

Package ‘runjags’

November 1, 2009

Version 0.9.5-2

Date 2009-31-10

Title Run Bayesian MCMC Models in the BUGS syntax from Within R

Author Matthew Denwood <m.denwood@vet.gla.ac.uk> (funded as part of the DEFRA VTRI project 0101).

Maintainer Matthew Denwood <m.denwood@vet.gla.ac.uk>

Depends R (>= 2.6), coda, lattice, stats, utils

SystemRequirements jags

Description A set of functions to allow any user specified model to be run in JAGS from within R, returning the MCMC chains as R objects. Includes functions to read external WinBUGS type textfiles, and allows several ways of automatically specifying model data from existing R objects or R functions. Also includes functions to automatically calculate model run length, autocorrelation and Gelman Rubin statistic diagnostics for all models to simplify the process of achieving chain convergence. Designed for maximum compatibility with WinBUGS syntax, although minor modification to existing .bug files will be required. Requires Just Another Gibbs Sampler (JAGS) for most functions, see: <http://www-fis.iarc.fr/~martyn/software/jags/>

License GPL

URL <http://cran.r-project.org/web/packages/runjags/>

Repository CRAN

Date/Publication 2009-11-01 17:34:08

R topics documented:

| | |
|----------------------------|----|
| ask | 2 |
| autorun.jags | 3 |
| autorun.jagsfile | 8 |
| combine.mcmc | 10 |
| dump.format | 12 |

| | |
|--------------|----|
| findjags | 13 |
| new_unique | 14 |
| read.winbugs | 15 |
| run.jags | 17 |
| run.jagsfile | 21 |
| testjags | 24 |
| timestring | 25 |

| | |
|--------------|-----------|
| Index | 26 |
|--------------|-----------|

| | |
|-----|---|
| ask | <i>Obtain Input from User With Error Handling</i> |
|-----|---|

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 |

Author(s)

Matthew Denwood (m.denwood@vet.gla.ac.uk) funded as part of the DEFRA VTRI project 0101.

See Also

[readline](#), [menu](#)

Examples

```
# Ask the user if they want to proceed
## Not run:
ask_yn("Do you want to start the program now? ", type="logical")
## End(Not run)
```

| | |
|--------------|--|
| autorun.jags | <i>Run a User Specified Bayesian MCMC Model in JAGS with Automatically Calculated Run Length and Convergence Diagnostics</i> |
|--------------|--|

Description

Runs a user specified JAGS (similar to WinBUGS) model from within R, returning a list of the MCMC chain(s) along with convergence diagnostics, autocorrelation diagnostics and monitored variable summaries. Chain convergence over the first run of the simulation is assessed using the 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 the Raftery and Lewis's diagnostic. The simulation is extended to the required sample size dependant on autocorrelation, and the number of chains.

This function is provided as an educational tool only, and is not a replacement for manually assessing convergence and Monte Carlo error for real-world applications. For more complex models, the use of `run.jags` directly with manual assessment of necessary run length is recommended. JAGS is called using the lower level function `run.jags`.

Usage

```
autorun.jags(model=stop("No model supplied"),
  monitor = stop("No monitored variables supplied"),
  data=NA, n.chains=2, inits = replicate(n.chains, NA),
  startburnin = 5000, startsample = 10000,
  psrf.target = 1.05, normalise.mcmc = TRUE,
  check.stochastic = TRUE, raftery.options = list(),
  crash.retry = 1, plots = TRUE, thin.sample = TRUE,
  jags = findjags(), silent.jags = FALSE,
  interactive=TRUE, max.time=Inf,
  adaptive=list(type="burnin", length=200), modules=c(""),
  thin = 1, monitor.deviance = FALSE, monitor.pd = FALSE,
  monitor.popt = FALSE, keep.jags.files = FALSE)
```

Arguments

| | |
|-----------------------|---|
| <code>model</code> | a character string of the model in the JAGS language. No default. |
| <code>monitor</code> | a character vector of the names of variables to monitor. For all models, specifying 'deviance' as a monitored variable will calculate the model deviance. No default. |
| <code>data</code> | either a named list or a character string in the R dump format containing the data. If left as NA, the model will be run without external data. |
| <code>n.chains</code> | 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. The minimum (and default) number of chains is 2. |

