/fitted variable information.
output	the type of output required for the residuals and fitted methods - options are: 'mean', 'mcmc', 'hpd', 'summary', 'runjags'.

Details

The functions and methods detailed here permit conversion of runjags objects to MCMC objects and to/from jags models created by `jags.model`. There are also S3 methods for print, summary and plot available for runjags class objects - see `add.summary` for details of the arguments available to these. The 'failed.jags' function allows the user to interrogate the details of JAGS models that failed to compile or produce MCMC output. By default, any simulation folders for models that failed to import are kept until the R session is ended - in some circumstances it may be possible to partially recover the results using `results.jags`. The cleanup.jags function can be used to remove simulation folders created in the current R session, and is called when the runjags package is unloaded.

See Also

`add.summary` for details on plot, print and summary methods for runjags class objects, `extract.runjags` for a method to extract peripheral information from runjags objects, `runjags.options` for general options available, and `run.jags` and `autorun.jags` for the functions that create objects of this class.

Examples

```
if(require('rjags')){
# Coercion between jags and runjags objects (requires loading the rjags package):
data(LINE)
jags.model <- LINE
runjags.model <- as.runjags(jags.model, monitor=c('alpha','beta'))
## Not run:
runjags.model <- extend.jags(runjags.model)
jags.model <- as.jags(runjags.model)
# Coercion to MCMC (requires loading the coda package):
library('coda')
mcmc <- as.mcmc.list(runjags.model)
summary(mcmc)

## End(Not run)
}
```

`runjags.options` *Options for the runjags package*

Description

Utility function to change the default options for the runjags package. Options will be used for all runjags function calls until the runjags package is unloaded. For a permanent solution, create a named list called '.runjags.options' containing the desired options in an R profile file - on loading, runjags will check to see if this object exists in the global environment and set the options automatically.

Usage

```
runjags.options(...)

runjags.getOption(name)

runJAGS.getOption(name)
```

Arguments

...	named option(s) to change - for a list of available options, see details below.
name	the name of the option to get the current value of - for a list of available options, see details below.

Details

The following default options can be specified:

- **jagspath** - the path to JAGS to use unless over-ridden in a function call (uses the findjags() function by default).
- **method** - the runjags method to use unless over-ridden in a function call (default is 'rjags' if the rjags package is installed, or 'interruptible' otherwise).
- **tempdir** - default to temporary directory unless over-ridden in a function call (default TRUE).
- **plot.layout** - the layout for plots unless over-ridden in a function call. Must be a numeric vector of length 2.
- **new.windows** - use multiple windows for plots unless over-ridden in a function call (default is platform dependent).
- **modules** - the modules to load unless over-ridden in a function call (default none).
- **factories** - the factories to load unless over-ridden in a function call (default none).
- **bg.alert** - an optional command to run once background JAGS processes have completed. Note that this command is run on the command line via system(), so will be system dependent. The default attempts to make an alert sound using a system appropriate method, which may not work on all platforms.
- **linenumbers** - display line numbers when printing runjags model, data and inits class objects unless over-ridden in a function call (default none).
- **inits.warning** - display warning messages about initial values being not specified or re-used.
- **rng.warning** - display warning messages relating to pseudo-random number generation for parallel chains.
- **summary.warning** - display a warning message if summary statistics are requested for a small number of samples (and a few other similar situations).
- **blockcombine.warning** - display a warning message if multiple data or inits blocks are combined in a model file.
- **blockignore.warning** - display a warning message if ignoring monitors, data or inits in the model file because a character argument was given for the same parameters to the run.jags function.

