Package 'postpack'

December 21, 2022

Title Utilities for Processing Posterior Samples Stored in 'mcmc.lists'

Version 0.5.4

Description The aim of 'postpack' is to provide the infrastructure for a standardized workflow for 'mcmc.list' objects.
These objects can be used to store output from models fitted with Bayesian inference using 'JAGS', 'WinBUGS', 'OpenBUGS', 'NIMBLE', 'Stan', or even custom MCMC algorithms. Although the 'coda' R package provides
some methods for these objects, it is somewhat limited in easily performing post-processing tasks for
specific nodes. Models are ever increasing in their complexity and the number of tracked nodes, and oftentimes
a user may wish to summarize/diagnose sampling behavior for only a small subset of nodes at a time
for a particular question or figure. Thus, many 'postpack' functions support performing tasks on a subset of nodes, where the subset is specified with regular expressions. The functions in 'postpack' streamline the extraction, summarization, and diagnostics of specific monitored nodes after model fitting.
Further, because there is rarely only ever one model under consideration, 'postpack' scales efficiently
to perform the same tasks on output from multiple models simultaneously, facilitating rapid assessment
of model sensitivity to changes in assumptions.

Depends R (>= 3.5.0)

Imports stringr (>= 1.3.1), coda, mcmcse, abind

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.1

URL https://bstaton1.github.io/postpack/

BugReports https://github.com/bstaton1/postpack/issues

Suggests knitr, rmarkdown, rstan, R2WinBUGS, R2jags, R2OpenBUGS, nimble, rjags, jagsUI

VignetteBuilder knitr

NeedsCompilation no

Author Ben Staton [aut, cre] (<https://orcid.org/0000-0002-2342-3482>)

Maintainer Ben Staton <statonbe@gmail.com>

Repository CRAN

Date/Publication 2022-12-21 00:40:02 UTC

R topics documented:

`array_format` *Convert a vector to the array structure used in model*

Description

Use element names to place vector elements in the appropriate location of an array.

Usage

```
array_format(v)
```

Arguments

v	A vector with names indicating the index location of each element in a new array. See the details (particularly the example) for more about what this means.

Details

Suppose you have an AxB matrix in your model, and you would like to create an object that stores the posterior means in the same AxB matrix as found in the model. For an AxB matrix, this is not too difficult to do "by-hand". However, if there are also dimensions C, D, and E, missing values, etc. it becomes more difficult.

Value

An array with elements of v placed in the appropriate location based on their index names.

Note

Up to 10 dimensions are currently supported. Please submit an issue should you find that you need more dimensions.

Examples

```
# load example mcmc.list
data(cjs)

# find an array node from your model
match_params(cjs, "SIG")

# extract the posterior mean of it
SIG_mean = post_summ(cjs, "SIG")["mean",]

# note that it has element names
SIG_mean

# create a matrix with elements in the proper place
array_format(SIG_mean)
```

cjs *Example mcmc.list 1*

Description

An example of samples from a joint posterior distribution from a Cormack-Jolly-Seber model. **The specific context does not matter, this object is provided to show examples of 'postpack' functionality**.

Usage

```
cjs
```

Format

A `mcmc.list` object.

Source

Posterior samples generated from a model fitted to hypothetical data set. See `vignette("example-mcmclists")` on the context, model, and monitored parameters.

cjs_no_rho *Example mcmc.list 2*

Description

An example of samples from a joint posterior distribution from a Cormack-Jolly-Seber model. **The specific context does not matter, this object is provided to show examples of 'postpack' functionality**.

Usage

```
cjs_no_rho
```

Format

A `mcmc.list` object.