| | |
|-------------------------------|--|
| <code>inits</code> | a character vector with length equal to either <code>n.chains</code> or length 1. Each element of the vector must be a character string in the R dump format representing the initial values for that chain. If <code>inits</code> is of length 1, all chains will be run using the same initial values (this is NOT recommended and will cause problems with convergence diagnostics). If left as NA, or if only a partial list of initial values is supplied, JAGS will sample initial values from the priors for remaining variables in each chain. The special variables <code>'.RNG.seed'</code> , <code>'.RNG.name'</code> , and <code>'.RNG.state'</code> can be used for explicit control over random number generators in JAGS. Default NA. |
| <code>startburnin</code> | the number of initial updates to discard before sampling. Only used on the initial run before checking convergence. Default 5000 iterations. |
| <code>startsample</code> | the number of samples on which to assess convergence. More samples will give a better chance of allowing the chain to converge, but will take longer to achieve. Also controls the length of the pilot chain used to assess the required sampling length. The minimum is 4000 samples, which is the minimum required number of samples for a model with no autocorrelation and good convergence. Default 10000 iterations. |
| <code>psrf.target</code> | the value of the point estimate for the potential scale reduction factor of the Gelman Rubin statistic below which the chains are deemed to have converged (must be greater than 1). Default 1.05. |
| <code>normalise.mcmc</code> | the Gelman Rubin statistic is based on the assumption that the posterior distribution of monitored variables is roughly normal. For very skewed posterior distributions, it may help to log/logit transform the posterior before calculating the Gelman Rubin statistic. If <code>normalise.mcmc == TRUE</code> , the normality of the untransformed and log/logit transformed posteriors are compared for each monitored variable and the least skewed is used to calculate the Gelman Rubin statistic (this may take some time for large numbers of monitored variables). If <code>FALSE</code> , the data are left untransformed (this may give problems calculating the statistic in extreme cases). Default <code>TRUE</code> . |
| <code>check.stochastic</code> | non-stochastic monitored variables will cause errors when calculating the Gelman-Rubin statistic, if <code>check.stochastic==TRUE</code> then all monitored variables will be checked to ensure they are stochastic beforehand. This has a computational cost, and can be bypassed if <code>check.stochastic==FALSE</code> . Default <code>TRUE</code> . |
| <code>raftery.options</code> | a named list which is passed as additional arguments to <code>raftery.diag</code> . Default none (default arguments to <code>raftery.diag</code> are used). |
| <code>crash.retry</code> | 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). |
| <code>plots</code> | should traceplots and density plots be produced for each monitored variable? If <code>TRUE</code> , the returned list will include elements <code>'trace'</code> and <code>'density'</code> which consist of a list of lattice objects. The alternative is to use <code>plot(results\$mcmc)</code> to look at the density and traceplots for each variable using the traditional graphics system. Default <code>TRUE</code> . |
| <code>thin.sample</code> | option to thin the final MCMC chain(s) before calculating summary statistics and returning the chains. Thinning very long chains allows summary statistics |

to be calculated more quickly. If TRUE, the chain is thinned to as close to a minimum of startsample iterations as possible (i.e. using a thinning interval of $\text{floor}(\text{chain.length}/\text{thin.sample})$ since the value must be an integer). If FALSE the chains are not thinned. 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.

| | |
|------------------|--|
| jags | the system call or path for activating JAGS. Default calls findjags() to attempt to locate JAGS on your system. |
| silent.jags | should the JAGS output be suppressed? (logical) If TRUE, no indication of the progress of individual models is supplied. Default FALSE. |
| 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. Default "1hr". |
| adaptive | a list of advanced options controlling the length of the adaptive mode of each simulation. Extended simulations do not require an adaptive phase, but JAGS prints a warning if one is not performed. Reduce the length of the adaptive phase for very time consuming models. 'type' must be one of 'adaptive' or 'burnin'. |
| modules | external modules to be loaded into JAGS. More than 1 module can be used. Default none. |
| 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. |
| monitor.deviance | option to monitor the deviance of each monitored variable using the DIC module for JAGS. If TRUE, a 'deviance' element is returned representing this value for each monitored variable at each iteration. For more information see the JAGS user manual section 4.4. Default FALSE. |
| monitor.pd | option to monitor the effective number of parameters using the DIC module for JAGS. If TRUE, a 'pd' element is returned representing this value at each iteration. For more information see the JAGS user manual section 4.4. Default FALSE. |

- `monitor.popt` option to monitor the optimism of the expected deviance using the DIC module for JAGS. If TRUE, a 'popt' element is returned representing this value at each iteration. For more information see the JAGS user manual section 4.4. Default FALSE.
- `keep.jags.files` option to keep the folder with files needed to call JAGS, rather than deleting it. May be useful for attempting to bug fix models. Since `autorun.jags` typically makes several calls to JAGS, all folders are kept - the order in which the folders were used can be ascertained from the creation dates of the files. Default FALSE.