- **tempdir.warning** - display a warning message if tempdir=TRUE is requested with a background method.
- **nodata.warning** - display a warning message if the model has been run without any data.
- **adapt.incomplete** - all models are checked to make sure that the adaptive phase has completed - this option controls the behaviour of runjags if this adaptation is not complete before MCMC sampling. If adapt.incomplete='silent' no action is taken, if 'warning' then the model run is continued but a warning is given after the simulation is finished, and if 'error' an error will be returned. Note that for most methods the error is returned immediately following the adapt/burnin phases (so the sample iterations are not run), but for the simple and snow methods the full model will be run before the error is given.
- **repeatable.methods** - option to ensure that the MCMC object produced by the rjags and rjparallel methods are identical to those produced by other methods (given the same starting values). This is primarily for extending compiled models, where additional burnin iterations will be done to replace unnecessary adpative steps for consistency with other methods, and following dic sampling, where the rjags model will be reset to the state it was in before dic sampling. Note that the precision of the numbers returned may differ between methods on some platforms.
- **silent.jags** - suppress output of JAGS (or rjags) when updating models.
- **silent.runjags** - suppress feedback provided by the runjags functions.
- **predraw.plots** - automatically pre-calculate convergence diagnostic plots (this will save time when displaying plots at the cost of increased storage requirement for the runjags object).
- **force.summary** - override the default behaviour to omit calculation of summary statistics for greater than 50 variables, and attempt to calculate summary statistics anyway (this may cause long delays in the processing of results).
- **mode.continuous** - calculate the mode of continuous variables for summary statistics (requires the "modeest" package to be installed).
- **timeout.import** - the maximum number of seconds for runjags to wait for coda files to finish being written before giving up. If a large number of monitored variables are being written, either the timeout can be increased or results.jags() can be used once the files have been written.
- **partial.import** - force runjags to read in successful simulations even when parallel simulations crashed. If this option is set to TRUE, it is not guaranteed that a model result will contain the requested number of chains!
- **keep.crashed.files** - allows folders containing crashed simulations to be preserved even if keep.jags.files = FALSE. Any folders kept will be deleted when runjags is unlaoded or when R quits.
- **full.cleanup** - when unloading the runjags package, should all simulation folders preserved using keep.jags.files=TRUE be deleted? This option may not work as expected on all systems when quitting R, but should always work for unloadNamespace('runjags'). Note also that folders for any failed JAGS runs are *always* deleted on exit - if you want to keep these, they will have to be copied manually.
- **debug** - display internal debugging output.

Value

The current value of all available runjags options (after applying any changes specified) is returned invisibly as a named list.

See Also

run.jags, findjags, runjags-class

Examples

```
## Not run:

# Create a list of options in the global environment (perhaps in an
# R startup profile file) BEFORE load()ing runjags:
.runjags.options <- list(inits.warning=FALSE, rng.warning=FALSE)
# Or if it is run in a different environment:
# .runjags.options <<- list(inits.warning=FALSE, rng.warning=FALSE)

# Then load runjags and verify that the options have been set:
library('runjags')
print(runjags.options())


# Change the default option to remove all feedback provided by
# runjags and JAGS/rjags (only errors will be printed to screen):
runjags.options(silent.jags=TRUE, silent.runjags=TRUE)


## End(Not run)
```

---

runjags.printmethods *Print methods for runjags helper classes*

Description

Print methods for a number of classes that are associated with runjags objects, such as model, data and initial values files etc.

Usage

```
## S3 method for class 'failedjags'
print(x, linenumbers = runjags.getOption("linenumbers"), ...)

## S3 method for class 'runjagsmodel'
print(x, linenumbers = runjags.getOption("linenumbers"), ...)

## S3 method for class 'runjagsdata'
```

```
print(x, linenumbers = runjags.getOption("linenumbers"), ...)

## S3 method for class 'runjagsinits'
print(x, linenumbers = runjags.getOption("linenumbers"), ...)

## S3 method for class 'runjagsoutput'
print(x, linenumbers = runjags.getOption("linenumbers"), ...)

## S3 method for class 'rjagsoutput'
print(x, ...)

## S3 method for class 'crosscorrstats'
print(x, vars = NA, digits = 5, ...)

## S3 method for class 'mcsestats'
print(x, vars = NA, digits = 5, ...)

## S3 method for class 'gelmanwithtarget'
print(x, vars = NA, digits = 3, ...)

## S3 method for class 'dicstats'
print(x, digits = 3, ...)

## S3 method for class 'runjagsbginfo'
print(x, ...)

## S3 method for class 'runjagsstudy'
print(x, ...)

## S3 method for class 'runjagsstudy'
summary(object, ...)

## S3 method for class 'runjagsstudy'
plot(x, ...)
```

Arguments

x	the object to be printed or converted.
linenumbers	option to display line numbers alongside model, data and initial values output (this may be helpful for debugging). Defualt uses the option set in runjags.options.
...	other arguments which are passed to the default print method for some methods but ignored (with/without a warning) for others
vars	an optional character vector of variable names. If supplied, only variable names in the object supplied with a partial match to anything in 'vars' will be used. Note that regular expressions are not allowed, but the caret (^) token can be used to specify the match at the start of a variable name, and a quoted vars will be matched exactly. Default NA meaning all variables available are returned.
digits	the number of digits to display for printed numerical output.

