Package: rjags (via r-universe)

November 23, 2026

Version 5-6

Date 2026-11-22

Title Bayesian Graphical Models using MCMC

Depends R (>= 2.14.0), coda (>= 0.13)

SystemRequirements JAGS (>= 5.0.0)

Suggests tcltk

Description Interface to the JAGS MCMC library.

License GPL (== 2)

URL https://mcmc-jags.sourceforge.io

Config/pak/sysreqs jags

Repository https://test.r-universe.dev

Date/Publication 2026-11-22 18:18:26 UTC

RemoteUrl https://github.com/jags/rjags

RemoteRef HEAD

RemoteSha ad224e351c8744ace28f600357b109c39a3439e3

Contents

rjags-package 2
adapt 3
coda.samples 4
collapse 5
control 6
dic.samples 7
diffdic 9
dimtags 10
jags.model 11
jags.module 13
jags.object 14
jags.samples 15
jags.seed 17

jags.version 18
line 18
mcarray.object 19
node.names 20
nthread 21
parallel 21
pkg.compile.flags 22
read.jagsdata 23
rjags-deprecated 24
update 25
waic.samples 26

Index **29**

`rjags-package` *Bayesian graphical models using MCMC*

Description

The rjags package provides an interface from R to the JAGS library for Bayesian data analysis. JAGS uses Markov Chain Monte Carlo (MCMC) to generate a sequence of dependent samples from the posterior distribution of the parameters.

Details

JAGS is a clone of BUGS (Bayesian analysis Using Gibbs Sampling). See Lunn et al (2009) for a history of the BUGS project. Note that the rjags package does not include a copy of the JAGS library: you must install this separately. For instructions on downloading JAGS, see the home page at `https://mcmc-jags.sourceforge.io`.

To fully understand how JAGS works, you need to read the JAGS User Manual. The manual explains the basics of modelling with JAGS and shows the functions and distributions available in the dialect of the BUGS language used by JAGS. It also describes the command line interface. The **rjags** package does not use the command line interface but provides equivalent functionality using R functions.

Analysis using the **rjags** package proceeds in steps:

1. Define the model using the BUGS language in a separate file.
2. Read in the model file using the `jags.model` function. This creates an object of class "jags".
3. Update the model using the `update` method for "jags" objects. This constitutes a 'burn-in' period.
4. Extract samples from the model object using the `coda.samples` function. This creates an object of class "mcmc.list" which can be used to summarize the posterior distribution. The **coda** package also provides convergence diagnostics to check that the output is valid for analysis (see Plummer et al 2006).

Author(s)

Martyn Plummer

References

Lunn D, Spiegelhalter D, Thomas A, Best N. (2009) The BUGS project: Evolution, critique and future directions. *Statistics in Medicine*, **28**:3049-67.

Plummer M, Best N, Cowles K, Vines K (2006). CODA: Convergence Diagnosis and Output Analysis for MCMC, *R News*, **6**:7-11.

`adapt` *Adaptive phase for JAGS models*

Description

Update the model in adaptive mode.

Usage

```
adapt(object, n.iter, end.adaptation=FALSE, ...)
```

Arguments

`object`	a `jags` model object
`n.iter`	length of the adaptive phase
`end.adaptation`	logical flag. If `TRUE` then adaptive mode will be turned off on exit.
`...`	additional arguments to the update method

Details

This function is not normally called by the user. It is called by the `jags.model` function when the model object is created.

When a JAGS model is compiled, it may require an initial sampling phase during which the samplers adapt their behaviour to maximize their efficiency (e.g. a Metropolis-Hastings random walk algorithm may change its step size). The sequence of samples generated during this adaptive phase is not a Markov chain, and therefore may not be used for posterior inference on the model.

The `adapt` function updates the model for `n.iter` iterations in adaptive mode. Then each sampler reports whether it has acheived optimal performance (e.g. whether the rejection rate of a Metropolis-Hasting sampler is close to the theoretical optimum). If any sampler reports failure of this test then `adapt` returns `FALSE`.

If `end.adaptation = TRUE`, then adaptive mode is turned off on exit, and further calls to `adapt()` do nothing. The model may be maintained in adaptive mode with the default option `end.adaptation = FALSE` so that successive calls to `adapt()` may be made until adaptation is satisfactory.

Value

Returns `TRUE` if all the samplers in the model have successfully adapted their behaviour to optimum performance and `FALSE` otherwise.

Author(s)

Martyn Plummer

`coda.samples` *Generate posterior samples in mcmc.list format*

Description

This is a wrapper function for `jags.samples` which sets a trace monitor for all requested nodes, updates the model, and coerces the output to a single `mcmc.list` object.

Usage

```
coda.samples(model, variable.names, n.iter, thin = 1, na.rm=TRUE, ...)
```

Arguments

`model`	a jags model object
`variable.names`	a character vector giving the names of variables to be monitored
`n.iter`	number of iterations to monitor
`thin`	thinning interval for monitors
`na.rm`	logical flag that indicates whether variables containing missing values should be omitted. See details.
`...`	optional arguments that are passed to the update method for jags model objects

Details

If `na.rm=TRUE` (the default) then elements of a variable that are missing (`NA`) for any iteration in at least one chain will be dropped.