Details

This function runs the specified model until the point estimate of the potential scale reduction factor of the Gelman-Rubin statistic for each monitored parameter is less than `psrf.target` (default 1.05). This is intended to make sure that the chains have converged before sampling from them (although this does not guarantee convergence so manual checking of traceplots is recommended). If convergence is not achieved within the time limit, then the function returns `pilot.mcmc` rather than `mcmc`, which will always be of length `startsample` since it is the unconverged `mcmc` object returned from the last run of the model. This chain is not suitable for making inference from, so a different name is used to avoid confusion. If convergence is achieved, the Raftery and Lewis's diagnostic is used to calculate the required number of samples based on the most heavily autocorrelated monitored variable. The chains are then extended to increase the sample size of each variable as necessary (so that sufficient samples are obtained from the COMBINED chains to satisfy the Raftery and Lewis's diagnostic) before being summarised and returned. Heavily autocorrelated models with large numbers of monitored variables may result in a required sample size larger than the available memory in R. If this is the case, try using the `thin` option to reduce autocorrelation (and therefore the required sample size) or monitor less variables.

Value

A list including the following elements:

- `mcmc` an MCMC object representing the model output for all chain(s). Renamed `pilot.mcmc` if the simulation is aborted before convergence or before the required sampling length is achieved
- `end.state` the end state of the last simulation extension performed. Can be used as initial values to extend the simulation further if required
- `req.samples` the minimum sample size required for the Markov chain as calculated using the Raftery and Lewis's diagnostic. This sample size is dependent on the thinning interval in JAGS
- `samples.to.conv` the number of sampled iterations discarded due to poor convergence. The total number of iterations performed for the simulation is equal to `req.samples + req.burnin + samples.to.conv` (unless the simulation was aborted before reaching the required sampling length)
- `thin` the thinning interval in JAGS used for the chains.

| | |
|----------|---|
| summary | the summary statistics for the monitored variables from the combined chains. Renamed pilot.summary if pilot.only==TRUE or if the simulation is aborted before the required sampling length is achieved |
| psrf | the Gelman Rubin statistic for the monitored variables (similar to output of gelman.diag()) |
| autocorr | the autocorrelation diagnostic for the monitored variables (output of autocorr.diag()) |
| trace | a list of lattice objects representing traceplots for each monitored variable. Calling each individual element will result in the traceplot for that variable being shown, calling the entire list will result in all traceplots being shown in new windows (which may cause problems with your R session if there are several monitored variables). Not produced if plots==FALSE. |
| density | a list of lattice objects representing density plots for each monitored variable. Calling each individual element will result in the density plot for that variable being shown, calling the entire list will result in all density plots being shown in new windows (which may cause problems with your R session if there are several monitored variables). Not produced if plots==FALSE. |

Author(s)

Matthew Denwood (m.denwood@vet.gla.ac.uk) funded as part of the DEFRA VTRI project 0101.

See Also

[run.jags](#), [autorun.jagsfile](#), [combine.mcmc](#), [raftery.diag](#), [gelman.diag](#), [autocorr.diag](#)

Examples

```
# run a model to calculate the intercept and slope of the expression  $y = m x + c$ , assuming n

## Not run:
# 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);
}"

# Convert the data to a named list
data <- list(X=x, Y=y, N=length(x))

# Run the model
results <- autorun.jags(model=model, monitor=c("m", "c", "precision"), data=data)
```

```
# Analyse traceplots of the results to assess convergence:
results$trace

# Summary of monitored variables:
results$summary
## End(Not run)
```

autorun.jagsfile *Read a User Specified Model in a WinBUGS Type Textfile or Character Variable, and Run the Simulation in JAGS with Automatically Calculated Run Length and Convergence Diagnostics*

Description

Runs a user specified JAGS (similar to WinBUGS) model in a WinBUGS type Textfile from within R, returning a list of the MCMC chain(s) along with convergence diagnostics, autocorrelation diagnostics and monitored variable summaries. This function is a wrapper for `run.jagsfile` with `autorun==TRUE`, and uses `read.winbugs` to extract model and data definitions from the specified file or string.

Usage

```
autorun.jagsfile(path=stop("No path or model string supplied"),
  datalist=NA, initlist=NA, n.chains=NA, data=NA, model=NA,
  inits=NA, monitor=NA, call.jags=TRUE, ...)
```

Arguments

| | |
|-----------------------|--|
| <code>path</code> | 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. A character vector of paths or strings is also acceptable, in which case the strings or contents of the files will be combined. 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 '}' . 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. The model block may also contain automatically generated data and initial values variables using '#data# variable' and '#inits# variable' , and more monitored variables using '#monitor# variable' . See <code>read.winbugs</code> for more information. |
| <code>datalist</code> | an optional named list containing variables used as data, or alternatively a function (with no arguments) that returns a named list. If any variables are specified in the model block using '#data# variable' , the value for the corresponding named variable is taken from datalist if present (or the result of datalist() if specified as a function which is useful for specifying randomly generated data), or |