`object` the object to be summarised.

References

Matthew J. Denwood (2016). runjags: An R Package Providing Interface Utilities, Model Templates, Parallel Computing Methods and Additional Distributions for MCMC Models in JAGS. Journal of Statistical Software, 71(9), 1-25. doi:10.18637/jss.v071.i09

See Also

`runjags-class` for print and plot methods associated with the main runjags class

---

`template.jags` *Generate a generalised linear mixed model (GLMM) specification in JAGS*

Description

Use an lme4 style syntax to create a JAGS model representation of a GLMM, including all data, initial values and monitor specifications required to run the model using `run.jags`.

Usage

```
template.jags(
  formula,
  data,
  file = "JAGSmodel.txt",
  family = "gaussian",
  write.data = TRUE,
  write.inits = TRUE,
  precision.prior = "dgamma(0.001, 0.001)",
  effect.prior = "dnorm(0, 10^-6)",
  n.chains = 2,
  precision.inits = c(0.01, 10),
  effect.inits = c(-1, 1),
  inits = NULL
)

template.JAGS(
  formula,
  data,
  file = "JAGSmodel.txt",
  family = "gaussian",
  write.data = TRUE,
  write.inits = TRUE,
  precision.prior = "dgamma(0.001, 0.001)",
  effect.prior = "dnorm(0, 10^-6)",
```

```
  n.chains = 2,
  precision.inits = c(0.01, 10),
  effect.inits = c(-1, 1),
  inits = NULL
)
```

Arguments

formula	a formula representation of the desired model, using lme4 style syntax. Two-way interactions for all variables are permitted, as are random intercepts.
data	a data frame containing the variables specified in formula. This must be specified.
file	the filename of the model to output. This will be over-written if it exists.
family	a character string representing the response distribution - options are: 'gaussian', 'binomial', 'Poisson', 'negative binomial', 'ZIB', 'ZIP', 'ZINB' (the latter denote zero-inflated distributions).
write.data	option to write the data to file with the model. If the data is very large it may be better not to write this to file, but the same data frame must be given to the subsequent run.jags call that runs the model.
write.inits	option to write the initial values to file with the model.
precision.prior	the prior distribution to be used for precision parameters.
effect.prior	the prior distribution to be used for linear and fixed effect terms, as well as interactions and the intercept.
n.chains	the number of chains to use.
precision.inits	a numeric vector of initial values from which the precision parameters in the model will be randomly chosen. It is recommended to make these over-dispersed, but if the values are too extreme the model may not compile.
effect.inits	a numeric vector of initial values from which the effect parameters in the model will be randomly chosen. It is recommended to make these over-dispersed, but if the values are too extreme the model may not compile.
inits	an optional list of named lists to specify initial values for one or more parameters in each chain. The number of named lists must match n.chains.

Details

This function is designed to allow new users to MCMC to create relatively simple GLMM models in JAGS using an lme4-style formula interface. Examining the template created by this function is a good way to learn about how the BUGS language is structured, as well as the options provided by the runjags package. After generating the template model, the user is encouraged to examine the model file and make whatever changes are necessary before running the model using 'run.jags'. You can also run the models with no changes and compapre the results to those obtained through more standard model fitting approaches to learn more about how the differently presented sets of inference relate to each other. Note that the effect of the reference level for factors is explicitly given as 0 in output from runjags. For more about the BUGS language, see Lunn et al (2012).

Value

The filename of the created model template.