This argument was added to handle incompletely defined variables. From JAGS version 4.0.0, users may monitor variables that are not completely defined in the BUGS language description of the model, e.g. if `y[i]` is defined in a `for` loop starting from `i=3` then `y[1]`, `y[2]` are not defined. The user may still monitor variable `y` and the monitored values corresponding to `y[1]`, `y[2]` will have value `NA` for all iterations in all chains. Most of the functions in the **coda** package cannot handle missing values so these variables are dropped by default.

Value

An `mcmc.list` object.

Author(s)

Martyn Plummer

See Also

jags.samples

Examples

```
data(LINE)
LINE$recompile()
LINE.out <- coda.samples(LINE, c("alpha","beta","sigma"), n.iter=1000)
summary(LINE.out)
```

---

collapse *Collapse tagged arrays*

Description

Interface to the apply function for tagged arrays

Usage

```
collapse(object, drop.tags, keep.tags, FUN, tag.result=FALSE, ...)
```

Arguments

object	An array with a dimtags attribute
drop.tags	tags of dimensions to be dropped
keep.tags	tags of dimensions to be preserved
FUN	the function to be applied
tag.result	binary indicator. If TRUE then the result will also be a tagged array if possible
...	additional arguments passed to FUN

Details

Using the apply function to simplify high-dimensional arrays can be complicated. This function provides a simplified interface for tagged arrays with two advantages over the basic apply function:

1. Dimensions are referred to by name (dimtags), not by number.
2. The user can define ether the dimensions to be kept or the dimensions to be dropped (but not both).

Value

Usually the output is a lower-dimensional array (vector, matrix, or array). If the output cannot be simplified because FUN returns values of varying length, the output will be a list.

If tag.result=TRUE, then the collapse function will add dimtags to the result if it can. This is generally only possible if FUN returns a scalar and the output is not a vector (which has no dim attribute).

Tagging the result allows a further call to the collapse function. For example, if MCMC output is represented as a tagged array, we may wish to calculate the posterior mean within each chain, and then the variance of the means between chains.

Author(s)

Martyn Plummer

See Also

dimtags, apply

---

control *Advanced control over JAGS*

Description

JAGS modules contain factory objects for samplers, monitors, and random number generators for a JAGS model. These functions allow fine-grained control over which factories are active.

Usage

```
list.factories(type)
set.factory(name, type, state)
```

Arguments

name	name of the factory to set
type	type of factory to query or set. Possible values are "sampler", "monitor", or "rng"
state	a logical. If TRUE then the factory will be active, otherwise the factory will become inactive.

Value

list.factories returns a data frame with two columns, the first column shows the names of the factory objects in the currently loaded modules, and the second column is a logical vector indicating whether the corresponding factory is active or not.

set.factory is called to change the future behaviour of factory objects. If a factory is set to inactive then it will be skipped.

Note

When a module is loaded, all of its factory objects are active. This is also true if a module is unloaded and then reloaded.

Author(s)

Martyn Plummer

Examples

```
list.factories("sampler")
list.factories("monitor")
list.factories("rng")
set.factory("base::Slice", "sampler", FALSE)
list.factories("sampler")
set.factory("base::Slice", "sampler", TRUE)
```

---

dic.samples *Generate penalized deviance samples*

Description

Function to extract random samples of the penalized deviance from a jags model.

Usage

```
dic.samples(model, n.iter, thin = 1, type, ...)
```

Arguments

model	a jags model object
n.iter	number of iterations to monitor
thin	thinning interval for monitors
type	type of penalty to use
...	optional arguments passed to the update method for jags model objects

Details

The dic.samples function generates penalized deviance statistics for use in model comparison. The two alternative penalized deviance statistics generated by dic.samples are the deviance information criterion (DIC) and the penalized expected deviance. These are chosen by giving the values "pD" and "popt" respectively as the type argument.

DIC (Spiegelhalter et al 2002) is calculated by adding the "effective number of parameters" (pD) to the expected deviance. The definition of pD used by dic.samples is the one proposed by Plummer (2002) and requires two or more parallel chains in the model.

DIC is an approximation to the penalized plug-in deviance, which is used when only a point estimate of the parameters is of interest. The DIC approximation only holds asymptotically when the effective number of parameters is much smaller than the sample size, and the model parameters have a normal posterior distribution.

The penalized expected deviance (Plummer 2008) is calculated by adding the optimism (`popt`) to the expected deviance. The `popt` penalty is at least twice the size of the `pD` penalty, and penalizes complex models more severely.

Value

An object of class "dic". This is a list containing the following elements:

`deviance`	A numeric vector, with one element for each observed stochastic node, containing the mean deviance for that node
`penalty`	A numeric vector, with one element for each observed stochastic node, containing an estimate of the contribution towards the penalty
`type`	A string identifying the type of penalty: "pD" or "popt"

Note

The `popt` penalty is estimated by importance weighting, and may be numerically unstable.

