

# Package ‘coda’

April 17, 2009

**Version** 0.13-4

**Date** 2009-01-25

**Title** Output analysis and diagnostics for MCMC

**Author** Martyn Plummer, Nicky Best, Kate Cowles, Karen Vines

**Maintainer** Martyn Plummer <plummer@iarc.fr>

**Depends** R (>= 2.5.0), lattice

**Description** Output analysis and diagnostics for Markov Chain Monte Carlo simulations.

**License** GPL (>= 2)

**Repository** CRAN

**Date/Publication** 2009-01-26 08:52:52

## R topics documented:

as.ts.mcmc . . . . .	2
autocorr . . . . .	3
autocorr.diag . . . . .	4
autocorr.plot . . . . .	4
batchSE . . . . .	5
bugs2jags . . . . .	6
coda.options . . . . .	7
codamenu . . . . .	8
Cramer . . . . .	8
crosscorr . . . . .	9
crosscorr.plot . . . . .	10
cumuplot . . . . .	10
densplot . . . . .	11
effectiveSize . . . . .	12
gelman.diag . . . . .	12
gelman.plot . . . . .	14

geweke.diag . . . . .	16
geweke.plot . . . . .	17
heidel.diag . . . . .	18
HPDinterval . . . . .	19
line . . . . .	20
mcmc . . . . .	20
mcmc.convert . . . . .	21
mcmc.list . . . . .	22
mcmc.subset . . . . .	23
mcmcUpgrade . . . . .	24
mcpair . . . . .	24
multi.menu . . . . .	25
nchain . . . . .	25
plot.mcmc . . . . .	26
raftery.diag . . . . .	27
read.and.check . . . . .	28
read.coda . . . . .	29
read.coda.interactive . . . . .	30
read.openbugs . . . . .	31
rejectionRate . . . . .	32
spectrum0 . . . . .	32
spectrum0.ar . . . . .	34
summary.mcmc . . . . .	35
thin . . . . .	35
time.mcmc . . . . .	36
traceplot . . . . .	37
trellisplots . . . . .	37
varnames . . . . .	41
window.mcmc . . . . .	42

<b>Index</b>	<b>43</b>
--------------	-----------

---

as.ts.mcmc	<i>Coerce mcmc object to time series</i>
------------	--

---

## Description

as.ts.mcmc will coerce an mcmc object to a time series.

## Usage

```
as.ts.mcmc(x, ...)
```

## Arguments

x	an mcmc object
...	unused arguments for compatibility with generic as.ts

**Author(s)**

Martyn Plummer

**See Also**[as.ts](#)

---

`autocorr`*Autocorrelation function for Markov chains*

---

**Description**

`autocorr` calculates the autocorrelation function for the Markov chain `mcmc.obj` at the lags given by `lags`. The lag values are taken to be relative to the thinning interval if `relative=TRUE`.

High autocorrelations within chains indicate slow mixing and, usually, slow convergence. It may be useful to thin out a chain with high autocorrelations before calculating summary statistics: a thinned chain may contain most of the information, but take up less space in memory. Re-running the MCMC sampler with a different parameterization may help to reduce autocorrelation.

**Usage**

```
autocorr(x, lags = c(0, 1, 5, 10, 50), relative=TRUE)
```

**Arguments**

<code>x</code>	an mcmc object
<code>lags</code>	a vector of lags at which to calculate the autocorrelation
<code>relative</code>	a logical flag. TRUE if lags are relative to the thinning interval of the chain, or FALSE if they are absolute difference in iteration numbers

**Value**

A vector or array containing the autocorrelations.

**Author(s)**

Martyn Plummer

**See Also**[acf](#), [autocorr.plot](#).

---

autocorr.diag      *Autocorrelation function for Markov chains*

---

### Description

`autocorr.diag` calculates the autocorrelation function for the Markov chain `mcmc.obj` at the lags given by `lags`. The lag values are taken to be relative to the thinning interval if `relative=TRUE`. Unlike `autocorr`, if `mcmc.obj` has many parameters it only computes the autocorrelations with itself and not the cross correlations. In cases where `autocorr` would return a matrix, this function returns the diagonal of the matrix. Hence it is more useful for chains with many parameters, but may not be as helpful at spotting parameters.

If `mcmc.obj` is of class `mcmc.list` then the returned vector is the average autocorrelation across all chains.

### Usage

```
autocorr.diag(mcmc.obj, ...)
```

### Arguments

`mcmc.obj`      an object of class `mcmc` or `mcmc.list`  
`...`          optional arguments to be passed to `autocorr`

### Value

A vector containing the autocorrelations.

### Author(s)

Russell Almond

### See Also

[autocorr](#), [acf](#), [autocorr.plot](#).

---

autocorr.plot      *Plot autocorrelations for Markov Chains*

---

### Description

Plots the autocorrelation function for each variable in each chain in `x`.

### Usage

```
autocorr.plot(x, lag.max, auto.layout = TRUE, ask, ...)
```

**Arguments**

<code>x</code>	A Markov Chain
<code>lag.max</code>	Maximum value at which to calculate acf
<code>auto.layout</code>	If TRUE then, set up own layout for plots, otherwise use existing one.
<code>ask</code>	If TRUE then prompt user before displaying each page of plots. Default is <code>dev.interactive()</code> in R and <code>interactive()</code> in S-PLUS.
<code>...</code>	graphical parameters

**See Also**

[autocorr.](#)

---

batchSE

*Batch Standard Error*

---

**Description**

Effective standard deviation of population to produce the correct standard errors.

**Usage**

```
batchSE(x, batchSize=100)
```

**Arguments**

<code>x</code>	An <code>mcmc</code> or <code>mcmc.list</code> object.
<code>batchSize</code>	Number of observations to include in each batch.

**Details**

Because of the autocorrelation, the usual method of taking  $\text{var}(x)/n$  overstates the precision of the estimate. This method works around the problem by looking at the means of batches of the parameter. If the batch size is large enough, the batch means should be approximately uncorrelated and the normal formula for computing the standard error should work.

The batch standard error procedure is usually thought to be not as accurate as the time series methods used in `summary` and `effectiveSize`. It is included here for completeness.

**Value**

A vector giving the standard error for each column of `x`.

**Author(s)**

Russell Almond

## References

Roberts, GO (1996) Markov chain concepts related to sampling algorithms, in Gilks, WR, Richardson, S and Spiegelhalter, DJ, *Markov Chain Monte Carlo in Practice*, Chapman and Hall, 45-58.

## See Also

[spectrum0.ar](#), [effectiveSize](#), [summary.mcmc](#)

---

bugs2jags

*Convert WinBUGS data file to JAGS data file*

---

## Description

bugs2jags converts a WinBUGS data in the format called "S-Plus" (i.e. the format created by the `dput` function) and writes it in `dump` format used by JAGS.