Source

This object stores samples from the same hypothetical example as for the `cjs` example object, with one small change to the model. The `rho` term that models correlation between slopes and intercepts was forced to be zero, rather than estimating it. Consult `vignette("example-mcmclists")` for more details.

density_plot *Create a density plot for a single desired node*

Description

Used by `diag_plots()`, not intended to be called separately

Usage

```
density_plot(post, param, show_diags = "if_poor_Rhat")
```

Arguments

`post`	A `mcmc.list` object.
`param`	A regular expression that matches a single element in the model. E.g., `"b0[1]"`, not `"b0"`. See `match_params()`.
`show_diags`	Control when to display numerical diagnostic summaries on plots. Must be one of `"always"`, `"never"`, or `"if_poor_Rhat"`. `"if_poor_Rhat"` (the default) will display the Rhat and effective MCMC samples if the Rhat statistic is greater than 1.1.

Value

A figure showing the posterior density, separated by chain.

Note

This is **not** a function users will generally use directly. Call `diag_plots()` instead.

diag_plots *Create MCMC diagnostic plots for nodes of interest*

Description

Allows quick visualization of posterior density and trace plots, **both** separated by chain, for the desired nodes of interest. Includes the ability to plot in the RStudio graphics device, an external device, or a PDF file. Further, with the auto settings, the dimensions of the plotting device scales to the job needed.

Usage

```
diag_plots(
  post,
  params,
  ext_device = FALSE,
  show_diags = "if_poor_Rhat",
  layout = "auto",
  dims = "auto",
  keep_percent = 1,
  save = FALSE,
  file = NULL,
  auto_escape = TRUE
)
```

Arguments

post	A mcmc.list object.
params	A vector of regular expressions specifying the nodes to match for plotting. Accepts multi-element vectors to match more than one node at a time. See match_params() and vignette("pattern-matching") for more details.
ext_device	Display plots in an external device rather than the active device? FALSE (the default) will plot in the active device (including RStudio window). TRUE will create a new graphics device.
show_diags	Control when to display numerical diagnostic summaries on plots. Must be one of "always", "never", or "if_poor_Rhat". "if_poor_Rhat" (the default) will display the Rhat and effective MCMC samples if the Rhat statistic is greater than 1.1.
layout	Control how parameter diagnostics are organized into "ROWSxCOLUMNS". For example, layout = "4x1" has 4 rows and 1 column of parameter diagnostics. Defaults to "auto", which selects between the only accepted options of "1x1", "2x1", "4x1", "4x2", and "5x3".
dims	Control the dimensions of the graphics device using "HEIGHTxWIDTH" in inches. For example, "5x7" would create a 5 inch tall and 7 inch wide plotting device. Defaults to "auto", which selects the dimensions that look nice when layout = "auto" as well.
keep_percent	Proportion (between 0 and 1) of samples to keep for trace plotting. Passed to post_thin().
save	Save the diagnostic plots in a PDF file? If so, specify file = "example.pdf" as well. Defaults to FALSE.
file	File name of a PDF file to save the plots to. Must include the ".pdf" extension. Saved to working directory by default, but can receive an absolute or relative file path as part of this argument.
auto_escape	Automatically escape "[" and "]" characters for pattern matching? See match_params() for details.

Value

A multi-panel figure showing the posterior density and trace plots for requested nodes. The device in which it is placed depends on the argument values.

Note

If saving as a pdf, these files can get very large with many samples and render slowly. The `keep_percent` argument is intended to help with this by thinning the chains at quasi-evenly spaced intervals.

See Also

`match_params()`, `density_plot()`, `trace_plot()`

Examples

```
if (interactive()) {
  #load example mcmc.list
  data(cjs)

  # use current device
  diag_plots(cjs, "B0")

  # use a new device
  diag_plots(cjs, "B0", ext_device = TRUE)

  # always show diagnostic summaries
  diag_plots(cjs, "B0", show_diags = "always")

  # use a different layout (leaving it as "auto" is usually best)
  diag_plots(cjs, c("sig", "b"), layout = "5x3")

  # save diagnostics for all nodes to a pdf file
  diag_plots(cjs, "", save = TRUE, file = "diags.pdf")
}
```

---

drop_index *Extract the base node name of a parameter*

Description

Removes square brackets, numbers, and commas that represent the index of the node element in question. Returns just the node name.

Usage

```
drop_index(params)
```

Arguments

params	Node names.