Author(s)

Martyn Plummer

References

Spiegelhalter, D., N. Best, B. Carlin, and A. van der Linde (2002), Bayesian measures of model complexity and fit (with discussion). *Journal of the Royal Statistical Society Series B* **64**, 583-639.

Plummer, M. (2002), Discussion of the paper by Spiegelhalter et al. *Journal of the Royal Statistical Society Series B* **64**, 620.

Plummer, M. (2008) Penalized loss functions for Bayesian model comparison. *Biostatistics* doi: 10.1093/biostatistics/kxm049

See Also

`diffdic`

`diffdic` *Differences in penalized deviance*

Description

Compare two models by the difference of two `dic` objects.

Usage

```
dic1 - dic2
diffdic(dic1, dic2)
```

Arguments

`dic1`, `dic2`	Objects inheriting from class "dic"

Details

A `diffdic` object represents the difference in penalized deviance between two models. A negative value indicates that `dic1` is preferred and vice versa.

Value

An object of class "diffdic". This is a numeric vector with an element for each observed stochastic node in the model.

The `diffdic` class has its own print method, which will display the sum of the differences, and its sample standard deviation.

Note

The problem of determining what is a noteworthy difference in DIC (or other penalized deviance) between two models is currently unsolved. Following the results of Ripley (1996) on the Akaike Information Criterion, Plummer (2008) argues that there is no absolute scale for comparison of two penalized deviance statistics, and proposes that the difference should be calibrated with respect to the sample standard deviation of the individual contributions from each observed stochastic node.

Author(s)

Martyn Plummer

References

Ripley, B. (1996) *Statistical Pattern Recognition and Neural Networks.* Cambridge University Press.

Plummer, M. (2008) Penalized loss functions for Bayesian model comparison. *Biostatistics* doi: 10.1093/biostatistics/kxm049

See Also

dic

---

dimtags *Get and set dimtags for a tagged array*

Description

Dimtags are designed to make working with high-dimensional arrays simpler by providing names for each dimension. We use the word "tags" instead of names for two reasons

1. To avoid confusion with dimnames which already exists and means something different
2. Because more than dimension can have the same tag. So tags do not uniquely identify each dimension

This infrastructure was put in place to support objects of class `mcarray`, which are tagged arrays with some additional attributes. The primary use of tagging is to simplify the manipulation of high-dimensional arrays using the `collapse` function instead of `apply`.

Usage

```
dimtags(x)
dimtags(x) <- value
```

Arguments

`x`	A matrix, array, or any object with a `dim` attribute
`value`	A character vector of tags, which must match the number of dimensions

Value

A character vector of dimtags. The replacement function returns the modified array.

See Also

`collapse`, `mcarray.object`,

jags.model *Create a JAGS model object*

Description

jags.model is used to create an object representing a Bayesian graphical model, specified with a BUGS-language description of the prior distribution, and a set of data.

Usage

```
jags.model(file, data=NULL, inits, n.chains = 1, n.adapt=1000,
           quiet=FALSE, n.threads = n.chains, initrep=10000)
```

Arguments

file	the name of the file containing a description of the model in the JAGS dialect of the BUGS language. Alternatively, file can be a readable text-mode connection, or a complete URL.
data	a list or environment containing the data. Any numeric objects in data corresponding to node arrays used in file are taken to represent the values of observed nodes in the model.
inits	optional specification of initial values in the form of a list or a function (see Initialization below). If omitted, initial values will be generated automatically. It is an error to supply an initial value for an observed node.
n.chains	the number of parallel chains for the model
n.adapt	the number of iterations for adaptation. See adapt for details. If n.adapt = 0 then no adaptation takes place.
quiet	if TRUE then messages generated during compilation will be suppressed, as well as the progress bar during adaptation.
n.threads	the number of threads to use for parallel updating
initrep	Number of replicate samples to draw from the prior when initializing the unobserved stochastic nodes. Set below.

Value

jags.model returns an object inheriting from class jags which can be used to generate dependent samples from the posterior distribution of the parameters

An object of class jags is a list of functions that share a common environment. This environment encapsulates the state of the model, and the functions can be used to query or modify the model state.

ptr()	Returns an external pointer to an object created by the JAGS library
data()	Returns a list containing the data that define the observed nodes in the model
model()	Returns a character vector containing the BUGS-language representation of the model

`state(internal=FALSE)`
: Returns a list of length equal to the number of parallel chains in the model. Each element of the list is itself a list containing the current parameter values in that chain. if `internal=TRUE` then the returned lists also include the RNG names (`.RNG.name`) and states (`.RNG.state`). This is not the user-level interface: use the `coef.jags` method instead.

`update(n.iter)`
: Updates the model by `n.iter` iterations. This is not the user-level interface: use the `update.jags` method instead.