| | |
|-----------|--|
| | the parent environment, or finally the global environment if not found anywhere else. Ignored if '#data#' variable' is not used in the model block. Default NA. |
| initlist | an optional named list containing variables used as initial values, or alternatively a function (with a single argument representing the chain number) that returns a named list. If any variables are specified in the model block using '#inits#' variable', the value for the corresponding named variable is taken from initlist if present (or the result of datalist(chain.no) if specified as a function which allows both randomly generated initial values and different values for each chain), or the parent environment, or finally the global environment if not found anywhere else. Ignored if '#inits#' variable' is not used in the model block. Note: different chains are all given the same starting values if specified as a named list or taken from any environment; if different values are desired for each chain initlist should be specified as a function. Default NA. |
| 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. If NA, the number of chains will be taken from the number of inits blocks in the model file. If NA and no inits blocks are found, 2 chains are used with a warning. Default NA. |
| data | OPTIONAL character vector in the R dump format (or named list) containing the data. If supplied (!=NA), all data in the model file is ignored. Default NA. |
| model | OPTIONAL model in JAGS syntax. If supplied (!=NA), the model in the model file is ignored. Default NA. |
| inits | OPTIONAL character vector(s) in the R dump format containing the initial value(s). If supplied (!=NA), all inits in the model file are ignored. Default NA. |
| monitor | OPTIONAL character vector containing the monitored variables. If supplied (!=NA), all monitor statements in the model block are ignored. Default NA. |
| call.jags | option results in either simulation being called if TRUE, or returns a named list of the data, model, initial values, monitored variables and number of chains (which can be supplied to autorun.jags) if FALSE. |
| ... | other options to be passed directly to autorun.jags (see autorun.jags). |

Value

The output of autorun.jags. See the help file [autorun.jags](#) for more information.

Author(s)

Matthew Denwood (m.denwood@vet.gla.ac.uk) funded as part of the DEFRA VTRI project 0101.

See Also

[run.jagsfile](#), [autorun.jags](#), [run.jags](#), [read.winbugs](#)

Examples

```

# run a model to calculate the intercept and slope of the expression  $y = m x + c$ , assuming n

## Not run:

# 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)))

results <- autorun.jagsfile(model, n.chains=2, initlist=initfunction)

# Analyse traceplots of the results to assess convergence:
results$trace

# Analyse the results
results$summary
## End(Not run)

```

combine.mcmc

Combine Two or More MCMC Objects With the Same Number of Chains Into One Longer MCMC Object

Description

Allows an MCMC object (with 1 or more chains) to be combined with another (or several other) MCMC object(s) representing extensions of the same simulation, to produce one MCMC object that contains the continuous combined Markov chains of the other MCMC objects. Also provides a safe way to thin a single MCMC object or list.

Usage

```
combine.mcmc(mcmc.objects=list(), thin=1, return.samples=NA, collapse.chains=if(ler
```

Arguments

| | |
|------------------------------|--|
| <code>mcmc.objects</code> | a list of MCMC objects, all with the same number of chains, or a single MCMC object or list. No default. |
| <code>thin</code> | 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 <code>combine.mcmc</code> . Ignored if <code>return.samples</code> is specified (<code>!=NA</code>). Default 1 (no additional thinning is performed). |
| <code>return.samples</code> | the number of samples to return after thinning. The chains will be thinned to as close to this minimum value as possible. Supersedes <code>thin</code> if both are specified. Ignored if <code>niter(mcmc.objects) < return.samples</code> . Default <code>NA</code> . |
| <code>collapse.chains</code> | 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 <code>TRUE</code> if a single <code>mcmc.object</code> is provided, or <code>FALSE</code> otherwise. |

Value

A single MCMC object of length equal to the sum of the lengths of the input MCMC objects (if `thin=1`)

Author(s)

Matthew Denwood (m.denwood@vet.gla.ac.uk) funded as part of the DEFRA VTRI project 0101.

See Also

[run.jags](#), [testjags](#)

Examples

```
# run a model, then extend the simulation and combine the two MCMC objects, with thinning to
## Not run:

# 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);
  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)

results1 <- run.jagsfile(model, n.chains=2, burnin=5000, sample=10000)
results2 <- run.jagsfile(model, inits=results1$end.state, burnin=0, sample=10000)

results <- combine.mcmc(list(results1$mcmc, results2$mcmc), return.samples=5000)

# Analyse the results
summary(results)
## End(Not run)

```

| | |
|-------------|---|
| dump.format | <i>Produce a Character String in the R Dump Format to Be Used With Jags</i> |
|-------------|---|

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 by JAGS as either data or initial values, using the run.jags function.

Usage

```
dump.format(data=list())
```

Arguments

| | |
|------|---|
| data | 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. Alternatively, if the list is of length 2 and is not named, and the first element is a character string, the first element will be used as the name for the second element. |
|------|---|

Details

This 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

A character string in the R dump format.