Value

A character vector with the same length as params, with no indices included. For example, "a[1]" becomes "a".

Note

This is **not** a function users will generally use directly.

---

get_params *Obtain the names of all nodes*

Description

Returns the names of all quantities stored in a mcmc.list object.

Usage

```
get_params(post, type = "base_only")
```

Arguments

post	A mcmc.list object.
type	Format of returned matches; only two options are accepted: • type = "base_only" (the default) to return only the unique node names (without indices). • type = "base_index" to return the node names with indices included.

Value

A character vector with all node names stored in the post object, formatted as requested by type.

Examples

```
 # load example mcmc.list
data(cjs)

# get only node names, no indices (default)
get_params(cjs, type = "base_only")

# get indices too, where applicable
get_params(cjs, type = "base_index")
```

id_mat *Extract chain and iteration IDs for each sample*

Description

Extract chain and iteration IDs for each sample

Usage

```
id_mat(post)
```

Arguments

post	A mcmc.list object.

Value

A matrix with columns "CHAIN" and "ITER".

Note

This is **not** a function users will generally use directly.

ins_regex_bracket *Insert escapes on regex brackets*

Description

Insert escapes on regex brackets

Usage

```
ins_regex_bracket(params)
```

Arguments

params	Node names.

Details

Searches the contents of a string for the occurrence of a square bracket or two, and inserts the necessary escapes for pattern matching via regular expressions.

Value

A character vector with all brackets escaped. For example, "a[1]" becomes "a\\[1\\]"

Note

This is **not** a function users will generally use directly.

---

ins_regex_lock *Insert the symbols to lock in a string for matching*

Description

To ensure that a regular expression will match exactly, it's necessary to specify so.

Usage

```
ins_regex_lock(params)
```

Arguments

params	Node names to paste a ^ and $ (if not already present) to lock in the match.

Value

A character vector with locking anchors inserted to force an exact match. For example, "a\\[1\\]" becomes "^a\\[1\\]$".

Note

This is **not** a function users will generally use directly.

---

list_out *List vector elements in a nice format*

Description

Converts a vector into a comma-separated list for use in sentences (error messages, warnings, etc.).

Usage

```
list_out(x, final = NULL, per_line = 1e+06, wrap = NULL, indent = NULL)
```

Arguments

x	A vector, will be coerced to a character.
final	Word that will separate the final element in the list from others. See the examples.
per_line	Number of elements printed per line. See the examples.
wrap	Optional character to wrap around each element, e.g., quotation marks.
indent	Optional string to place in front of the first element on each line. See the examples.

Value

A character vector with length == 1; ready to be passed to `base::stop()`, `base::warning()`, or `base::cat()`, to provide a useful message.

`match_params` *Find matching node names*

Description

Returns all the node names stored in a `mcmc.list` object that match a provided string.

Usage

```
match_params(post, params, type = "base_index", auto_escape = TRUE)
```

Arguments

`post`	A `mcmc.list` object.
`params`	A vector of regular expressions specifying the nodes to match. Accepts multi-element vectors to match more than one node at a time. See examples and `vignette("pattern-matching")` for more details.
`type`	Format of returned matches; only two options are accepted: • `type = "base_only"` to return only the unique node names (without indices). • `type = "base_index"` (the default) to return the node names with indices included.
`auto_escape`	Automatically escape `"["` and `"]"` characters for pattern matching? `FALSE` will treat `"["` and `"]"` as special regular expression characters (unless explicitly escaped by user), `TRUE` will treat these symbols as plain text to be matched. It is generally recommended to keep this as `TRUE` (the default), unless you are performing complex regex searches that require the `"["` and `"]"` symbols to be special characters.

Details

This function is called as one of the first steps in many of the more downstream functions in this package. It is thus fairly important to get used to how the regular expressions work. This function can be used directly to hone in on the correct regular expression. See the examples below.

Value

A character vector with all node names in `post` that match `params`, formatted as requested by `type`.. If no matches are found, an error will be returned with the base node names found in `post` to help the next try.