NB WinBUGS stores its arrays in row order. This is different from R and JAGS which both store arrays in column order. This difference is taken into account by bugs2jags which will automatically reorder the data in arrays, without changing the dimension.

Not yet available in S-PLUS.

## Usage

```
bugs2jags(infile, outfile)
```

## Arguments

<code>infile</code>	name of the input file
<code>outfile</code>	name of the output file

## Note

If the input file is saved from WinBUGS, it must be saved in plain text format. The default format for files saved from WinBUGS is a binary compound document format (with extension `odc`) that cannot be read by bugs2jags.

## Author(s)

Martyn Plummer

## References

Spiegelhalter DJ, Thomas A, Best NG and Lunn D (2003). *WinBUGS version 1.4 user manual* MRC Biostatistics Unit, Cambridge, UK.

## See Also

[dput](#), [dump](#).

## Description

`coda.options` is a utility function that queries and sets options for the `codamenu()` function. These settings affect the behaviour of the functions in the coda library only when they are called via the `codamenu()` interface.

The `coda.options()` function behaves just like the `options()` function in the base library, with the additional feature that `coda.options(default=TRUE)` will reset all options to the default values.

Options can be pretty-printed using the `display.coda.options()` function, which groups the options into sections.

Available options are

**bandwidth** Bandwidth function used when smoothing samples to produce density estimates. Defaults to Silverman's "Rule of thumb".

**combine.corr** Logical option that determines whether to combine multiple chains when calculating cross-correlations.

**combine.plots** Logical option that determines whether to combine multiple chains when plotting.

**combine.plots** Logical option that determines whether to combine multiple chains when calculating summary statistics.

**data.saved** For internal use only.

**densplot** Logical option that determines whether to plot a density plot when plot methods are called for mcmc objects.

**digits** Number of significant digits to use when printing.

**frac1** For Geweke diagnostic, fraction to use from start of chain. Defaults to 0.1

**frac2** For Geweke diagnostic, fraction to use from end of chain. Default to 0.5.

**gr.bin** For Geweke-Brooks plot, number of iterations to use per bin.

**gr.max** For Geweke-Brooks plot, maximum number of bins to use. This option overrides `gr.bin`.

**halfwidth** For Heidelberger and Welch diagnostic, the target value for the ratio of half width to sample mean.

**lowess** Logical option that controls whether to plot a smooth line through a trace plot when plotting MCMC output.

**q** For Raftery and Lewis diagnostic, the target quantile to be estimated

**r** For Raftery and Lewis diagnostic, the required precision.

**s** For Raftery and Lewis diagnostic, the probability of obtaining an estimate in the interval  $(q-r, q+r)$ .

**quantiles** Vector of quantiles to print when calculating summary statistics for MCMC output.

**trace** Logical option that determines whether to plot a trace of the sampled output when plotting MCMC output.

**user.layout** Logical option that determines whether current value of `par("mfrow")` should be used for plots (TRUE) or whether the optimal layout should be calculated (FALSE).

**Usage**

```

coda.options(...)
display.coda.options(stats = FALSE, plots = FALSE, diags = FALSE)
.Coda.Options
.Coda.Options.Default

```

**Arguments**

stats	logical flag: show summary statistic options?
plots	logical flag: show plotting options?
diags	logical flag: show plotting options?
...	list of options

**See Also**

[options](#)

---

codamenu	<i>Main menu driver for the coda package</i>
----------	--

---

**Description**

codamenu presents a simple menu-based interface to the functions in the coda package. It is designed for users who know nothing about the R/S language.

**Usage**

```
codamenu()
```

**Author(s)**

Kate Cowles, Nicky Best, Karen Vines, Martyn Plummer

---

Cramer	<i>The Cramer-von Mises Distribution</i>
--------	--

---

**Description**

Distribution function of the Cramer-von Mises distribution.

**Usage**

```
pcramer(q, eps)
```

**Arguments**

`q` vector of quantiles.  
`eps` accuracy required

**Value**

`pcramer` gives the distribution function,

**References**

Anderson TW. and Darling DA. Asymptotic theory of certain ‘goodness of fit’ criteria based on stochastic processes. *Ann. Math. Statist.*, **23**, 192-212 (1952).

Csorgo S. and Faraway, JJ. The exact and asymptotic distributions of the Cramer-von Mises statistic. *J. Roy. Stat. Soc. (B)*, **58**, 221-234 (1996).

---

`crosscorr`*Cross correlations for MCMC output*

---

**Description**

`crosscorr` calculates cross-correlations between variables in Markov Chain Monte Carlo output. If `x` is an `mcmc.list` then all chains in `x` are combined before calculating the correlation.

**Usage**

```
crosscorr(x)
```

**Arguments**

`x` an `mcmc` or `mcmc.list` object.

**Value**

A matrix or 3-d array containing the correlations.

**See Also**

[crosscorr.plot](#), [autocorr](#).

---

`crosscorr.plot`      *Plot image of correlation matrix*

---

### Description

`crosscorr.plot` provides an image of the correlation matrix for `x`. If `x` is an `mcmc.list` object, then all chains are combined.

The range `[-1,1]` is divided into a number of equal-length categories given by the length of `col` and assigned the corresponding color. By default, topographic colours are used as this makes it easier to distinguish positive and negative correlations.

### Usage

```
crosscorr.plot (x, col = topo.colors(10), ...)
```

### Arguments

<code>x</code>	an <code>mcmc</code> or <code>mcmc.list</code> object
<code>col</code>	color palette to use
<code>...</code>	graphical parameters

### See Also

[crosscorr](#), [image](#), [topo.colors](#).

---

`cumuplot`      *Cumulative quantile plot*

---

### Description

Plots the evolution of the sample quantiles as a function of the number of iterations.

### Usage

```
cumuplot(x, probs=c(0.025,0.5,0.975), ylab="",  
         lty=c(2,1), lwd=c(1,2), type="l", ask,  
         auto.layout=TRUE, col=1, ...)
```

**Arguments**

<code>x</code>	an mcmc object
<code>probs</code>	vector of desired quantiles
<code>ylab, lty, lwd, type, col</code>	graphical parameters
<code>auto.layout</code>	If TRUE, then set up own layout for plots, otherwise use existing one.
<code>ask</code>	If TRUE then prompt user before displaying each page of plots. Default is <code>dev.interactive()</code> in R and <code>interactive()</code> in S-PLUS.
<code>...</code>	further graphical parameters

**Author(s)**

Arni Magnusson <arnima@u.washington.edu>

---

`densplot`

*Probability density function estimate from MCMC output*

---

**Description**

Displays a plot of the density estimate for each variable in `x`, calculated by the `density` function.