Initialization

Initial values may be supplied for unobserved stochastic nodes via the `inits` argument. There are various ways to specify initial values:

1. A list of numeric values. Initial values for a single chain may supplied as a named list of numeric values. If there are multiple parallel chains then the same list is re-used for each chain.
2. A list of lists. Distinct initial values for each chain may be given as a list of lists. In this case, the list should have the same length as the number of chains in the model.
3. A function. A function may be supplied that returns a list of initial values. The function is called repeatedly to generate initial values for each chain. Normally this function should call some random number generating functions so that it returns different values every time it is called. The function should either have no arguments, or have a single argument named `chain`. In the latter case, the supplied function is called with the chain number as argument. In this way, initial values may be generated that depend systematically on the chain number.

If initial values are not supplied then they will be generated automatically. This process is controlled by the `initrep` argument. Setting `initrep=1` draws initial values from the prior distribution. For `initrep > 1` the initial value is a summary of `initrep` replicate samples from the prior:

- For unbounded continuous parameters, the initial values is the sample mean.
- For discrete and bounded parameters, the initial value is the sample with the highest prior density.

As `initrep` increases these initial values tend to the prior mean and prior mode respectively. The default value `initrep=10000` is designed for maximal compatibility with earlier versions of JAGS which set initial values to a deterministic "typical value" of the prior distribution. A large value of `initrep` is necessary when the model contains diffuse priors, such as a normal distribution with very low precision. It can be reduced for informative or weakly-informative priors.

For more fine-grained control over initialization, the `initrep` argument may be a vector of length 2. See the JAGS manual for details.

Random number generators

Each chain in a model has its own random number generator (RNG). RNGs and their initial seed values are assigned automatically when the model is created. The automatic seeds are calculated from the current time.

If you wish to make the output from the model reproducible, you may specify the RNGs to be used for each chain, and their starting seeds as part of the `inits` argument (see `Initialization`

above). This is done by supplementing the list of initial parameter values for a given chain with two additional elements named ".RNG.name", and ".RNG.seed":

`.RNG.name` a character vector of length 1. The names of the RNGs supplied in the base module are:

- "base::Wichmann-Hill"
- "base::Marsaglia-Multicarry"
- "base::Super-Duper"
- "base::Mersenne-Twister"

If the lecuyer module is loaded, it provides "lecuyer::RngStream"

`.RNG.seed` a numeric vector of length 1 containing an integer value.

Note that it is also possible to specify ".RNG.state" rather than ".RNG.seed" - see for example the output of `parallel.seeds`

Author(s)

Martyn Plummer

`jags.module` *Dynamically load JAGS modules*

Description

A JAGS module is a dynamically loaded library that extends the functionality of JAGS. These functions load and unload JAGS modules and show the names of the currently loaded modules. The load.pkg.module function is provided specifically for use from other R packages containing JAGS modules and checks for binary compatibility between rjags and the indicated package (given by the name argument) before loading the module embedded within the package. Note that this would normally be done in the .onAttach function.

Usage

```
load.module(name, path, quiet=FALSE)
load.pkg.module(pkgname, libname, bin.compat, quiet=FALSE,
  check.only=FALSE)
unload.module(name, quiet=FALSE)
list.modules()
```

Arguments

`name`	name of the module to be loaded
`path`	file path to the location of the DLL. If omitted, the option `jags.moddir` is used to locate the modules
`quiet`	a logical. If `TRUE`, no message will be printed about loading the module
`pkgname`	name of the R package containing the module to be loaded

libname	path to the library directory where the R package can be found
bin.compat	a list generated by pkg.compile.flags at build time by the package containing the JAGS module to be loaded
check.only	a logical. If TRUE, binary compatibility checks will be performed but the module will not be loaded (and the package named by pkgname need not exist). This is useful for testing purposes.

Author(s)

Martyn Plummer

Examples

```
list.modules()
load.module("glm")
list.modules()
unload.module("glm")
list.modules()
```

---

jags.object *Functions for manipulating jags model objects*

Description

A jags object represents a Bayesian graphical model described using the BUGS language.

Usage

```
## S3 method for class 'jags'
coef(object, chain=1, ...)
## S3 method for class 'jags'
variable.names(object, ...)
list.samplers(object)
```

Arguments

object	a jags model object
chain	chain number to query
...	additional arguments to the call (ignored)

Value

The `coef` function returns a list with an entry for each Node array that contains an unobserved Node. Elements corresponding to observed Nodes or deterministic Nodes are given missing values.

The `variable.names` function returns a character vector of names of node arrays used in the model.

The `list.samplers` function returns a named list with an entry for each Sampler used by the model. Each list element is a character vector containing the names of stochastic Nodes that are updated together in a block. The names of the list elements indicate the sampling methods that are used to update each block. Stochastic nodes that are updated by forward sampling from the prior are not listed.

Author(s)

Martyn Plummer

Examples

```
data(LINE)
LINE$recompile()
coef(LINE)
variable.names(LINE)
list.samplers(LINE)
```

`jags.samples` *Generate posterior samples*

Description

Function to extract random samples from the posterior distribution of the parameters of a `jags` model.