Examples

```
# load example mcmc.list
data(cjs)

# these produce same output b/c of regex pattern matching
match_params(cjs, params = c("b0", "b1"))
match_params(cjs, params = c("b"))

# return only base names, not indices as well
match_params(cjs, params = "b", type = "base_only")

# force a match to start with B
match_params(cjs, "^B")

# force a match to end with 0
match_params(cjs, "0$")

# use a wild card to get b0[3] and b1[3]
match_params(cjs, "b.[3]")

# repeat a wild card
match_params(cjs, "s.+0")

# turn off auto escape to use [] in regex syntax rather than matching them as text
match_params(cjs, params = "[:digit:]$", auto_escape = FALSE)

# pass a dot to match all (same as get_params)
match_params(cjs, ".")
```

---

mytitle *Add a title between two figures*

Description

Used by `diag_plots()` to place a common title over top of two figures: one density and one trace for a given node.

Usage

```
mytitle(text)
```

Arguments

`text`	The text string to include as a centered title over two adjacent plots.

Note

This is **not** a function users will generally use directly.

postpack *Utilities for Processing Posterior Samples Stored in mcmc.lists*

Description

The aim of 'postpack' is to provide the infrastructure for a standardized workflow for `mcmc.list` objects. These objects can be used to store output from models fitted with Bayesian inference using JAGS, Win/OpenBUGS, NIMBLE, Stan, or even custom MCMC algorithms (see `post_convert()` for converting samples to `mcmc.list` format). Although the 'coda' package provides some methods for these objects, it is somewhat limited in easily performing post-processing tasks for **particular nodes**. Models are ever increasing in their complexity and the number of tracked nodes, and oftentimes a user may wish to summarize/diagnose sampling behavior for only a small subset of nodes at a time for a particular question or figure. Thus, many 'postpack' functions support performing tasks on a subset of nodes, where the subset is specified with regular expressions. The functions in this package streamline the extraction, summarization, and diagnostics of particular nodes monitored after model fitting. Further, because there is rarely only ever one model under consideration, 'postpack' scales efficiently to perform the same tasks on output from multiple models simultaneously, facilitating rapid assessment of model sensitivity to changes in assumptions.

post_bind *Combine two objects containing posterior samples*

Description

Intended for use when derived quantities are calculated from monitored posterior samples, and you wish to combine them into the master `mcmc.list`, as though they were calculated and monitored during MCMC sampling. It is not advised to combine samples from two MCMC runs, because covariance of MCMC sampling would be lost.

Usage

```
post_bind(post1, post2, dup_id = "_p2")
```

Arguments

`post1`	A `mcmc.list` or `matrix` object.
`post2`	A `mcmc.list` or `matrix` object.
`dup_id`	If any node names are duplicated in `post2`, what should be appended to the end of these node names in the output? If this occurs a warning will be returned. Defaults to `"_p2"`

Details

Some important things to note:

- If the object passed to `post1` is a `matrix`, `post2` must be a `mcmc.list`, and vice versa.
- That is, two `mcmc.list` objects are allowed, but not two `matrix` objects.
- For `matrix` objects, nodes should be stored as columns and samples should be stored as rows. Column names should be present.
- The objects passed to `post1` and `post2` must have the same number of chains, iterations, burnin, and thinning interval.
- If the node names are empty (e.g., missing column names in a `matrix`), the node names will be coerced to `"var1"`, `"var2"`, etc. and a warning will be returned.

Value

A single `mcmc.list` object containing samples of the nodes from both `post1` and `post2`.

Examples

```
# load example mcmc.list
data(cjs)

# create two subsets from cjs: one as mcmc.list and one as matrix
# also works if both are mcmc.list objects
p1 = post_subset(cjs, "b0")
p2 = post_subset(cjs, "b1", matrix = TRUE)

# combine them into one mcmc.list
head(post_bind(p1, p2))
```

---

post_convert *Convert MCMC samples to mcmc.list format*

Description

Wrapper around several ways of converting objects to `mcmc.list` format, automated based on the input object class.