Author(s)

Matthew Denwood (m.denwood@vet.gla.ac.uk) funded as part of the DEFRA VTRI project 0101.

See Also[run.jags](#)**Examples**

```
initial.values.1 <- dump.format(list(mean="1", sd="0.1", lambda=matrix(1, ncol=3, nrow=10)))
initial.values.2 <- dump.format(list(mean="10", sd="10", lambda=matrix(10, ncol=3, nrow=10)))
data <- dump.format(list(N="10", Count=c(4,2,7,0,6,9,1,4,12,1)))
```

`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.

Usage

```
findjags(ostype = .Platform$OS.type)
```

Arguments

`ostype` the operating system type. There is probably no reason to want to change this...

Value

A path or command for the most likely location of JAGS 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"

Author(s)

Matthew Denwood (m.denwood@vet.gla.ac.uk) funded as part of the DEFRA VTRI project 0101.

See Also[testjags](#), [run.jags](#)**Examples**

```
findjags()
```

`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?")
```

Arguments

| | |
|---------------------|--|
| <code>name</code> | 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. |
| <code>suffix</code> | the file extension (including '.') to use (character string). Default none. |
| <code>ask</code> | 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. |
| <code>prompt</code> | what text string should be used to prompt the user? (character string) Ignored is ask==FALSE. A generic default is supplied. |

Value

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

Author(s)

Matthew Denwood (m.denwood@vet.gla.ac.uk) funded as part of the DEFRA VTRI project 0101.

See Also

[ask](#)

Examples

```
# Create a file name that is unlikely to exist already, with a .R extension.
new_unique(c("new_file", NA), ".R", ask=FALSE)
```

| | |
|--------------|--|
| read.winbugs | <i>Extract Any Models, Data, Monitored Variables or Initial Values As Character Vectors from a Winbugs Type Textfile</i> |
|--------------|--|

Description

Read a user specified WinBUGS type textfile or character variable and extract any models, data, monitored variables or initial values as character vectors. Used by (auto)run.jagsfile to interpret the input file(s) or strings.

Usage

```
read.winbugs (path)
```

Arguments

| | |
|------|---|
| path | either a relative or absolute path to a textfile (including the file extension) containing a model in the JAGS language and possibly data and/or initial values, or a character string of the same. 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 <code>run.jagsfile</code> from <code>datalist</code> , <code>initlist</code> 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 is given below. |
|------|---|

Value

A named list of 'model' containing the model description, 'data' containing the data given in the data block(s), 'autodata' containing data variables specified using '#data#' in the model block, 'inits' containing the initial values given in the initial value block(s), 'autoinits' containing initial value variables specified using '#inits#' in the model block, and 'monitor' containing the monitored variables specified in the monitor blocks and by using '#monitor#' within the model block. This function is specified primarily for WinBugs compatibility, so data blocks would normally contain the data in a list format rather than the code format that is allowed in JAGS to perform data transformations (see JAGS manual section 7.0.5). These JAGS format data blocks can be specified, and the function will attempt to differentiate the two types of data from the presence of syntactical

cues such as square brackets, for loops, 'list' and .Dim structural assignments. If none of these are found, the data block is assumed to be a WinBugs type data block and is passed to JAGS as data. This behaviour can be over-ridden by inserting '#jagsdata#' or '#bugsdata#' into the data block as appropriate. More than one data block is allowed, and each will be differentiated independently.

Author(s)

Matthew Denwood (m.denwood@vet.gla.ac.uk) funded as part of the DEFRA VTRI project 0101.

See Also

[run.jagsfile](#)

Examples

```
## Not run:

# ALL SYNTAX GIVEN BELOW IS EQUIVALENT

# Use a modified WinBUGS text file with manual inits and manual data and a seperate monitor

# Contents of a textfile 'mymodel.bug':

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)
}

monitor{
    mean
}

# end text file
```

```

read.winbugs('pathtofile/mymodel.bug')

# Use internal character variable, define monitors in the model, use autodata and manual ini

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
}
"

read.winbugs(string)

# Use 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

}
"

read.winbugs(string)

## End(Not run)

```

Description

Runs a user specified JAGS (similar to WinBUGS) model from within R, returning a list of the MCMC chain(s) along with convergence diagnostics, autocorrelation diagnostics and monitored variable summaries. Data and initial values must be supplied in the R dump format (see `dump.format()` for an easy way to do this). A character vector of variables to monitor must also be supplied.

Requires Just Another Gibbs Sampler (JAGS), see <http://www-fis.iarc.fr/~martyn/software/jags/>. The GUI interface for R in Windows may not continually refresh the output window, making it difficult to track the progress of the simulation (if `silent.jags` is `FALSE`). To avoid this, you can run the function from the terminal version of R (located in the Program Files/R/bin/ folder).

Usage