**Usage**

```
densplot(x, show.obs=TRUE, bwf, main, ylim, ...)
```

**Arguments**

<code>x</code>	An mcmc or <code>mcmc.list</code> object
<code>show.obs</code>	Show observations along the x-axis
<code>bwf</code>	Function for calculating the bandwidth. If omitted, the bandwidth is calculate by 1.06 times the minimum of the standard deviation and the interquartile range divided by 1.34 times the sample size to the negative one fifth power
<code>main</code>	Title. See <code>par()</code>
<code>ylim</code>	Limits on y axis. See <code>par()</code>
<code>...</code>	Further graphical parameters

**Note**

You can call this function directly, but it is more usually called by the `plot.mcmc` function.

If a variable is bounded below at 0, or bounded in the interval  $[0,1]$ , then the data are reflected at the boundary before being passed to the `density()` function. This allows correct estimation of a non-zero density at the boundary.

**See Also**

[density](#), [plot.mcmc](#).

---

`effectiveSize`      *Effective sample size for estimating the mean*

---

### Description

Sample size adjusted for autocorrelation.

### Usage

```
effectiveSize(x)
```

### Arguments

`x`                      An `mcmc` or `mcmc.list` object.

### Details

For a time series `x` of length `N`, the standard error of the mean is  $\text{var}(x)/n$  where `n` is the effective sample size. `n = N` only when there is no autocorrelation.

Estimation of the effective sample size requires estimating the spectral density at frequency zero. This is done by the function `spectrum0.ar`

For a `mcmc.list` object, the effective sizes are summed across chains. To get the size for each chain individually use `lapply(x, effectiveSize)`.

### Value

A vector giving the effective sample size for each column of `x`.

### See Also

[spectrum0.ar](#).

---

`gelman.diag`                      *Gelman and Rubin's convergence diagnostic*

---

### Description

The ‘potential scale reduction factor’ is calculated for each variable in `x`, together with upper and lower confidence limits. Approximate convergence is diagnosed when the upper limit is close to 1. For multivariate chains, a multivariate value is calculated that bounds above the potential scale reduction factor for any linear combination of the (possibly transformed) variables.

The confidence limits are based on the assumption that the stationary distribution of the variable under examination is normal. Hence the ‘transform’ parameter may be used to improve the normal approximation.

**Usage**

```
gelman.diag(x, confidence = 0.95, transform=FALSE, autoburnin=TRUE)
```

**Arguments**

<code>x</code>	An <code>mcmc.list</code> object with more than one chain, and with starting values that are overdispersed with respect to the posterior distribution.
<code>confidence</code>	the coverage probability of the confidence interval for the potential scale reduction factor
<code>transform</code>	a logical flag indicating whether variables in <code>x</code> should be transformed to improve the normality of the distribution. If set to <code>TRUE</code> , a log transform or logit transform, as appropriate, will be applied.
<code>autoburnin</code>	a logical flag indicating whether only the second half of the series should be used in the computation. If set to <code>TRUE</code> (default) and <code>start(x)</code> is less than <code>end(x) / 2</code> then start of series will be adjusted so that only second half of series is used.

**Theory**

Gelman and Rubin (1992) propose a general approach to monitoring convergence of MCMC output in which  $m > 1$  parallel chains are run with starting values that are overdispersed relative to the posterior distribution. Convergence is diagnosed when the chains have ‘forgotten’ their initial values, and the output from all chains is indistinguishable. The `gelman.diag` diagnostic is applied to a single variable from the chain. It is based a comparison of within-chain and between-chain variances, and is similar to a classical analysis of variance.

There are two ways to estimate the variance of the stationary distribution: the mean of the empirical variance within each chain,  $W$ , and the empirical variance from all chains combined, which can be expressed as

$$\hat{\sigma}^2 = \frac{(n-1)W}{n} + \frac{B}{n}$$

where  $n$  is the number of iterations and  $B/n$  is the empirical between-chain variance.

If the chains have converged, then both estimates are unbiased. Otherwise the first method will *underestimate* the variance, since the individual chains have not had time to range all over the stationary distribution, and the second method will *overestimate* the variance, since the starting points were chosen to be overdispersed.

The convergence diagnostic is based on the assumption that the target distribution is normal. A Bayesian credible interval can be constructed using a t-distribution with mean

$$\hat{\mu} = \text{Sample mean of all chains combined}$$

and variance

$$\hat{V} = \hat{\sigma}^2 + \frac{B}{mn}$$

and degrees of freedom estimated by the method of moments

$$d = \frac{2 * \hat{V}^2}{\text{Var}(\hat{V})}$$

Use of the t-distribution accounts for the fact that the mean and variance of the posterior distribution are estimated.

The convergence diagnostic itself is

$$R = \sqrt{\frac{(d+3)\widehat{V}}{(d+1)W}}$$

Values substantially above 1 indicate lack of convergence. If the chains have not converged, Bayesian credible intervals based on the t-distribution are too wide, and have the potential to shrink by this factor if the MCMC run is continued.

### Note

The multivariate a version of Gelman and Rubin's diagnostic was proposed by Brooks and Gelman (1997). Unlike the univariate proportional scale reduction factor, the multivariate version does not include an adjustment for the estimated number of degrees of freedom.

### References

Gelman, A and Rubin, DB (1992) Inference from iterative simulation using multiple sequences, *Statistical Science*, **7**, 457-511.

Brooks, SP. and Gelman, A. (1997) General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics*, **7**, 434-455.

### See Also

[gelman.plot](#).

---

gelman.plot

*Gelman-Rubin-Brooks plot*

---

### Description

This plot shows the evolution of Gelman and Rubin's shrink factor as the number of iterations increases.

### Usage

```
gelman.plot(x, bin.width = 10, max.bins = 50,
confidence = 0.95, transform = FALSE, auto.layout = TRUE,
ask, col, lty, xlab, ylab, type, ...)
```

**Arguments**

<code>x</code>	an mcmc object
<code>bin.width</code>	Number of observations per segment, excluding the first segment which always has at least 50 iterations.
<code>max.bins</code>	Maximum number of bins, excluding the last one.
<code>confidence</code>	Coverage probability of confidence interval.
<code>transform</code>	Automatic variable transformation (see <code>gelman.diag</code> )
<code>auto.layout</code>	If TRUE then, set up own layout for plots, otherwise use existing one.
<code>ask</code>	Prompt user before displaying each page of plots. Default is <code>dev.interactive()</code> in R and <code>interactive()</code> in S-PLUS.
<code>col</code>	graphical parameter (see <code>par</code> )
<code>lty</code>	graphical parameter (see <code>par</code> )
<code>xlab</code>	graphical parameter (see <code>par</code> )
<code>ylab</code>	graphical parameter (see <code>par</code> )
<code>type</code>	graphical parameter (see <code>par</code> )
<code>...</code>	further graphical parameters.