Usage

```
post_convert(obj)
```

Arguments

`obj`	An object storing posterior samples from an MCMC algorithm. Accepted classes are `list`, `matrix`, `stanfit`, `bugs`, `rjags`.

Details

Accepted classes are produced by several packages, including but probably not limited to:

- `stanfit` objects are created by `rstan::stan()`, which also provides `rstan::As.mcmc.list()`. Rather than requiring users to have 'rstan' installed to use 'postpack', `post_convert()` will instruct users to use this function if supplied a `stanfit` object.
- `bugs` objects are created by `R2WinBUGS::bugs()` and `R2OpenBUGS::bugs()`.
- `rjags` objects are created by `R2jags::jags()`.
- `list` objects are created by `nimble::runMCMC()`, 'MCMCpack' functions, or custom MCMC algorithms.
- `matrix` objects are created by `post_subset()` and is often the format of posterior quantities derived from monitored nodes.
- `mcmc.list` objects are created by `rjags::coda.samples()`, `jagsUI::jags.basic()`, and `jagsUI::jags()$samples`. If a `mcmc.list` object is passed to `obj`, an error will be returned telling the user this function is not necessary.

If you find that a critical class conversion is missing, please submit an issue requesting its addition with a minimum working example of how it can be created.

Value

The same information as passed in the `obj` argument, but formatted as `mcmc.list` class.

Note

- If samples are stored in a `list` object, the individual elements must be `matrix` or `mcmc` class, storing the samples (rows) across parameters (columns, with names) for each chain (`list` elements). If `list` elements are in `matrix` format, they will be coerced to `mcmc` format, and thinning, start, and end intervals may be inaccurate.
- If samples are stored in a `matrix` object, rows should store samples and columns should store nodes. Multiple chains should be combined using `base::rbind()`. Two additional columns **must** be present: `"CHAIN"` and `"ITER"`, which denote the MCMC chain and iteration numbers, respectively.

See Also

`coda::as.mcmc.list()`, `coda::as.mcmc()`

Examples

```
## EXAMPLE 1
# load example mcmc.list
data(cjs)

# take a subset from cjs as a matrix, retain chain and iter ids
cjs_sub = post_subset(cjs, "^B", matrix = TRUE, chains = TRUE, iters = TRUE)

# convert back to mcmc.list
```

```
class(post_convert(cjs_sub))

## EXAMPLE 2: create mcmc.list from hypothetical MCMC samples; chains are list elements
# create hypothetical samples; can't use postpack on this - not an mcmc.list
samps = lapply(1:3, function(i) {
  m = matrix(rnorm(100), 20, 5)
  colnames(m) = paste0("param", 1:5)
  m
})

# convert
samps_new = post_convert(samps)

# can use postpack now
post_summ(samps_new, "param")

## EXAMPLE 3: create mcmc.list from hypothetical MCMC samples; chains rbind-ed matrices
# create samples
f = function() {
  m = matrix(rnorm(100), 20, 5)
  colnames(m) = paste0("param", 1:5)
  m
}
samps = rbind(f(), f(), f())

# assign chain and iter IDs to each sample
samps = cbind(CHAIN = rep(1:3, each = 20), ITER = rep(1:20, 3), samps)

# convert
samps_new = post_convert(samps)

# can use postpack now
post_summ(samps_new, "param")
```

`post_dim` *Obtain MCMC dimensions from an mcmc.list*

Description

Quickly query the number of burn-in samples, post-burnin, thinning, number of chains, etc. from a `mcmc.list` object.

Usage

```
post_dim(post, types = NULL)
```

Arguments

`post`	A `mcmc.list` object.

types — The dimension types to return. Must contain some of "burn", "post_burn", "thin", "chains", "nodes". Defaults to NULL, in which case all of these are returned.

Value

A numeric vector with named elements, which may contain:

- burn: The burn-in period + adapting phase (per chain).
- post_burn: The post-burn-in period (per chain).
- thin: The thinning interval post-burn-in.
- chains: The number of chains.
- saved: The number of saved samples across all chains.
- params: The number of nodes with MCMC samples.