```
run.jags(model=stop("No model supplied"),
         monitor = stop("No monitored variables supplied"),
         data = NA, n.chains = 2, inits = replicate(n.chains, NA),
         burnin = 5000*thin, sample = 10000*thin,
         adapt=if(burnin<200) 100 else 0, jags=findjags(),
         silent.jags = FALSE, check.conv = TRUE, plots = TRUE,
         psrf.target = 1.05, normalise.mcmc = TRUE,
         check.stochastic = TRUE, modules=c(""), thin = 1,
         monitor.deviance = FALSE, monitor.pd = FALSE,
         monitor.popt = FALSE, keep.jags.files = FALSE)
```

Arguments

| | |
|-----------------------|---|
| <code>model</code> | a character string of the model in the JAGS language. No default. |
| <code>monitor</code> | a character vector of the names of variables to monitor. For all models, specifying 'deviance' as a monitored variable will calculate the model deviance. No default. |
| <code>data</code> | a character string in the R dump format (or a named list) containing the data. If left as NA, no external data is used in the model. Default NA. |
| <code>n.chains</code> | the number of chains to use for the simulation. Must be a positive integer. Default 2. |
| <code>inits</code> | a character vector with length equal to the number of chains the model will be run using. 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 sampled from the prior distribution by JAGS. Values left as NA result in all initial values for that chain being sampled 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. Default NA. |
| <code>burnin</code> | the number of burnin iterations (not sampled) to use (numeric). Default 5000 iterations. |
| <code>sample</code> | the number of sampling iterations to use (numeric). Default 10000 iterations. |
| <code>adapt</code> | advanced option to control the length of the adaptive phase directly, which is otherwise half the length of the burnin period. Default is 0, unless burnin is less than 200 in which case 100 adaptive iterations are used. |

| | |
|-------------------------------|---|
| <code>jags</code> | the system call or path for activating JAGS. Default calls <code>findjags()</code> to attempt to locate JAGS on your system. |
| <code>silent.jags</code> | should the JAGS output be suppressed? (logical) If TRUE, no indication of the progress of individual models is supplied. Default FALSE. |
| <code>check.conv</code> | should the convergence be assessed after the model has completed? If TRUE, each monitored variable will be assessed for a potential scale reduction factor of the Gelman Rubin statistic of less than 1.05, which indicates adequate convergence. 2 or more chains are required to assess convergence. Default TRUE. |
| <code>plots</code> | should traceplots and density plots be produced for each monitored variable? If TRUE, the returned list will include elements 'trace' and 'density' which consist of a list of lattice objects. The alternative is to use <code>plot(results\$mcmc)</code> to look at the density and traceplots for each variable using the traditional graphics system. Default TRUE. |
| <code>psrf.target</code> | the value of the point estimate for the potential scale reduction factor of the Gelman Rubin statistic below which the chains are deemed to have converged (must be greater than 1). Ignored if <code>check.conv==FALSE</code> . Default 1.05. |
| <code>normalise.mcmc</code> | the Gelman Rubin statistic is based on the assumption that the posterior distribution of monitored variables is roughly normal. For very skewed posterior distributions, it may help to log/logit transform the posterior before calculating the Gelman Rubin statistic. If <code>normalise.mcmc == TRUE</code> , the normality of the untransformed and log/logit transformed posteriors are compared for each monitored variable and the least skewed is used to calculate the Gelman Rubin statistic. If FALSE, the data are left untransformed (this may give problems calculating the statistic in extreme cases). Ignored if <code>check.conv==FALSE</code> . Default TRUE. |
| <code>check.stochastic</code> | non-stochastic monitored variables will cause errors when calculating the Gelman-Rubin statistic, if <code>check.stochastic==TRUE</code> then all monitored variables will be checked to ensure they are stochastic beforehand. This has a computational cost, and can be bypassed if <code>check.stochastic==FALSE</code> . Default TRUE. |
| <code>modules</code> | external modules to be loaded into JAGS. More than 1 module can be used. Default none. |
| <code>thin</code> | 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. Default 1. |
| <code>monitor.deviance</code> | option to monitor the deviance of each monitored variable using the DIC module for JAGS. If TRUE, a 'deviance' element is returned representing this value for each monitored variable at each iteration. For more information see the JAGS user manual section 4.4. Default FALSE. |
| <code>monitor.pd</code> | option to monitor the effective number of parameters using the DIC module for JAGS. If TRUE, a 'pd' element is returned representing this value at each iteration. For more information see the JAGS user manual section 4.4. Default FALSE. |

| | |
|------------------------------|---|
| <code>monitor.popt</code> | option to monitor the optimism of the expected deviance using the DIC module for JAGS. If TRUE, a 'popt' element is returned representing this value at each iteration. For more information see the JAGS user manual section 4.4. Default FALSE. |
| <code>keep.jags.files</code> | option to keep the folder with files needed to call JAGS, rather than deleting it. May be useful for attempting to bug fix models. Default FALSE. |

Value

The results of the simulation are returned as list including the following items (omitting the last 3 items if `check.conv==FALSE`):

| | |
|------------------------|---|
| <code>mcmc</code> | an MCMC list of MCMC objects representing the chains |
| <code>end.state</code> | a character vector of length equal to the number of chains representing a description of the model state in the R dump format for each chain at the last iteration. This can be used as an initial values vector to restart a new simulation in the same place as the previous simulation ended. |
| <code>burnin</code> | number of burnin iterations discarded before sampling. |
| <code>sample</code> | number of sampled iterations. |
| <code>thin</code> | the thinning interval in JAGS used for the chains. |
| <code>summary</code> | the summary of each monitored variable from the combined chains, equivalent to <code>summary(combine.mcmc(mcmc, collapse.chains=TRUE))</code> . |
| <code>psrf</code> | the output of the Gelman Rubin diagnostic (similar [but not exactly equivalent] to <code>gelman.diag(mcmc)</code>). |
| <code>autocorr</code> | the output of the autocorrelation diagnostic (equivalent to <code>autocorr.diag(mcmc)</code>). |
| <code>trace</code> | a list of lattice objects representing traceplots for each monitored variable. Calling each individual element will result in the traceplot for that variable being shown, calling the entire list will result in all traceplots being shown in new windows (which may cause problems with your R session if there are several monitored variables). Not produced if <code>plots==FALSE</code> . |
| <code>density</code> | a list of lattice objects representing density plots for each monitored variable. Calling each individual element will result in the density plot for that variable being shown, calling the entire list will result in all density plots being shown in new windows (which may cause problems with your R session if there are several monitored variables). Not produced if <code>plots==FALSE</code> . |

Author(s)

Matthew Denwood (m.denwood@vet.gla.ac.uk) funded as part of the DEFRA VTRI project 0101.

See Also

[autorun.jags](#), [run.jagsfile](#), [combine.mcmc](#), [testjags](#), [dump.format](#), [summary.mcmc](#), [gelman.diag](#), [autocorr.diag](#)

Examples