Usage

```
jags.samples(model, variable.names, n.iter, thin = 1,
             stat="value", summary = "trace", simplify=2, ...)
```

Arguments

`model`	A jags model object
`variable.names`	A character vector giving the names of variables to be monitored
`n.iter`	number of iterations to monitor
`thin`	thinning interval for monitors
`stat`	character vector of statistics to be monitored
`summary`	character vector of summaries
`simplify`	scalar integer or logical flag that determines whether the return value will be simplified. See Value below.
`...`	optional arguments passed to the update method for jags model objects

Details

The jags.samples function creates monitors for the given variables, runs the model for n.iter iterations and returns the monitored samples.

Value

A nested list of mcarray objects is returned. The top level denotes the requested stat, the second level the summary, and the third level the monitored variable. Individual mcarray objects can be extrected with samples$stat$summary$variable substituting appropriate names for stat, summary, and variable, e.g. samples$value$trace$alpha.

In mathematical terms, the return value is a tree with mcarray objects as leaf nodes. If the root node of the tree is unary, i.e. it has only one child node, then we can simplify the return value by returning the next level of the tree. This is controlled by the simplify parameter. If simplify is an integer between 1 and 3 then it determines the maximum depth of the simplification. A value of 0 indicates no simplification. For backward compatibility with JAGS 4, the default value is simplify=2 meaning that levels corresponding to stat and summary will be simplified, if they are unary, but not value.

The simplify parameter may also be a logical flag where FALSE means no simplification and TRUE is equivalent to simplify=3. This is most useful when only a single stat/summary/variable combination is monitored, in which case a single mcarray object is returned instead of a nested list.

Author(s)

Martyn Plummer

See Also

jags.model, coda.samples

Examples

```
data(LINE)
LINE$recompile()
## With simplify=FALSE the output is a nested list of depth 3
LINE.samples <- jags.samples(LINE, c("alpha","beta","sigma"),
                             n.iter=1000, simplify=FALSE)
## Access the value trace monitor for node alpha
LINE.samples$value$trace$alpha

## With simplify=2, the top 2 levels of the list are removed
## because all monitors are of stat type "value" and summary type "trace"
LINE.samples <- jags.samples(LINE, c("alpha","beta","sigma"),
                             n.iter=1000, simplify=2)
## Simpler access to the value trace monitor for alpha
LINE.samples$alpha

## If we have monitors of the same stat type but different summary
## types then simplify will only remove the top level of the list,
## even if we request maximum simplification
LINE.samples <- jags.samples(LINE, "alpha",  n.iter=1000,
```

```
                                summary=c("mean","var"),
                                simplify=3)
    LINE.samples$mean$alpha
```

`jags.seed` *Set initial value for JAGS RNGs*

Description

Sets the random seed used by JAGS.

Usage

```
jags.seed(seed)
```

Arguments

`seed`	A single integer value.

Details

This function sets the seed for the pseudo-random number generators (RNGs) used by JAGS making subsequent simulations reproducible.

Once the random seed is set by `jags.seed`, the same seed will be applied to RNG factories in subsequently loaded modules. In practical terms this means that a call to `jags.seed` may come before or after a call to `load.module` and will have the same effect.

Note

The JAGS RNGs are independent of the RNGs used by R. A call to the base function `set.seed` has no effect on the RNGs used by JAGS, and vice versa.

Author(s)

Martyn Plummer

See Also

`list.factories` to find the names of available RNG factories.

jags.version *Get JAGS version*

Description

Function to find the version of the JAGS library linked to the rjags package.

Usage

```
jags.version()
```

Details

The rjags package links to the JAGS library when loaded. This function returns the version of the linked JAGS library.

In principle, the major version of the rjags package should match major version of JAGS. The jags.version() function also returns information on the minor version and release number of JAGS.

Value

An object of class "numeric_version".

Author(s)

Matt Denwood

See Also

numeric_version

Examples

```
jags.version()
```

line *Linear regression example*

Description

The LINE model is a trivial linear regression model with only 5 observations. It's main use is to allow automated checks of the **rjags** package.

Format

A jags.model object, which must be recompiled before use.

mcarray.object *Objects for representing MCMC output*

Description

An mcarray object is used by the jags.samples function to represent monitored output from a JAGS model. It is a tagged array with tags "value", "iteration" and "chain".

Usage

```
## S3 method for class 'mcarray'
summary(object, FUN, ...)
## S3 method for class 'mcarray'
print(x, ...)
## S3 method for class 'mcarray'
as.mcmc.list(x, na.rm=TRUE, ...)
```

Arguments

object, x	an mcarray object
FUN	a function to be used to generate summary statistics
na.rm	indicates whether elements with missing values should be removed
...	additional arguments to the call

Details

The coda package defines mcmc objects for representing output from an MCMC sampler, and mcmc.list for representing output from multiple parallel chains. These objects emphasize the time-series aspect of the MCMC output, but lose the original array structure of the variables they represent. The mcarray class attempts to rectify this by preserving the dimensions of the original node array defined in the JAGS model.