**Details**

The Markov chain is divided into bins according to the arguments `bin.width` and `max.bins`. Then the Gelman-Rubin shrink factor is repeatedly calculated. The first shrink factor is calculated with observations 1:50, the second with observations 1 : (50 +  $n$ ) where  $n$  is the bin width, the third contains samples 1 : (50 + 2 $n$ ) and so on.

**Theory**

A potential problem with `gelman.diag` is that it may mis-diagnose convergence if the shrink factor happens to be close to 1 by chance. By calculating the shrink factor at several points in time, `gelman.plot` shows if the shrink factor has really converged, or whether it is still fluctuating.

**References**

Brooks, S P. and Gelman, A. (1998) General Methods for Monitoring Convergence of Iterative Simulations. *Journal of Computational and Graphical Statistics*. 7. p434-455.

**See Also**

[gelman.diag](#).

---

`geweke.diag`*Geweke's convergence diagnostic*

---

### Description

Geweke (1992) proposed a convergence diagnostic for Markov chains based on a test for equality of the means of the first and last part of a Markov chain (by default the first 10% and the last 50%). If the samples are drawn from the stationary distribution of the chain, the two means are equal and Geweke's statistic has an asymptotically standard normal distribution.

The test statistic is a standard Z-score: the difference between the two sample means divided by its estimated standard error. The standard error is estimated from the spectral density at zero and so takes into account any autocorrelation.

The Z-score is calculated under the assumption that the two parts of the chain are asymptotically independent, which requires that the sum of `frac1` and `frac2` be strictly less than 1.

### Usage

```
geweke.diag(x, frac1=0.1, frac2=0.5)
```

### Arguments

<code>x</code>	an mcmc object
<code>frac1</code>	fraction to use from beginning of chain
<code>frac2</code>	fraction to use from end of chain

### Value

Z-scores for a test of equality of means between the first and last parts of the chain. A separate statistic is calculated for each variable in each chain.

### References

Geweke, J. Evaluating the accuracy of sampling-based approaches to calculating posterior moments. In *Bayesian Statistics 4* (ed JM Bernardo, JO Berger, AP Dawid and AFM Smith). Clarendon Press, Oxford, UK.

### See Also

[geweke.plot.](#)

---

`geweke.plot`*Geweke-Brooks plot*

---

## Description

If `geweke.diag` indicates that the first and last part of a sample from a Markov chain are not drawn from the same distribution, it may be useful to discard the first few iterations to see if the rest of the chain has "converged". This plot shows what happens to Geweke's Z-score when successively larger numbers of iterations are discarded from the beginning of the chain. To preserve the asymptotic conditions required for Geweke's diagnostic, the plot never discards more than half the chain.

The first half of the Markov chain is divided into `nbins - 1` segments, then Geweke's Z-score is repeatedly calculated. The first Z-score is calculated with all iterations in the chain, the second after discarding the first segment, the third after discarding the first two segments, and so on. The last Z-score is calculated using only the samples in the second half of the chain.

## Usage

```
geweke.plot(x, frac1 = 0.1, frac2 = 0.5, nbins = 20,  
           pvalue = 0.05, auto.layout = TRUE, ask, ...)
```

## Arguments

<code>x</code>	an mcmc object
<code>frac1</code>	fraction to use from beginning of chain.
<code>frac2</code>	fraction to use from end of chain.
<code>nbins</code>	Number of segments.
<code>pvalue</code>	p-value used to plot confidence limits for the null hypothesis.
<code>auto.layout</code>	If TRUE then, set up own layout for plots, otherwise use existing one.
<code>ask</code>	If TRUE then prompt user before displaying each page of plots. Default is <code>dev.interactive()</code> in R and <code>interactive()</code> in S-PLUS.
<code>...</code>	Graphical parameters.

## Note

The graphical implementation of Geweke's diagnostic was suggested by Steve Brooks.

## See Also

[geweke.diag](#).

---

heidel.diag                      *Heidelberger and Welch's convergence diagnostic*

---

### Description

heidel.diag is a run length control diagnostic based on a criterion of relative accuracy for the estimate of the mean. The default setting corresponds to a relative accuracy of two significant digits.

heidel.diag also implements a convergence diagnostic, and removes up to half the chain in order to ensure that the means are estimated from a chain that has converged.

### Usage

```
heidel.diag(x, eps=0.1, pvalue=0.05)
```

### Arguments

x	An mcmc object
eps	Target value for ratio of halfwidth to sample mean
pvalue	significance level to use

### Details

The convergence test uses the Cramer-von-Mises statistic to test the null hypothesis that the sampled values come from a stationary distribution. The test is successively applied, firstly to the whole chain, then after discarding the first 10%, 20%, ... of the chain until either the null hypothesis is accepted, or 50% of the chain has been discarded. The latter outcome constitutes 'failure' of the stationarity test and indicates that a longer MCMC run is needed. If the stationarity test is passed, the number of iterations to keep and the number to discard are reported.

The half-width test calculates a 95% confidence interval for the mean, using the portion of the chain which passed the stationarity test. Half the width of this interval is compared with the estimate of the mean. If the ratio between the half-width and the mean is lower than `eps`, the halfwidth test is passed. Otherwise the length of the sample is deemed not long enough to estimate the mean with sufficient accuracy.

### Theory

The `heidel.diag` diagnostic is based on the work of Heidelberger and Welch (1983), who combined their earlier work on simulation run length control (Heidelberger and Welch, 1981) with the work of Schruben (1982) on detecting initial transients using Brownian bridge theory.

### Note

If the half-width test fails then the run should be extended. In order to avoid problems caused by sequential testing, the test should not be repeated too frequently. Heidelberger and Welch (1981) suggest increasing the run length by a factor  $I > 1.5$ , each time, so that estimate has the same, reasonably large, proportion of new data.

## References

- Heidelberger P and Welch PD. A spectral method for confidence interval generation and run length control in simulations. *Comm. ACM.* **24**, 233-245 (1981)
- Heidelberger P and Welch PD. Simulation run length control in the presence of an initial transient. *Opns Res.*, **31**, 1109-44 (1983)
- Schruben LW. Detecting initialization bias in simulation experiments. *Opns. Res.*, **30**, 569-590 (1982).

---

 HPDinterval

*Highest Posterior Density intervals*


---

## Description

Create Highest Posterior Density (HPD) intervals for the parameters in an MCMC sample.

## Usage

```
HPDinterval(obj, prob = 0.95, ...)
## S3 method for class 'mcmc':
HPDinterval(obj, prob = 0.95, ...)
## S3 method for class 'mcmc.list':
HPDinterval(obj, prob = 0.95, ...)
```

## Arguments

obj	The object containing the MCMC sample - usually of class "mcmc" or "mcmc.list"
prob	A numeric scalar in the interval (0,1) giving the target probability content of the intervals. The nominal probability content of the intervals is the multiple of $1/nrow(obj)$ nearest to prob.
...	Optional additional arguments for methods. None are used at present.