```

# run a model to calculate the intercept and slope of the expression  $y = m x + c$ , assuming n

## Not run:

# 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);
}"

# Use dump.format to convert the data and initial values files into the R dump format, with
data <- dump.format(list(X=X, Y=Y, N=length(X)))
inits1 <- dump.format(list(m=1, c=1, precision=1, .RNG.name="base::Super-Duper", .RNG.seed=1
inits2 <- dump.format(list(m=0.1, c=10, precision=1, .RNG.name="base::Wichmann-Hill", .RNG.s

# Run the model and produce plots
results <- run.jags(model=model, monitor=c("m", "c", "precision"), data=data, n.chains=2, in
inits2), plots = TRUE)

# Density plots of the monitored variables:
results$density

# Analyse the results
results$summary
## End(Not run)

```

run.jagsfile

Read a User Specified Model in a WinBUGS Type Textfile or Character Variable, and Run the Simulation in JAGS from Within R

Description

Runs a user specified JAGS (similar to WinBUGS) model in a WinBUGS type textfile or character variable from within R, returning a list of the MCMC chain(s) along with optional convergence diagnostics, autocorrelation diagnostics and monitored variable summaries. JAGS is called using the lower level function `run.jags`, and `read.winbugs` is used to extract model and data definitions from the specified file or string.

Usage