Objects of class mcarray are tagged arrays with a limited set of supported tags:

"value" dimensions corresponding to the dimension of the object in the JAGS model

"iteration" dimension corresponding to the evolution of the Markov chain over iterations

"chain" dimension corresponding to replication of the Markov chain over parallel chains

There can be multiple dimensions with tag "value" but at most one with tag "iteration" and at most one with tag "chain".

Value

The summary method for mcarray objects collapses the object over "chain" and "iteration" dimensions leaving "value" dimensions.

The print method prints the mean, collapsing over "chain" and "iteration" dimensions. It also includes information about the "stat" and "summary" of the JAGS monitor and the size of the collapsed dimensions.

The as.mcmc.list method coerces an mcarray to an mcmc.list object so that the diagnostics provided by the coda package can be applied to the MCMC output it represents.

Author(s)

Martyn Plummer

See Also

collapse

---

node.names *Utilities for extracting node names from JAGS models*

Description

These utility functions allow the user to obtain the names of nodes within a compiled JAGS model. The node.names function returns the names of all nodes within the graph (without array indices), and the observed.stochastic.nodes function returns only the node names and incidies that represent observed stochastic variables. The expand.name function converts a base array name (as returned by node.names) along with a provided vector of array dimensions into the vector of node names as returned by observed.stochastic.nodes and the coda.samples function.

Usage

```
node.names(model)
observed.stochastic.nodes(model)
expand.name(name, dimensions)
```

Arguments

model	a jags model object
name	a length-1 character string giving the variable name to expand
dimensions	an integer vector giving the dimensions of the variable

Author(s)

Matt Denwood

Examples

```
data(LINE)
LINE$recompile()
node.names(LINE)
observed.stochastic.nodes(LINE)
expand.name("Y", 5)
```

nthread *Get and set number of threads used by a JAGS model*

Description

`nthread()` returns the number of parallel threads used by a JAGS model.

Usage

```
nthread(model)
nthread(model) <- value
```

Arguments

`model`	a `jags` model object
`value`	A positive integer

Value

A scalar integer, or NULL.

See Also

`jags.model`,

parallel *Get initial values for parallel RNGs*

Description

On a multi-processor system, you may wish to run parallel chains using multiple `jags.model` objects, each running a single chain on a separate processor. This function returns a list of values that may be used to initialize the random number generator of each chain.

Usage

```
parallel.seeds(factory, nchain)
```

Arguments

`factory`	Name of the RNG factory to use.
`nchain`	Number of chains for which to initialize RNGs.

Value

`parallel.seeds` returns a list of RNG states. Each element is a list of length 2 with the following elements:

`.RNG.name`	The name of the RNG
`.RNG.state`	An integer vector giving the state of the RNG.

Note

To make the results of `parallel.seeds` reproducible, the function `jags.seed` should be called first.

Author(s)

Martyn Plummer

See Also

`jags.model`, section "Random number generators", for further details on RNG initialization; `list.factories` to find the names of available RNG factories. `jags.seed` to set the seed for the random number generators used by JAGS.

Examples

```
##The BaseRNG factory generates up to four distinct types of RNG. If
##more than 4 chains are requested, it will recycle the RNG types, but
##use different initial values
parallel.seeds("base::BaseRNG", 4)

## The lecuyer module provides the RngStream factory, which allows large
## numbers of independent parallel RNGs to be generated.
load.module("lecuyer")
list.factories(type="rng")
parallel.seeds("lecuyer::RngStream", 5);
```

---

pkg.compile.flags *Get package compile flags*

Description

Function to check binary compatibility for R packages implementing JAGS modules, as well as retrieving the necessary compile flags to link to the same version of JAGS that rjags itself uses.

Usage

```
pkg.compile.flags(type = c("ALL", "JAGS_ROOT", "JAGS_VERSION", "RJAGS_VERSION",
  "CPPFLAGS", "CXXFLAGS", "LIBS"), print=type!="ALL")
```

Arguments

type	the compile flag to display (see value)
print	by default the value is printed using cat (for easier use in Makevars files) except when the type is ALL

Details

This function is provided for authors of R packages implementing JAGS modules for use within the Makevars(.win/ucrt) files in order to ensure that these packages are binary compatible with rjags. See also load.pkg.module

Value

The function prints and returns (invisibly) one of the following character strings, depending on the type requested:

ALL A list of all the values below (default).

JAGS_ROOT The location of the JAGS installation that rjags is linked to.

JAGS_VERSION The version of JAGS linked to.

RJAGS_VERSION The version of the rjags package.

CPPFLAGS The required PKG_CPPFLAGS.

CXXFLAGS The required PKG_CXXFLAGS (may be empty).

LIBS The required PKG_LIBS.

Author(s)

Matt Denwood

Examples

```
pkg.compile.flags()
```

---

read.jagsdata *Read data files for jags models*

Description

Read data for a JAGS model from a file.