All of these will be returned if types = NULL, a subset can be returned by specifying (for example) types = c("burn", "thin").

Note

If the post object was thinned after MCMC completed using post_thin(), then the "burn" and "thin" dimensions will be improperly calculated. post_thin() performs post-MCMC thinning of mcmc.list objects, and is solely for developing long-running post-processing code, so this is okay.

Examples

```
# load example mcmc.list
data(cjs)

# get all relevant dimensions
post_dim(cjs)

# get only the number of chains
post_dim(cjs, "chains")

# get the thinning and burn-in intervals
post_dim(cjs, c("burn", "thin"))
```

---

post_remove *Remove nodes from mcmc.list*

Description

Just like post_subset(), but keep all nodes **except** those that match.

Usage

```
post_remove(post, params, ask = TRUE, auto_escape = TRUE)
```

Arguments

post	A mcmc.list object.
params	A vector of regular expressions specifying the nodes to match for removal. Accepts multi-element vectors to match more than one node at a time. See match_params() and vignette("pattern-matching") for more details.
ask	Prompt user for a response prior to removing nodes?
auto_escape	Automatically escape "[" and "]" characters? See match_params() for details.

Value

A mcmc.list, identical in all ways to the original except that nodes matched by the params argument are removed. If the user responds "no" to the question when ask = TRUE, post is returned unaltered.

Examples

```
# load example mcmc.list
data(cjs)

# get names of all nodes
get_params(cjs)

# remove the SIG nodes
new_cjs = suppressMessages(post_remove(cjs, "SIG", ask = FALSE))

# get names of new output
get_params(new_cjs)
```

---

post_subset *Extract samples from specific nodes*

Description

Subsets a smaller portion from a mcmc.list object corresponding only to the node(s) requested.

Usage

```
post_subset(
  post,
  params,
  matrix = FALSE,
  iters = FALSE,
  chains = FALSE,
  auto_escape = TRUE
)
```

Arguments

post	A mcmc.list object.
params	A vector of regular expressions specifying the nodes to match for subsetting. Accepts multi-element vectors to match more than one node at a time. See match_params() and vignette("pattern-matching") for more details.
matrix	Return samples in matrix rather than mcmc.list format?
iters	Retain the iteration number of each sample if matrix = TRUE? Not used otherwise.
chains	Retain the chain number of each sample if matrix = TRUE? Not used otherwise.
auto_escape	Automatically escape "[" and "]" characters for pattern matching? See match_params() for details.

Value

A mcmc.list or matrix object, depending on the value of the matrix argument. Object contains all nodes that match the params argument; an error will be returned if no matches are found.

See Also

match_params()

Examples

```
# load example mcmc.list
data(cjs)

# create mcmc.list with all nodes that contain "B0"
x1 = post_subset(cjs, "B0")

# create mcmc.list with all nodes that contain "b" or "B"
x2 = post_subset(cjs, c("b", "B"))

# perform the subset and return a matrix as output, while retaining the chain ID
x3 = post_subset(cjs, "B0", matrix = TRUE, chain = TRUE)
```

---

post_summ *Obtain posterior summaries and diagnostics of specific nodes*

Description

Allows rapid calculation of summaries and diagnostics from **specific nodes** stored in mcmc.list objects.

Usage

```
post_summ(
  post,
  params,
  digits = NULL,
  probs = c(0.5, 0.025, 0.975),
  Rhat = FALSE,
  neff = FALSE,
  mcse = FALSE,
  by_chain = FALSE,
  auto_escape = TRUE
)
```