## Details

For each parameter the interval is constructed from the empirical cdf of the sample as the shortest interval for which the difference in the ecdf values of the endpoints is the nominal probability. Assuming that the distribution is not severely multimodal, this is the HPD interval.

## Value

For an "mcmc" object, a matrix with columns "lower" and "upper" and rows corresponding to the parameters. The attribute "Probability" is the nominal probability content of the intervals. A list of such matrices is returned for an "mcmc.list" object.

## Author(s)

Douglas Bates

**Examples**

```
data(line)
HPDinterval(line)
```

---

line	<i>Simple linear regression example</i>
------	---

---

**Description**

Sample MCMC output from a simple linear regression model given in the BUGS manual.

**Usage**

```
data(line)
```

**Format**

An `mcmc` object

**Source**

Spiegelhalter, D.J., Thomas, A., Best, N.G. and Gilks, W.R. (1995) BUGS: Bayesian inference using Gibbs Sampling, Version 0.5, MRC Biostatistics Unit, Cambridge.

---

mcmc	<i>Markov Chain Monte Carlo Objects</i>
------	---

---

**Description**

The function ‘`mcmc`’ is used to create a Markov Chain Monte Carlo object. The data are taken to be a vector, or a matrix with one column per variable.

An `mcmc` object may be summarized by the `summary` function and visualized with the `plot` function.

MCMC objects resemble time series (`ts`) objects and have methods for the generic functions `time`, `start`, `end`, `frequency` and `window`.

**Usage**

```
mcmc(data= NA, start = 1, end = numeric(0), thin = 1)
as.mcmc(x)
is.mcmc(x)
```

**Arguments**

<code>data</code>	a vector or matrix of MCMC output
<code>start</code>	the iteration number of the first observation
<code>end</code>	the iteration number of the last observation
<code>thin</code>	the thinning interval between consecutive observations
<code>x</code>	An object that may be coerced to an mcmc object

**Note**

The format of the mcmc class has changed between coda version 0.3 and 0.4. Older mcmc objects will now cause `is.mcmc` to fail with an appropriate warning message. Obsolete mcmc objects can be upgraded with the `mcmcUpgrade` function.

**Author(s)**

Martyn Plummer

**See Also**

[mcmc.list](#), [mcmcUpgrade](#), [thin](#), [window.mcmc](#), [summary.mcmc](#), [plot.mcmc](#).

---

mcmc.convert

*Conversions of MCMC objects*

---

**Description**

These are methods for the generic functions `as.matrix()`, `as.array()` and `as.mcmc()`.

`as.matrix()` strips the MCMC attributes from an mcmc object and returns a matrix. If `iters = TRUE` then a column is added with the iteration number. For `mcmc.list` objects, the rows of multiple chains are concatenated and, if `chains = TRUE` a column is added with the chain number.

`mcmc.list` objects can be coerced to 3-dimensional arrays with the `as.array()` function.

An `mcmc.list` object with a single chain can be coerced to an mcmc object with `as.mcmc()`. If the argument has multiple chains, this causes an error.

**Usage**

```
## S3 method for class 'mcmc':
as.matrix(x, iters = FALSE, ...)
## S3 method for class 'mcmc.list':
as.matrix(x, iters = FALSE, chains = FALSE, ...)
## S3 method for class 'mcmc.list':
as.array(x, drop, ...)
```

**Arguments**

<code>x</code>	An <code>mcmc</code> or <code>mcmc.list</code> object
<code>iters</code>	logical flag: add column for iteration number?
<code>chains</code>	logical flag: add column for chain number? (if <code>mcmc.list</code> )
<code>drop</code>	logical flag: if <code>TRUE</code> the result is coerced to the lowest possible dimension
<code>...</code>	optional arguments to the various methods

**See Also**

`as.matrix`, `as.array`, `as.mcmc`,

---

`mcmc.list`

*Replicated Markov Chain Monte Carlo Objects*

---

**Description**

The function ‘`mcmc.list`’ is used to represent parallel runs of the same chain, with different starting values and random seeds. The list must be balanced: each chain in the list must have the same iterations and the same variables.

Diagnostic functions which act on `mcmc` objects may also be applied to `mcmc.list` objects. In general, the chains will be combined, if this makes sense, otherwise the diagnostic function will be applied separately to each chain in the list.

Since all the chains in the list have the same iterations, a single time dimension can be ascribed to the list. Hence there are time series methods `time`, `window`, `start`, `end`, `frequency` and `thin` for `mcmc.list` objects.

An `mcmc.list` can be indexed as if it were a single `mcmc` object using the `[]` operator (see examples below). The `[[` operator selects a single `mcmc` object from the list.

**Usage**

```
mcmc.list(...)  
as.mcmc.list(x, ...)  
is.mcmc.list(x)
```

**Arguments**

<code>...</code>	a list of <code>mcmc</code> objects
<code>x</code>	an object that may be coerced to <code>mcmc.list</code>

**Author(s)**

Martyn Plummer

**See Also**[mcmc.](#)**Examples**

```

data(line)
x1 <- line[[1]]           #Select first chain
x2 <- line[,1, drop=FALSE] #Select first var from all chains
varnames(x2) == varnames(line)[1] #TRUE

```

mcmc.subset

*Extract or replace parts of MCMC objects***Description**

These are methods for subsetting `mcmc` objects. You can select iterations using the first dimension and variables using the second dimension. Selecting iterations will return a vector or matrix, not an `mcmc` object. If you want to do row-subsetting of an `mcmc` object and preserve its dimensions, use the `window` function.

Subsetting applied to an `mcmc.list` object will simultaneously affect all the parallel chains in the object.

**Usage**

```

## S3 method for class 'mcmc':
x[i, j, drop=missing(i)]
## S3 method for class 'mcmc.list':
x[i, j, drop=TRUE]

```

**Arguments**

<code>x</code>	An <code>mcmc</code> object
<code>i</code>	Row to extract
<code>j</code>	Column to extract
<code>drop</code>	if <code>TRUE</code> , the redundant dimensions are dropped

**See Also**[\[, window.mcmc](#)

---

mcmcUpgrade	<i>Upgrade mcmc objects in obsolete format</i>
-------------	--

---

### Description

In previous releases of CODA, an `mcmc` object could be a single or multiple chains. A new class `mcmc.list` has now been introduced to deal with multiple chains and `mcmc` objects can only have data from a single chain.

Objects stored in the old format are now obsolete and must be upgraded.

