

Package ‘rjags’

November 10, 2009

Version 1.0.3-12

Date 2009-11-09

Title Bayesian graphical models using MCMC

Author Martyn Plummer

Maintainer Martyn Plummer <plummer@iarc.fr>

Depends R (>= 2.7.0), coda (>= 0.13)

SystemRequirements jags (== 1.0.3)

Description Interface to the JAGS MCMC library

License GPL (== 2)

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

Repository CRAN

Date/Publication 2009-11-10 10:15:24

R topics documented:

adapt	2
coda.samples	3
dic.samples	3
diffdic	5
jags.model	6
jags.module	8
jags.object	9
jags.samples	10
mccarray.object	11
read.data	12
update	13
Index	14

`adapt`*Adaptive phase for JAGS models*

Description

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

Usage

```
adapt(object, n.iter, ...)
```

Arguments

<code>object</code>	a <code>jags</code> model object
<code>n.iter</code>	length of the adaptive phase
<code>...</code>	additional arguments to the update method

Details

This function is not normally called by the user.

When a JAGS model is compiled, it may require an initial sampling phase during which the samplers adapt their parameters to maximize their efficiency. 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. For this reason, the `adapt` function must be called before samples can be generated from the model using the `update` method. Normally, this is done by the `jags.model` function when the model object is created.

The `adapt` function can only be called once on a `jags` model object since it turns off the adaptive phase. Subsequent calls to `adapt` do nothing.

Value

This function modifies the original object and returns `NULL`

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, ...)
```

Arguments

<code>model</code>	a jags model object
<code>variable.names</code>	a character vector giving the names of variables to be monitored
<code>n.iter</code>	number of iterations to monitor
<code>thin</code>	thinning interval for monitors
<code>...</code>	optional arguments that are passed to the update method for jags model objects

Value

An `mcmc.list` object.

Author(s)

Martyn Plummer

See Also

[jags.samples](#), [mcmc.list](#)

`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, ...)
## S3 method for class 'dic':
as.mcmc(x)
```

Arguments

<code>model</code>	a jags model object
<code>n.iter</code>	number of iterations to monitor
<code>thin</code>	thinning interval for monitors
<code>type</code>	type of penalty to use
<code>x</code>	An object inheriting from class “dic”
<code>...</code>	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 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 always larger than the pD penalty, and penalizes complex models more severely.

Value

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

<code>deviance</code>	A list of <code>marray</code> objects, one for each observed stochastic node, containing samples of the deviance
<code>penalty</code>	A list of <code>marray</code> objects, one for each observed stochastic node, containing samples of the penalty function
<code>type</code>	A string identifying the type of penalty: “pD” or “popt”

An object of class `dic` can be coerced to an `mcmc` object using the `as.mcmc` generic function. The resulting `mcmc` object has two variables: the mean deviance over all chains and the penalty.

Note

The `popt` penalty is estimated by importance weighting, and may be numerically unstable. It is recommended to inspect the `dic` object after coercing it to a `mcmc` object using functions from the `coda` package.

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	<i>Differences in penalized deviance</i>
---------	------------------------------------------

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=sys.frame(sys.parent()), inits, n.chains = 1,
           n.adapt=1000, nchain)
```

Arguments

<code>file</code>	a file containing a description of the model in the JAGS dialect of the BUGS language
<code>data</code>	a list or environment containing the data. Any numeric objects in <code>data</code> corresponding to node arrays used in <code>file</code> are taken to represent the values of observed nodes in the model
<code>inits</code>	optional specification of initial values in the form of a list or a function (see <code>Initialization</code> below). If omitted, initial values will be generated automatically. It is an error to supply an initial value for an observed node.
<code>n.chains</code>	the number of parallel chains for the model
<code>n.adapt</code>	the number of iterations for adaptation. See adapt for details. If <code>n.adapt = 0</code> then no adaptation takes place.
<code>nchain</code>	deprecated argument for the number of parallel chains. Use <code>n.chain</code> instead. If both <code>n.chain</code> and <code>nchain</code> are supplied then the value of <code>nchain</code> is ignored

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 function 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.

<code>ptr()</code>	Returns an external pointer to an object created by the JAGS library
<code>data()</code>	Returns a list containing the data that defines the observed nodes in the model
<code>model()</code>	Returns a character vector containing the BUGS-language representation of the model
<code>state(internal=FALSE)</code>	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 <code>internal=TRUE</code> then the returned lists also include the RNG names (<code>.RNG.name</code>) and states (<code>.RNG.state</code>). This is not the user-level interface: use the <code>coef.jags</code> method instead.
<code>update(n.iter)</code>	Updates the model by <code>n.iter</code> iterations. This is not the user-level interface: use the <code>update.jags</code> 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 be 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 seeds are calculated from the current time, so that two models created by identical calls to `jags.model` will not, by default, produce the same output.

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”

Further RNGs may be available if other JAGS modules are loaded.

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

Author(s)

Martyn Plummer

jags.module

Dynamically load JAGS modules

Description

A JAGS module is a shared library that extends the functionality of JAGS. This function loads a vector named `modules`.

Usage