Arguments

post	A mcmc.list object.
params	A vector of regular expressions specifying the nodes to match for summarization. Accepts multi-element vectors to match more than one node at a time. See match_params() and vignette("pattern-matching") for more details.
digits	Control rounding of summaries. Passed to base::round() and defaults to NULL, which produces no rounding.
probs	Posterior quantiles to calculate. Passed to stats::quantile(). Defaults to probs = c(0.5, 0.025, 0.975) (i.e., median and equal-tailed 95 percent credible interval).
Rhat	Calculate the Rhat convergence diagnostic using coda::gelman.diag()? Fair warning: this can take a bit of time to run on many nodes/samples.
neff	Calculate the number of effective MCMC samples using coda::effectiveSize()? Fair warning: this can take a bit of time to run on many nodes/samples.
mcse	Calculate the Monte Carlo standard error for the posterior mean and reported quantiles using the mcmcse::mcse() and mcmcse::mcse.q() functions (batch means method with batch size automatically calculated)? Fair warning: this can take a bit of time to run on many nodes/samples.
by_chain	Calculate posterior summaries for each chain rather than for the aggregate across chains? Defaults to FALSE. The arguments Rhat, neff, and mcse are ignored if by_chain = TRUE and a warning will be returned.
auto_escape	Automatically escape "[" and "]" characters for pattern matching? See match_params() for details.

Value

A matrix object with summary statistics as rows and nodes as columns. If by_chain = TRUE, an array with chain-specific summaries as the third dimension is returned instead.

See Also

match_params(), coda::gelman.diag(), coda::effectiveSize(), mcmcse::mcse(), mcmcse::mcse.q()

Examples

```
# load example mcmc.list
data(cjs)

# calculate posterior summaries for the "p" nodes
# ("p[1]" doesn't exist in model)
post_summ(cjs, "p")

# do this by chain
post_summ(cjs, "p", by_chain = TRUE)

# calculate Rhat and Neff diagnostic summaries as well
# multiple node names too
post_summ(cjs, c("b0", "p"), Rhat = TRUE, neff = TRUE)

# calculate Monte Carlo SE for mean and quantiles, with rounding
post_summ(cjs, "p", mcse = TRUE, digits = 3)

# summarize different quantiles: median and central 80%
post_summ(cjs, "p", probs = c(0.5, 0.1, 0.9))
```

---

post_thin *Perform post-MCMC thinning*

Description

Removes iterations from each chain of a `mcmc.list` object at quasi-evenly spaced intervals. Post-MCMC thinning is useful for developing long-running post-processing code with a smaller but otherwise identical `mcmc.list`.

Usage

```
post_thin(post, keep_percent, keep_iters)
```

Arguments

`post`	A `mcmc.list` object.
`keep_percent`	Proportion (between 0 and 1) of samples to keep from each chain. Setting `keep_percent = 0.2` will remove approximately 80 percent of the samples.
`keep_iters`	Number of samples to keep from each chain.

Details

The samples will be removed at as evenly spaced intervals as possible, however, this is not perfect. It is therefore recommended to use the full posterior for final post-processing calculations, but this should be fine for most development of long-running code.

If both `keep_percent` and `keep_iters` are supplied, an error will be returned requesting that only one be used.

Value

A mcmc.list object, identical to post, but with fewer samples of each node.

Note

Iteration numbers are reset after thinning the samples. So if running post_dim() on output passed through post_thin(), you cannot trust the burn-in or thinning counts. Again, this is not an issue for developing post-processing code.

Examples

```
# load example mcmc.list
data(cjs)

# take note of original dimensions
post_dim(cjs)

# keep ~20% of the samples
cjs_thin1 = post_thin(cjs, keep_percent = 0.2)

# note burn-in and thin intervals no longer correct!
# but desired outcome achieved - identical object but smaller
post_dim(cjs_thin1)

# keep 30 samples per chain
cjs_thin2 = post_thin(cjs, keep_iters = 30)
post_dim(cjs_thin2)
```

---

rm_regex_bracket *Remove escapes on regex brackets*

Description

Remove escapes on regex brackets

Usage

```
rm_regex_bracket(params)
```

Arguments

params	Node names.

Details

Searches the contents of a string for the occurrence of a square bracket or two (that has been escaped), and removes the escaping that was necessary for matching via regular expressions.

Value

A character vector with all brackets escaped. For example, `"a\\[1\\]"` becomes `"a[1]"`.

Note

This is **not** a function users will generally use directly.