```
run.jagsfile(path=stop("No path or model string supplied"),
  datalist = NA, initlist = NA, n.chains=NA, data=NA,
  model=NA, inits=NA, monitor=NA, call.jags=TRUE,
  autorun=FALSE, ...)
```

Arguments

| | |
|----------|---|
| path | 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. 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 '}' . 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. The model block may also contain automatically generated data and initial values variables using '#data# variable' and '#inits# variable' , and more monitored variables using '#monitor# variable' . See read.winbugs for more information. |
| datalist | an optional named list containing variables used as data, or alternatively a function (with no arguments) that returns a named list. If any variables are specified in the model block using '#data# variable' , the value for the corresponding named variable is taken from datalist if present (or the result of datalist() if specified as a function which is useful for specifying randomly generated data), or the parent environment, or finally the global environment if not found anywhere else. Ignored if '#data# variable' is not used in the model block. Default NA. |
| initlist | an optional named list containing variables used as initial values, or alternatively a function (with a single argument representing the chain number) that returns a named list. If any variables are specified in the model block using '#inits# variable' , the value for the corresponding named variable is taken from initlist if present (or the result of datalist(chain.no) if specified as a function which allows both randomly generated initial values and different values for each chain), or the parent environment, or finally the global environment if not found anywhere else. Ignored if '#inits# variable' is not used in the model block. Note: different chains are all given the same starting values if specified as a named list or taken from any environment; if different values are desired for each chain initlist should be specified as a function. Default NA. |
| 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. If NA, the number of chains will be taken from the number of inits blocks in the model file. If NA and no inits blocks are found, 2 chains are used with a warning. Default NA. |
| data | OPTIONAL character vector in the R dump format (or named list) containing the data. If supplied (!=NA), all data in the model file is ignored. Default NA. |
| model | OPTIONAL model in JAGS syntax. If supplied (!=NA), the model in the model file is ignored. Default NA. |

| | |
|-----------|--|
| inits | OPTIONAL character vector(s) in the R dump format containing the initial value(s). If supplied (!=NA), all inits in the model file are ignored. Default NA. |
| monitor | OPTIONAL character vector containing the monitored variables. If supplied (!=NA), all monitor statements in the model block are ignored. Default NA. |
| call.jags | option results in either simulation being called if TRUE, or returns a named list of the data, model, initial values, monitored variables and number of chains (which can be supplied to run.jags) if FALSE. |
| autorun | option to call autorun.jags rather than run.jags for the simulation, which allows automatic calculation of the necessary run length and convergence diagnostics. If TRUE, burnin, sample and check.conv are ignored. See also autorun.jagsfile for a wrapper for this function. Default FALSE. |
| ... | other options to be passed directly to run.jags (see run.jags) or autorun.jags (see autorun.jags). |

Value

The output of run.jags (or autorun.jags if autorun==TRUE). See the help file [run.jags](#) or [autorun.jags](#) for more information.

Author(s)

Matthew Denwood (m.denwood@vet.gla.ac.uk) funded as part of the DEFRA VTRI project 0101.

See Also

[autorun.jagsfile](#), [run.jags](#), [autorun.jags](#), [read.winbugs](#)

Examples

```
# run a model to calculate the intercept and slope of the expression  $y = m x + c$ , assuming n
## Not run:

# 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)))

results <- run.jagsfile(model, n.chains=2, initlist=initfunction)

# Analyse the results
results$summary
## End(Not run)
```

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=findjags(), silent=FALSE)
```

Arguments

| | |
|---------------------|---|
| <code>jags</code> | the system call or path for activating JAGS. Default calls <code>findjags()</code> 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. |
| <code>silent</code> | should on-screen feedback be suppressed? Default FALSE. |

Value

The following self-explanatory information is returned: `os`, `JAGS.available` (logical), `JAGS.path`, `popen.support` (internalisation of terminal output), `JAGS.version`, `R.version`, `R.GUI`, `R.package.type`, `username`.

Author(s)

Matthew Denwood <m.denwood@vet.gla.ac.uk> funded as part of the DEFRA VTRI project 0101.

See Also

[run.jags](#), [findjags](#)

Examples

```
testjags()
```

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, or a length of time in seconds. |
| 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.

Author(s)

Matthew Denwood <m.denwood@vet.gla.ac.uk> funded as part of the DEFRA VTRI project 0101.

See Also

[Sys.time](#)

Examples

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

pre.time <- Sys.time()
for (i in 1:10000000) hold <- exp(100) # PROCESS TO TIME
post.time <- Sys.time()
timestring(pre.time, post.time)

# Convert 4687 seconds into hours:

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

Index

*Topic **methods**

ask, 2
combine.mcmc, 10
dump.format, 12
findjags, 13
new_unique, 14
read.winbugs, 15
testjags, 24
timestring, 25

*Topic **models**

autorun.jags, 3
autorun.jagsfile, 8
run.jags, 18
run.jagsfile, 21

ask, 2, 14
autocorr.diag, 7, 20
autorun.JAGS (*autorun.jags*), 3
autorun.jags, 3, 9, 20, 23
autorun.JAGSfile
 (*autorun.jagsfile*), 8
autorun.jagsfile, 7, 8, 23

combine.MCMC (*combine.mcmc*), 10
combine.mcmc, 7, 10, 20

dump.format, 12, 20

findJAGS (*findjags*), 13
findjags, 13, 24

gelman.diag, 7, 20

menu, 2

new_unique, 14

raftery.diag, 4, 7
read.WinBUGS (*read.winbugs*), 15
read.winbugs, 8, 9, 15, 21–23
readline, 2

run.JAGS (*run.jags*), 18
run.jags, 3, 7, 9, 11, 13, 17, 21, 23, 24
run.JAGSfile (*run.jagsfile*), 21
run.jagsfile, 8, 9, 15, 16, 20, 21

summary.mcmc, 20
Sys.time, 25

testJAGS (*testjags*), 24
testjags, 11, 13, 20, 24
timestring, 25