### Usage

```
mcmcUpgrade(x)
```

### Arguments

`x` an obsolete `mcmc` object.

### Author(s)

Martyn Plummer

### See Also

[mcmc](#).

---

mcpars	<i>Mcpars attribute of MCMC objects</i>
--------	---

---

### Description

The ‘`mcpars`’ attribute of an MCMC object gives the start iteration the end iteration and the thinning interval of the chain.

It resembles the ‘`tsp`’ attribute of time series (`ts`) objects.

### Usage

```
mcpars(x)
```

### Arguments

`x` An `mcmc` or `mcmc.list` object

### See Also

[ts](#), [mcmc](#), [mcmc.list](#),

---

`multi.menu`*Choose multiple options from a menu*

---

### Description

`multi.menu` presents the user with a menu of choices labelled from 1 to the number of choices. The user may choose one or more options by entering a comma separated list. A range of values may also be specified using the ":" operator. Mixed expressions such as "1,3:5, 6" are permitted.

If `allow.zero` is set to TRUE, one can select '0' to exit without choosing an item.

### Usage

```
multi.menu(choices, title, header, allow.zero = TRUE)
```

### Arguments

<code>choices</code>	Character vector of labels for choices
<code>title</code>	Title printed before menu
<code>header</code>	Character vector of length 2 giving column titles
<code>allow.zero</code>	Permit 0 as an acceptable response

### Value

Numeric vector giving the numbers of the options selected, or 0 if no selection is made.

### Author(s)

Martyn Plummer

### See Also

[menu.](#)

---

`nchain`*Dimensions of MCMC objects*

---

### Description

These functions give the dimensions of an MCMC object

**niter(x)** returns the number of iterations.

**nvar(x)** returns the number of variables.

**nchain(x)** returns the number of parallel chains.

**Usage**

```
niter(x)
nvar(x)
nchain(x)
```

**Arguments**

`x` An `mcmc` or `mcmc.list` object

**Value**

A numeric vector of length 1:

**See Also**

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

---

`plot.mcmc`

*Summary plots of mcmc objects*

---

**Description**

`plot.mcmc` summarizes an `mcmc` or `mcmc.list` object with a trace of the sampled output and a density estimate for each variable in the chain.

**Usage**

```
## S3 method for class 'mcmc':
plot(x, trace = TRUE, density = TRUE, smooth = TRUE, bwf,
      auto.layout = TRUE, ask = dev.interactive(), ...)
```

**Arguments**

<code>x</code>	an object of class <code>mcmc</code> or <code>mcmc.list</code>
<code>trace</code>	Plot trace of each variable
<code>density</code>	Plot density estimate of each variable
<code>smooth</code>	Draw a smooth line through trace plots
<code>bwf</code>	Bandwidth function for density plots
<code>auto.layout</code>	Automatically generate output format
<code>ask</code>	Prompt user before each page of plots
<code>...</code>	Further arguments

**Author(s)**

Martyn Plummer

**See Also**

[densplot](#), [traceplot](#).

---

raftery.diag	<i>Raftery and Lewis's diagnostic</i>
--------------	---------------------------------------

---

**Description**

`raftery.diag` is a run length control diagnostic based on a criterion of accuracy of estimation of the quantile  $q$ . It is intended for use on a short pilot run of a Markov chain. The number of iterations required to estimate the quantile  $q$  to within an accuracy of  $\pm r$  with probability  $p$  is calculated. Separate calculations are performed for each variable within each chain.

If the number of iterations in `data` is too small, an error message is printed indicating the minimum length of pilot run. The minimum length is the required sample size for a chain with no correlation between consecutive samples. Positive autocorrelation will increase the required sample size above this minimum value. An estimate  $I$  (the 'dependence factor') of the extent to which autocorrelation inflates the required sample size is also provided. Values of  $I$  larger than 5 indicate strong autocorrelation which may be due to a poor choice of starting value, high posterior correlations or 'stickiness' of the MCMC algorithm.

The number of 'burn in' iterations to be discarded at the beginning of the chain is also calculated.

**Usage**

```
raftery.diag(data, q=0.025, r=0.005, s=0.95, converge.eps=0.001)
```

**Arguments**

<code>data</code>	an mcmc object
<code>q</code>	the quantile to be estimated.
<code>r</code>	the desired margin of error of the estimate.
<code>s</code>	the probability of obtaining an estimate in the interval $(q-r, q+r)$ .
<code>converge.eps</code>	Precision required for estimate of time to convergence.

**Value**

A list with class `raftery.diag`. A print method is available for objects of this class. the contents of the list are

<code>tspar</code>	The time series parameters of <code>data</code>
<code>params</code>	A vector containing the parameters <code>r</code> , <code>s</code> and <code>q</code>
<code>Niters</code>	The number of iterations in <code>data</code>
<code>resmatrix</code>	A 3-d array containing the results: $M$ the length of "burn in", $N$ the required sample size, $Nmin$ the minimum sample size based on zero autocorrelation and $I = (M + N)/Nmin$ the "dependence factor"

## Theory

The estimated sample size for variable  $U$  is based on the process  $Z_t = d(U_t \leq u)$  where  $d$  is the indicator function and  $u$  is the  $q$ th quantile of  $U$ . The process  $Z_t$  is derived from the Markov chain data by marginalization and truncation, but is not itself a Markov chain. However,  $Z_t$  may behave as a Markov chain if it is sufficiently thinned out. `raftery.diag` calculates the smallest value of thinning interval  $k$  which makes the thinned chain  $Z_t^k$  behave as a Markov chain. The required sample size is calculated from this thinned sequence. Since some data is ‘thrown away’ the sample size estimates are conservative.

The criterion for the number of ‘burn in’ iterations  $m$  to be discarded, is that the conditional distribution of  $Z_m^k$  given  $Z_0$  should be within `converge.eps` of the equilibrium distribution of the chain  $Z_t^k$ .

## Note

`raftery.diag` is based on the FORTRAN program ‘gibbsit’ written by Steven Lewis, and available from the Statlib archive.

## References

Raftery, A.E. and Lewis, S.M. (1992). One long run with diagnostics: Implementation strategies for Markov chain Monte Carlo. *Statistical Science*, **7**, 493-497.

Raftery, A.E. and Lewis, S.M. (1995). The number of iterations, convergence diagnostics and generic Metropolis algorithms. *In Practical Markov Chain Monte Carlo* (W.R. Gilks, D.J. Spiegelhalter and S. Richardson, eds.). London, U.K.: Chapman and Hall.

---

`read.and.check`      *Read data interactively and check that it satisfies conditions*

---

## Description

Input is read interactively and checked against conditions specified by the arguments `what`, `lower`, `upper` and `answer.in`. If the input does not satisfy all the conditions, an appropriate error message is produced and the user is prompted to provide input. This process is repeated until a valid input value is entered.

