Package 'rjags'

February 20, 2016

Version 4-6

Date 2016-02-19

Title Bayesian Graphical Models using MCMC

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

SystemRequirements JAGS 4.x.y

URL http://mcmc-jags.sourceforge.net

Suggests tcltk

Description Interface to the JAGS MCMC library.

License GPL (== 2)

NeedsCompilation yes

Author Martyn Plummer [aut, cre],
Alexey Stukalov [ctb],
Matt Denwood [ctb]

Maintainer Martyn Plummer <plummerm@iarc.fr>

Repository CRAN

Date/Publication 2016-02-19 23:59:59

R topics documented:

`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 `http://mcmc-jags.sourceforge.net`.

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)
```

`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`

`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, inits,
           n.chains = 1, n.adapt=1000, quiet=FALSE)
```

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.

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

There are various ways to specify initial values for a JAGS model. If no initial values are supplied, then they will be generated automatically by JAGS. See the JAGS User Manual for details. Otherwise, the options are as follows:

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.

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.

Usage

```
load.module(name, path, quiet=FALSE)
unload.module(name, quiet=FALSE)
list.modules()
```

Arguments

name	name of the load 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 package

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,
             type="trace", ...)
```

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
`type`	type of monitor
`...`	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 list of mcarray objects, with one element for each element of the variable.names argument.

Author(s)

Martyn Plummer

See Also

jags.model, coda.samples

Examples

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

---

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 MCMC output from a JAGS model. It is an array with named dimensions, for which the dimensions "iteration" and "chain" have a special status

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, ...)
```

Arguments

object,x	an mcarray object
FUN	a function to be used to generate summary statistics
...	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.

Value

The summary method for mcarray objects applies the given function to the array, marginalizing the "chain" and "iteration" dimensions.

The print method applies the summary function with FUN=mean.

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

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

It is not yet possible to make the results of `parallel.seeds` reproducible. This will be fixed in a future version of JAGS.

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.

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", 3)

## 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);
```

`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

Index