```
jags.module(names, path)
```

Arguments

<code>names</code>	a vector of names of the jags modules to be loaded
<code>path</code>	the file path to the location of the modules. If omitted, the option <code>jags.moddir</code> is used to locate the modules

Author(s)

Martyn Plummer

`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

<code>object</code>	a <code>jags</code> model object
<code>chain</code>	chain number to query
<code>...</code>	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` returns a named list with an entry for each Sampler used by the model. Each element of the list is a character vector containing the names of stochastic Nodes that are updated together in a block at each iteration. The names of the list elements indicate the sampling methods that are used to update each block of Nodes.

Author(s)

Martyn Plummer

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

<code>model</code>	a <code>jags</code> model object
<code>variable.names</code>	a character vector giving the names of variables to be monitored
<code>n.iter</code>	number of iterations to monitor
<code>thin</code>	thinning interval for monitors
<code>type</code>	type of monitor
<code>...</code>	optional arguments passed to the update method for <code>jags</code> 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 `marray` objects, with one element for each element of the `variable.names` argument.

Author(s)

Martyn Plummer

See Also

[jags.model](#), [coda.samples](#)

marray.object *Objects for representing MCMC output*

Description

An `marray` 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 'marray':  
summary(object, FUN, ...)  
## S3 method for class 'marray':  
print(x, ...)  
## S3 method for class 'marray':  
as.mcmc.list(x, ...)
```

Arguments

<code>object, x</code>	an <code>marray</code> object
<code>FUN</code>	a function to be used to generate summary statistics
<code>...</code>	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 `marray` class attempts to rectify this by preserving the dimensions of the original node array defined in the JAGS model.

Value

The `summary` method for `marray` 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 `marray` 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

`read.data`*Read data files for jags models*

Description

Read data for a JAGS model from a file.

Usage

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

Arguments

<code>file</code>	name of a file containing a text representation of the data for a jags model
<code>format</code>	format of the data. See Details below

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.data` function, with `format="jags"`, 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.data` function, with `format="bugs"` reads OpenBUGS "S-PLUS" format files and permutes the elements of arrays so that they appear in the correct order.

Either choice of format 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.

Author(s)

Martyn Plummer

update	<i>Update jags models</i>
--------	---------------------------

Description

Update the Markov chain associated with the model.

Usage

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

Arguments

<code>object</code>	a jags model object
<code>n.iter</code>	number of iterations of the Markov chain to run
<code>by</code>	refresh frequency for progress bar. See Details
<code>progress.bar</code>	type of progress bar. Possible values are "text", "gui", and "none". See Details.
<code>...</code>	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

*Topic **file**

read.data, 11

*Topic **interface**

jags.module, 8

*Topic **models**

adapt, 1

coda.samples, 2

dic.samples, 3

diffdic, 5

jags.model, 6

jags.object, 8

jags.samples, 9

mcarray.object, 10

update, 12

adapt, 1, 6

as.mcmc.dic (*dic.samples*), 3

as.mcmc.list.mcarray
(*mcarray.object*), 10

coda.samples, 2, 10

coef.jags, 7

coef.jags (*jags.object*), 8

dic, 6

dic (*dic.samples*), 3

dic.samples, 3

diffdic, 4, 5

jags.model, 6, 10

jags.module, 8

jags.object, 8

jags.samples, 3, 9

list.samplers (*jags.object*), 8

mcarray.object, 10

mcmc.list, 3

print.mcarray (*mcarray.object*), 10

read.data, 11

summary.mcarray (*mcarray.object*),
10

update, 12

update.jags, 7

variable.names.jags
(*jags.object*), 8