## Usage

```
read.and.check(message = "", what = numeric(), lower, upper, answer.in,
default)
```

## Arguments

<code>message</code>	message displayed before prompting for user input.
<code>what</code>	the type of <code>what</code> gives the type of data to be read.
<code>lower</code>	lower limit of input, for numeric input only.

upper	upper limit of input, for numeric input only.
answer.in	the input must correspond to one of the elements of the vector <code>answer.in</code> , if supplied.
default	value assumed if user enters a blank line.

**Value**

The value of the valid input. When the `default` argument is specified, a blank line is accepted as valid input and in this case `read.and.check` returns the value of `default`.

**Note**

Since the function does not return a value until it receives valid input, it extensively checks the conditions for consistency before prompting the user for input. Inconsistent conditions will cause an error.

**Author(s)**

Martyn Plummer

---

read.coda                      *Read output files in CODA format*

---

**Description**

`read.coda` reads Markov Chain Monte Carlo output in the CODA format produced by OpenBUGS and JAGS. By default, all of the data in the file is read, but the arguments `start`, `end` and `thin` may be used to read a subset of the data. If the arguments given to `start`, `end` or `thin` are incompatible with the data, they are ignored.

**Usage**

```
read.coda(output.file, index.file, start, end, thin, quiet=FALSE)
read.jags(file = "jags.out", start, end, thin, quiet=FALSE)
```

**Arguments**

<code>output.file</code>	The name of the file containing the monitored output
<code>index.file</code>	The name of the file containing the index, showing which rows of the output file correspond to which variables
<code>file</code>	For JAGS output, the name of the output file. The extension ".out" may be omitted. There must be a corresponding ".ind" file with the same file stem.
<code>start</code>	First iteration of chain
<code>end</code>	Last iteration of chain
<code>thin</code>	Thinning interval for chain
<code>quiet</code>	Logical flag. If true, a progress summary will be printed

**Value**

An object of class `mcmc` containing a representation of the data in the file.

**Author(s)**

Karen Vines, Martyn Plummer

**References**

Spiegelhalter DJ, Thomas A, Best NG and Gilks WR (1995). *BUGS: Bayesian inference Using Gibbs Sampling, Version 0.50*. MRC Biostatistics Unit, Cambridge.

**See Also**

`mcmc`, `read.coda.interactive`, `read.openbugs`.

---

```
read.coda.interactive
```

*Read CODA output files interactively*

---

**Description**

`read.coda.interactive` reads Markov Chain Monte Carlo output in the format produced by the classic BUGS program. No arguments are required. Instead, the user is prompted for the required information.

**Usage**

```
read.coda.interactive()
```

**Value**

An object of class `mcmc.list` containing a representation of the data in one or more BUGS output files.

**Note**

This function is normally called by the `codamenu` function, but can also be used on a stand-alone basis.

**Author(s)**

Nicky Best, Martyn Plummer

**References**

Spiegelhalter DJ, Thomas A, Best NG and Gilks WR (1995). *BUGS: Bayesian inference Using Gibbs Sampling, Version 0.50*. MRC Biostatistics Unit, Cambridge.

**See Also**

[mcmc](#), [mcmc.list](#), [read.coda](#), [codamenu](#).

---

read.openbugs      *Read CODA output files produced by OpenBUGS*

---

**Description**

`read.openbugs` reads Markov Chain Monte Carlo output in the CODA format produced by OpenBUGS.

This is a convenience wrapper around the function `read.coda` which allows you to read all the data output by OpenBUGS by specifying only the file stem.

**Usage**

```
read.openbugs(stem="", start, end, thin, quiet=FALSE)
```

**Arguments**

<code>stem</code>	Character string giving the stem for the output files. OpenBUGS produces files with names "<stem>CODAindex.txt", "<stem>CODAchain1.txt", "<stem>CODAchain2.txt", ...
<code>start</code>	First iteration of chain
<code>end</code>	Last iteration of chain
<code>thin</code>	Thinning interval for chain
<code>quiet</code>	Logical flag. If true, a progress summary will be printed

**Value**

An object of class `mcmc.list` containing output from all chains.

**Author(s)**

Martyn Plummer

**References**

Spiegelhalter DJ, Thomas A, Best NG and Lunn D (2004). *WinBUGS User Manual, Version 2.0, June 2004*, MRC Biostatistics Unit, Cambridge.

**See Also**

[read.coda](#).

---

`rejectionRate`      *Rejection Rate for Metropolis–Hastings chains*

---

### Description

`rejectionRate` calculates the fraction of time that a Metropolis–Hastings type chain rejected a proposed move. The rejection rate is calculated separately for each variable in the `mcmc.obj` argument, irregardless of whether the variables were drawn separately or in a block. In the latter case, the values returned should be the same.

### Usage

```
rejectionRate(x)
```

### Arguments

`x`                      An `mcmc` or `mcmc.list` object.

### Details

For the purposes of this function, a "rejection" has occurred if the value of the time series is the same at two successive time points. This test is done naively using `==` and may produce problems due to rounding error.

### Value

A vector containing the rejection rates, one for each variable.

### Author(s)

Russell Almond

---

`spectrum0`              *Estimate spectral density at zero*

---

### Description

The spectral density at frequency zero is estimated by fitting a glm to the low-frequency end of the periodogram. `spectrum0(x)/length(x)` estimates the variance of `mean(x)`.

### Usage

```
spectrum0(x, max.freq = 0.5, order = 1, max.length = 200)
```

**Arguments**

<code>x</code>	A time series.
<code>max.freq</code>	The glm is fitted on the frequency range (0, max.freq]
<code>order</code>	Order of the polynomial to fit to the periodogram.
<code>max.length</code>	The data <code>x</code> is aggregated if necessary by taking batch means so that the length of the series is less than <code>max.length</code> . If this is set to <code>NULL</code> no aggregation occurs.

**Details**

The raw periodogram is calculated for the series `x` and a generalized linear model with family Gamma and log link is fitted to the periodogram.

The linear predictor is a polynomial in terms of the frequency. The degree of the polynomial is determined by the parameter `order`.

**Value**

A list with the following values

<code>spec</code>	The predicted value of the spectral density at frequency zero.
-------------------	--

**Theory**

Heidelberger and Welch (1991) observed that the usual non-parametric estimator of the spectral density, obtained by smoothing the periodogram, is not appropriate for frequency zero. They proposed an alternative parametric method which consisted of fitting a linear model to the log periodogram of the batched time series. Some technical problems with model fitting in their original proposal can be overcome by using a generalized linear model.

Batching of the data, originally proposed in order to save space, has the side effect of flattening the spectral density and making a polynomial fit more reasonable. Fitting a polynomial of degree zero is equivalent to using the ‘batched means’ method.