References

Lunn D, Jackson C, Best N, Thomas A, Spiegelhalter D (2012). The BUGS book: A practical introduction to Bayesian analysis. CRC press; and Matthew J. Denwood (2016). runjags: An R Package Providing Interface Utilities, Model Templates, Parallel Computing Methods and Additional Distributions for MCMC Models in JAGS. Journal of Statistical Software, 71(9), 1-25. doi:10.18637/jss.v071.i09

See Also

run.jags to run the model, add.summary for details of summary statistics available from the fitted model, and runjags-class for details of how to extract information such as residuals and the fitted values.

Examples

```
## Not run:
# Create a simple linear model and compare the results to LM:

# This is based on the example in ?lm:
ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)
group <- gl(2, 10, 20, labels = c("Ctl","Trt"))
weight <- c(ctl, trt)
D9 <- data.frame(weight, group)
lm.D9 <- lm(weight ~ group, data=D9)

# The JAGS equivalent:
model <- template.jags(weight ~ group, D9, n.chains=2,
family='gaussian')
JAGS.D9 <- run.jags(model)
summary(JAGS.D9)
summary(lm.D9)
# Note that lm reports sigma and JAGS the precision - to
# make them more comparable we could use a mutate function:
JAGS.D9 <- run.jags(model, mutate=list(prec2sd, 'precision'))
summary(JAGS.D9)
summary(lm.D9)
# Compare the estimated residuals:
plot(residuals(lm.D9), residuals(JAGS.D9, output='mean'))

# For more examples see:
vignette('quickjags', package='runjags')

## End(Not run)
```

`template_huiwalter` *Create a Hui-Walter model based on paired test data for an arbitrary number of tests and populations*

Description

Create a Hui-Walter model based on paired test data for an arbitrary number of tests and populations

Usage

```
template_huiwalter(
  testdata,
  outfile = "huiwalter_model.txt",
  covariance = FALSE,
  se_priors = "dbeta(1,1)",
  sp_priors = "dbeta(1,1)",
  cov_as_cor = FALSE
)
```

Arguments

`testdata`	the input paired test data, where each column name corresponds to a test result - except possibly "ID" which is ignored, and "Population" indicating a population identifier for that row. Each row must represent test results from the same individual either as logical or a factor with two levels (and where the first level indicates a negative test result). Data may be missing at random (except for Population).
`outfile`	the name of the text file to save the model representation
`covariance`	should covariance terms be activated or omitted?
`se_priors`	the priors to use for sensitivity parameters (can be adjusted in the model once it is generated)
`sp_priors`	the priors to use for specificity parameters (can be adjusted in the model once it is generated)
`cov_as_cor`	option for the prior for covariance terms to be put on the correlation rather than covariance directly

Examples

```
N <- 600
status <- rbinom(N, 1, rep(c(0.25,0.5,0.75), each=N/3))
testdata <- data.frame(Population = rep(1:3, each=N/3),
    Test1 = rbinom(N, 1, status*0.75 + (1-status)*0.05),
    Test2 = rbinom(N, 1, status*0.75 + (1-status)*0.05),
    Test3=rbinom(N, 1, status*0.75 + (1-status)*0.05)
)
template_huiwalter(testdata, outfile="huiwalter_model.txt", covariance=TRUE)
```

```
## Not run:
results <- run.jags("huiwalter_model.txt")

## End(Not run)
unlink("huiwalter_model.txt")
```

`testjags` *Analyse the System to Check That JAGS Is Installed*

Description

Test the users system to determine the operating system, version of R installed, and version of JAGS installed. Some information is collected from other functions such as .platform and Sys.info. Used by the run.jags function.

Usage

```
testjags(jags = runjags.getOption("jagspath"), silent = FALSE)
```

Arguments

`jags`	the system call or path for activating JAGS. Default calls findjags() to attempt to locate JAGS on your system automatically. In unix the system call should always be 'jags', in Windows a path to the JAGS executable or the enclosing /bin or /JAGS folder is required.
`silent`	should on-screen feedback be suppressed? Default FALSE.

Value

A named list of values containing information about the JAGS installs found on the user's system (returned invisibly).

See Also

`run.jags`, `findjags`

Examples

```
# Run the function to determine if JAGS is installed:
testjags()
testjags('some/jags/path')
```

`timestring` *Calculate the Elapsed Time in Sensible Units*

Description

Function to calculate the elapsed time between 2 time periods (in seconds), or to calculate a number of seconds into a time measurement in more sensible units.