---

rm_regex_lock *Remove the symbols that lock in a string for matching*

Description

Undoes the work of `ins_regex_lock()`.

Usage

```
rm_regex_lock(params)
```

Arguments

`params`	Node names to remove a `^` and `$` from (if present).

Value

A character vector with locking anchors inserted to force an exact match. For example, `"^a\\[1\\]$"` becomes `"a\\[1\\]"`.

Note

This is **not** a function users will generally use directly.

---

trace_plot *Create a trace plot for a single desired node*

Description

Create a trace plot for a single desired node

Usage

```
trace_plot(post, param, keep_percent = 1)
```

Arguments

post	A mcmc.list object.
param	A regular expression that matches a single element in the model. E.g., "b0[1]", not "b0". See match_params().
keep_percent	A numeric vector of length == 1 and on the range (0,1]. Percent of samples you'd like to keep for trace plotting and passed to post_thin().

Note

If saving as a pdf file, these files can get very large with many samples and render slowly. The keep_percent argument is intended to help with this by thinning the chains at quasi-evenly spaced intervals. This is **not** a function users will generally use directly. Call diag_plots() instead.

---

vcov_decomp *Decompose the posterior of a variance-covariance node*

Description

For each posterior sample, extract the standard deviation and correlation components of a monitored node representing a variance-covariance matrix.

Usage

```
vcov_decomp(
  post,
  param,
  sigma_base_name = "sigma",
  rho_base_name = "rho",
  invert = FALSE,
  check = TRUE,
  auto_escape = TRUE
)
```

Arguments

post	A mcmc.list object.
param	A vector of regular expressions specifying the nodes to match for plotting. Must match only one base node name in post, and that node must store samples from a matrix within the model. See match_params() and vignette("pattern-matching") for more details.
sigma_base_name	Base node name to assign to the standard deviation vector component? Defaults to "sigma", which becomes "sigma[1]", "sigma[2]", etc. in the output.
rho_base_name	Same as sigma_base_name, but for the correlation matrix component.

invert	Take the inverse of the matrix node matched by param prior to performing the calculations? This would be necessary if the matrix node was expressed as a precision matrix as used in the BUGS language. Triggers a call to base::solve().
check	Perform checks sequentially that the matrix node is (a) square, (b) symmetrical, and (c) positive definite before proceeding with the calculations? If set to FALSE, unexpected output may be returned or other errors related to items a, b, and c may be triggered - this is not advised, though may be required if wishing to set invert = TRUE.
auto_escape	Automatically escape "[" and "]" characters for pattern matching? See match_params() for details.

Value

A mcmc.list object.

Examples

```
# load example mcmc.list
data(cjs)

# "SIG" is a covariance matrix node
SIG_decomp = vcov_decomp(cjs, "SIG")

# extract the posterior mean correlation matrix, and reformat
array_format(post_summ(SIG_decomp, "rho")["mean",])
```

---

write_model *Export BUGS/JAGS model from function to file*

Description

Performs the same basic function as R2OpenBUGS::write.model()

Usage

```
write_model(fun, file)
```

Arguments

fun	A function object containing BUGS/JAGS model code
file	A character vector of length == 1: the name of the file to write to

Details

Performs the same basic function as R2OpenBUGS::write.model(), but with slightly better output (scientific notation, spacing, etc.). The main reason it was created for use in 'postpack' was to remove the need for using the 'R2OpenBUGS' package when not using OpenBUGS.

Value

Nothing, but `file` is written to disk.

Examples

```
if (interactive()) {
  # define some simple BUGS model as an R function
  # note the use of %_% to include a truncation
  mod = function() {
    # PRIORS
    mu ~ dnorm(0,0.001) %_% T(0,)
    sig ~ dunif(0,10)
    tau <- 1/sig^2

    # LIKELIHOOD
    for (i in 1:n) {
      y[i] ~ dnorm(mu, tau)
    }
  }

  # write model to a text file to be called by BUGS/JAGS
  write_model(mod, "model.txt")
}
```

Index