Usage

```
read.jagsdata(file)
read.bugsdata(file)
```

Arguments

file	name of a file containing a text repesentation of the data for a jags model

Details

The command line interface for JAGS reads data and initial values from a text file. The data format used for jags data files is the same as the R dump function. Thus the data values can be read into an R session using the source function, but this will create objects in the global environment. The read.jagsdata function is a simple wrapper that reads the data into a list instead.

OpenBUGS also reads data and initial values from a text file. The format of these files is described as "S-PLUS" format by the OpenBUGS authors. It superficially resembles the format used by the dput function (and in fact can be parsed by the dget function). However, in BUGS "S-PLUS" format, arrays are stored in row-major order instead of the column-major order used by R. The read.bugsdata function reads OpenBUGS "S-PLUS" format files and permutes the elements of arrays so that they appear in the correct order.

Either function returns a list which can be used as the data or inits argument of jags.model.

Value

A named list of numeric vectors or arrays.

Note

Earlier versions of the rjags package had a read.data function which read data in either format, but the function name was ambiguous (There are many data file format in R) so this is now deprecated.

Author(s)

Martyn Plummer

---

rjags-deprecated *Deprecated Functions in the rjags package*

Description

These functions are provided for compatibility with older versions of the rjags package and will soon be defunct.

Usage

```
read.data(file, format=c("jags","bugs"))
```

Arguments

file	name of a file containing a text repesentation of the data for a jags model
format	format of the data

Details

read.data with format="jags" is a deprecated synonym for read.jagsdata and with format="bugs" is a deprecated synonym for read.bugsdata.

---

update *Update jags models*

Description

Update the Markov chain associated with the model.

Usage

```
## S3 method for class 'jags'
update(object, n.iter=1, by, progress.bar, ...)
```

Arguments

object	a jags model object
n.iter	number of iterations of the Markov chain to run
by	refresh frequency for progress bar. See Details
progress.bar	type of progress bar. Possible values are "text", "gui", and "none". See Details.
...	additional arguments to the update method (ignored)

Details

Since MCMC calculations are typically long, a progress bar is displayed during the call to update. The type of progress bar is determined by the progress.bar argument. Type "text" is displayed on the R console. Type "gui" is a graphical progress bar in a new window. The progress bar is suppressed if progress.bar is "none" or NULL, if the update is less than 100 iterations, or if R is not running interactively.

The default progress bar type is taken from the option jags.pb.

The progress bar is refreshed every by iterations. The update can only be interrupted when the progress bar is refreshed. Therefore it is advisable not to set by to a very large value. By default by is either n.iter/50 or 100, whichever is smaller.

Value

The update method for jags model objects modifies the original object and returns NULL.

Author(s)

Martyn Plummer

`waic.samples` *Generate samples for the Widely Applicable Information Criterion*

Description

Function to extract random samples of the Widely Applicable Information Criterion (WAIC) from a `jags` model.

Usage

```
waic.samples(model, n.iter, thin = 1, nodes="_observed_", scale=2*n, trace=FALSE, ...)
```

Arguments

`model`	A jags model object.
`n.iter`	Number of iterations to monitor.
`thin`	Thinning interval for monitors.
`nodes`	character vector of variable names containing nodes to be monitored. A variable name can be the name of a whole node array, e.g. "x", or subset, e.g. "x[2:5,2]". The monitored nodes will be any nodes partially or completely contained in the given variable array or subset. The default value "_observed_" monitors all observed stochastic nodes.
`scale`	Scaling factor for WAIC. The default scaling by twice the sample size `n` gives a WAIC statistic comparable with AIC and DIC.
`trace`	Logical flag. If TRUE then a trace of the likelihood will be returned for each monitored node in addition to the WAIC table.
`...`	optional arguments passed to the update method for jags model objects

Details

The `waic.samples` function calculates the Widely Applicable Information Criterion (WAIC) (Watanabe 2010), a predictive information criterion for model choice based on singular learning theory (Watanabe 2009).

Suppose that $Y_1 \dots Y_n$ are exchangeable observations that are conditionally independent given parameters θ. The posterior predictive density $q(y)$ is defined as:

$$q(y) = \mathrm{E}_\theta \left[p(y \mid \theta) \mid Y \right]$$

The princple behind WAIC is that model fit should be measured by the Bayes generalization error BL_g, which is minus the expectation of the log predictive density for a new observation Y_{n+1}.

$$BL_g = \mathrm{E} \left[\log q(Y_{n+1}) \right]$$

where the expectation is over the true, unknown distribution of Y_{n+1}. In practice, this cannot be evaluated but we can approximate the Bayes generalization error by replacing the expectation with the empirical average over the observed data. This gives the Bayes training loss.

$$BL_t = -\frac{1}{n}\sum_{i=1}^{n}\log\left[q(Y_i)\right]$$

The training loss is a biased estimate of the generalization error. It systematically under-estimates the generalization error because it uses the data twice: once to evaluate the posterior predictive distribution and again to evaluate the expectation over the data.