Usage

```
timestring(time1, time2 = NA, units = NA, show.units = TRUE)
```

Arguments

`time1`	either the time index (from Sys.time()) at the start of the time period, a length of time in seconds, or an object of class 'difftime'.
`time2`	either the time index (from Sys.time()) at the end of the time period, or missing data if converting a single length of time. Default NA.
`units`	either missing, in which case a sensible time unit is chosen automatically, or one of 's', 'm', 'h', 'd', 'w', 'y' to force a specific unit. Default NA.
`show.units`	if TRUE, then the time is returned with units, if FALSE then just an integer is returned. Default TRUE.

Value

A time measurement, with or without units.

See Also

`Sys.time`

Examples

```
# Time how long it takes to complete a task:

pre.time <- Sys.time()
Sys.sleep(2)  # PROCESS TO TIME
post.time <- Sys.time()
timestring(pre.time, post.time)

# Convert 4687 seconds into hours:

timestring(4687, units='hours', show.units=FALSE)
```

write.jagsfile *Write a complete JAGS model to a text file*

Description

Writes the JAGS model, data, initial values and monitored variables etc to a file. The model can then be run using a call to link{run.jags} with the filename as the model argument.

Usage

```
write.jagsfile(
  runjags.object,
  file,
  remove.tags = TRUE,
  write.data = TRUE,
  write.inits = TRUE
)

write.JAGSfile(
  runjags.object,
  file,
  remove.tags = TRUE,
  write.data = TRUE,
  write.inits = TRUE
)
```

Arguments

runjags.object	a valid (but not necessarily updated) runjags object to be saved to file. No default.
file	a filename to which the model will be written. Note that any files already existing in that location will be overwritten with no warning (see new_unique for a way to generate unique filenames). No default.
remove.tags	should the runjags tags #data#, #inits#, #monitors#, #modules#, #factories#, #residual#, #fitted# and #response# be removed from the original model code before writing it to file? If left in, these may create conflicts with the tags automatically added to the new file.
write.data	should the data also be written to file? If FALSE, the model may not run from the file without specifying a new source of data.
write.inits	should the data also be written to file? If FALSE, the model may not run from the file without specifying new initial values.

Value

Returns the filename that the model was saved to (invisibly)

References

Lunn D, Jackson C, Best N, Thomas A, Spiegelhalter D (2012). The BUGS book: A practical introduction to Bayesian analysis. CRC press; and Matthew J. Denwood (2016). runjags: An R Package Providing Interface Utilities, Model Templates, Parallel Computing Methods and Additional Distributions for MCMC Models in JAGS. Journal of Statistical Software, 71(9), 1-25. doi:10.18637/jss.v071.i09

See Also

read.jagsfile and run.jags for the reverse operation

Examples

```
# Set up a model:
# y = m x + c, assuming normal observation errors for y:

# Simulate the data
X <- 1:100
Y <- rnorm(length(X), 2*X + 10, 1)

# Model in the JAGS format
model <- "model {
for(i in 1 : N){
Y[i] ~ dnorm(true.y[i], precision);
true.y[i] <- (m * X[i]) + c
}
m ~ dunif(-1000,1000)
c ~ dunif(-1000,1000)
precision ~ dexp(1)
}"

# Data and initial values in a named list format,
# with explicit control over the random number
# generator used for each chain (optional):
data <- list(X=X, Y=Y, N=length(X))
inits1 <- list(m=1, c=1, precision=1,
.RNG.name="base::Super-Duper", .RNG.seed=1)
inits2 <- list(m=0.1, c=10, precision=1,
.RNG.name="base::Wichmann-Hill", .RNG.seed=2)

## Not run:
# Compile the model but don't update it (sample=0):
compiled <- run.jags(model=model, monitor=c("m", "c", "precision"),
data=data, n.chains=2, inits=list(inits1,inits2), sample=0)

# Save the complete model to a file:
filepath <- write.jagsfile(compiled, file='model.txt')

# And run the model from the file:
results <- run.jags(filepath)
```

```
## End(Not run)
```

Index