Watanabe (2009, 2010) showed that the bias of the Bayes training loss can be removed asymptotically for large sample size n by adding a penalty p_W/n to give the WAIC:

$$\text{WAIC} = BL_t + \frac{p_W}{n}$$

where

$$p_W = \sum_{i=1}^{n}\text{Var}_\theta\left[\log p\left(Y_i \mid \theta\right) \mid Y\right]$$

The penalty p_W can be considered a measure of model complexity. In regular models with non-informative priors, p_W reduces to the dimension of the parameters θ.

It is common practice to use version of WAIC scaled by $2n$. This version of WAIC can be written:

$$WAIC = -2\sum_{i=1}^{n}\log\left[q(Y_i)\right] + 2p_W$$

In this form, WAIC is comparable to the Akaike Information Criterion (AIC) and the Deviance Information Criterion (Spiegelhalter et al 2002). Scaling of WAIC is controlled by the `scale` parameter.

Value

An object of class "JAGSwaic". This is a list containing the following elements:

`waic.table` A matrix with one column for each chain and 3 rows

training_loss The Bayes training loss multiplied by the `scale` parameter.

p_waic The WAIC penalty, measuring the dimension of the model.

waic Penalized version of the Bayes training loss estimating the Bayes generalization error.

`mcarray` A list of objects of class "mcarray" containing at least the following elements

likelihood_mean The posterior mean of the likelihood for each node. This is the posterior predictive density evaluated at the observed value.

loglikelihood_var The posterior variance of the likelihood for each node. This is the contribution of each node to the WAIC penalty `p_waic` and can also be considered a measure of local influence for each observation (Plummer 2025).

Optionally, if `trace=TRUE` the list includes

loglikelihood_trace Trace of the log likelihood for each monitored node. If there are a large number of nodes this may use a lot of memory.

Author(s)

Matt Denwood, Martyn Plummer

References

Plummer, M. (2025) Bayesian measures of leverage and influence. *Arxiv*, https://arxiv.org/html/2503.19996

Spiegelhalter, D., N. Best, B. Carlin, and A. van der Linde (2002), Bayesian measures of model complexity and fit (with discussion). *Journal of the Royal Statistical Society Series B* **64**, 583-639.

Watanabe, S (2009), Algebraic Geometry and Statistical Learning Theory, Cambridge University Press.

Watanabe, S (2010) Asymptotic Equivalence of Bayes Cross Validation and Widely Applicable Information Criterion in Singular Learning Theory, Journal of Machine Learning Research, 11, 116, 3571–3594.

See Also

dic.samples

Index

∗ **datasets**
  line, 18
∗ **file**
  read.jagsdata, 23
∗ **interface**
  jags.module, 13
  node.names, 20
∗ **manip**
  collapse, 5
  dimtags, 10
  nthread, 21
∗ **misc**
  rjags-deprecated, 24
∗ **models**
  adapt, 3
  coda.samples, 4
  control, 6
  dic.samples, 7
  diffdic, 9
  jags.model, 11
  jags.object, 14
  jags.samples, 15
  jags.seed, 17
  mcarray.object, 19
  parallel, 21
  update, 25
  waic.samples, 26
∗ **package**
  rjags-package, 2
∗ **utilities**
  jags.version, 18
  pkg.compile.flags, 22

adapt, 3, *11*
apply, *6*
as.mcmc.list.mcarray *(*mcarray.object*)*, 19

coda.samples, 4, *16*, *20*
coef.jags, *12*
coef.jags *(*jags.object*)*, 14
collapse, 5, *10*, *20*
control, 6

dic, *10*
dic *(*dic.samples*)*, 7
dic.samples, 7, *28*
diffdic, *8*, 9
dimtags, *6*, 10
dimtags<- *(*dimtags*)*, 10

expand.name *(*node.names*)*, 20

jags.model, 11, *16*, *21*, *22*
jags.module, 13
jags.object, 14
jags.samples, *5*, 15
jags.seed, 17, *22*
jags.version, 18

LINE *(*line*)*, 18
line, 18
list.factories, *17*, *22*
list.factories *(*control*)*, 6
list.modules *(*jags.module*)*, 13
list.samplers *(*jags.object*)*, 14
load.module *(*jags.module*)*, 13
load.pkg.module, *23*
load.pkg.module *(*jags.module*)*, 13

mcarray.object, *10*, 19

node.names, 20
nthread, 21
nthread<- *(*nthread*)*, 21

observed.stochastic.nodes *(*node.names*)*, 20

parallel, 21
parallel.seeds, *13*

pkg.compile.flags, *14*, 22
print.mcarray *(*mcarray.object*)*, 19

read.bugsdata, *25*
read.bugsdata *(*read.jagsdata*)*, 23
read.data *(*rjags-deprecated*)*, 24
read.jagsdata, 23, *25*
rjags *(*rjags-package*)*, 2
rjags-deprecated, 24
rjags-package, 2

set.factory *(*control*)*, 6
summary.mcarray *(*mcarray.object*)*, 19

unload.module *(*jags.module*)*, 13
update, 25
update.jags, *12*

variable.names.jags *(*jags.object*)*, 14

waic *(*waic.samples*)*, 26
waic.samples, 26