**Note**

The definition of the spectral density used here differs from that used by `spec.pgram`. We consider the frequency range to be between 0 and 0.5, not between 0 and `frequency(x)/2`.

The model fitting may fail on chains with very high autocorrelation.

**References**

Heidelberger, P and Welch, P.D. A spectral method for confidence interval generation and run length control in simulations. Communications of the ACM, Vol 24, pp233-245, 1981.

**See Also**

[spectrum](#), [spectrum0.ar](#), [glm](#).

---

spectrum0.ar	<i>Estimate spectral density at zero</i>
--------------	--

---

### Description

The spectral density at frequency zero is estimated by fitting an autoregressive model. `spectrum0(x)/length(x)` estimates the variance of `mean(x)`.

### Usage

```
spectrum0.ar(x)
```

### Arguments

x	A time series.
---	----------------

### Details

The `ar()` function to fit an autoregressive model to the time series `x`. For multivariate time series, separate models are fitted for each column. The value of the spectral density at zero is then given by a well-known formula.

### Value

A list with the following values

spec	The predicted value of the spectral density at frequency zero.
order	The order of the fitted model

### Note

The definition of the spectral density used here differs from that used by `spec.pgram`. We consider the frequency range to be between 0 and 0.5, not between 0 and `frequency(x)/2`.

### See Also

[spectrum](#), [spectrum0](#), [glm](#).

---

summary.mcmc

*Summary statistics for Markov Chain Monte Carlo chains*


---

**Description**

summary.mcmc produces two sets of summary statistics for each variable:

Mean, standard deviation, naive standard error of the mean (ignoring autocorrelation of the chain) and time-series standard error based on an estimate of the spectral density at 0.

Quantiles of the sample distribution using the `quantiles` argument.

**Usage**

```
## S3 method for class 'mcmc':
summary(object, quantiles = c(0.025, 0.25, 0.5, 0.75, 0.975), ...)
```

**Arguments**

<code>object</code>	an object of class <code>mcmc</code> or <code>mcmc.list</code>
<code>quantiles</code>	a vector of quantiles to evaluate for each variable
<code>...</code>	a list of further arguments

**Author(s)**

Martyn Plummer

**See Also**

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

---

thin

*Thinning interval*


---

**Description**

thin returns the interval between successive values of a time series. `thin(x)` is equivalent to `1/frequency(x)`.

This is a generic function. Methods have been implemented for `mcmc` objects.

**Usage**

```
thin(x, ...)
```

**Arguments**

x                    a regular time series  
 ...                  a list of arguments

**Author(s)**

Martyn Plummer

**See Also**

[time.](#)

---

time.mcmc

*Time attributes for mcmc objects*

---

**Description**

These are methods for mcmc objects for the generic time series functions.

**Usage**

```
## S3 method for class 'mcmc':
time(x, ...)
## S3 method for class 'mcmc':
start(x, ...)
## S3 method for class 'mcmc':
end(x, ...)
## S3 method for class 'mcmc':
thin(x, ...)
```

**Arguments**

x                    an mcmc or mcmc.list object  
 ...                  extra arguments for future methods

**See Also**

[time](#), [start](#), [frequency](#), [thin](#).

---

`traceplot`*Trace plot of MCMC output*

---

**Description**

Displays a plot of iterations *vs.* sampled values for each variable in the chain, with a separate plot per variable.

**Usage**

```
traceplot(x, smooth = TRUE, col, type, ylab, ...)
```

**Arguments**

<code>x</code>	An <code>mcmc</code> or <code>mcmc.list</code> object
<code>smooth</code>	draw smooth line through trace plot
<code>col</code>	graphical parameter (see <code>par</code> )
<code>type</code>	graphical parameter (see <code>par</code> )
<code>ylab</code>	graphical parameter (see <code>par</code> )
<code>...</code>	further graphical parameters

**Note**

You can call this function directly, but it is more usually called by the `plot.mcmc` function.

**See Also**

[densplot](#), [plot.mcmc](#).

---

`trellisplots`*Trellis plots for mcmc objects*

---

**Description**

These methods use the Trellis framework as implemented in the `lattice` package to produce space-conserving diagnostic plots from "mcmc" and "mcmc.list" objects. The `xyplot` methods produce trace plots. The `densityplot` methods and `qqmath` methods produce empirical density and probability plots. The `levelplot` method depicts the correlation of the series. The `acfplot` methods plot the auto-correlation in the series.

Not yet available in S-PLUS.

**Usage**

```

## S3 method for class 'mcmc':
densityplot(x, data,
            outer, aspect = "xy",
            default.scales = list(relation = "free"),
            start = 1, thin = 1,
            main = attr(x, "title"),
            xlab = "",
            plot.points = "rug",
            ...,
            subset)
## S3 method for class 'mcmc.list':
densityplot(x, data,
            outer = FALSE, groups = !outer,
            aspect = "xy",
            default.scales = list(relation = "free"),
            start = 1, thin = 1,
            main = attr(x, "title"),
            xlab = "",
            plot.points = "rug",
            ...,
            subset)
## S3 method for class 'mcmc':
levelplot(x, data, main = attr(x, "title"),
          start = 1, thin = 1,
          ...,
          xlab = "", ylab = "",
          cuts = 10, at,
          col.regions = topo.colors(100),
          subset)
## S3 method for class 'mcmc':
qqmath(x, data,
        outer, aspect = "xy",
        default.scales = list(y = list(relation = "free")),
        prepanel = prepanel.qqmathline,
        start = 1, thin = 1,
        main = attr(x, "title"),
        ylab = "",
        ...,
        subset)
## S3 method for class 'mcmc.list':
qqmath(x, data,
        outer = FALSE, groups = !outer,
        aspect = "xy",
        default.scales = list(y = list(relation = "free")),
        prepanel = prepanel.qqmathline,
        start = 1, thin = 1,
        main = attr(x, "title"),

```

```

        ylab = "",
        ...,
        subset)
## S3 method for class 'mcmc':
xyplot(x, data,
        outer, layout = c(1, ncol(x)),
        default.scales = list(y = list(relation = "free")),
        type = 'l',
        start = 1, thin = 1,
        ylab = "",
        xlab = "Iteration number",
        main = attr(x, "title"),
        ...,
        subset)
## S3 method for class 'mcmc.list':
xyplot(x, data, outer = FALSE, groups = !outer,
        aspect = "xy", layout = c(1, ncol(x[[1]])),
        default.scales = list(y = list(relation = "free")),
        type = 'l',
        start = 1, thin = 1,
        main = attr(x, "title"),
        ylab = "",
        ...,
        subset)
acfplot(x, data, ...)
## S3 method for class 'mcmc':
acfplot(x, data, outer,
        prepanel, panel,
        type = 'h',
        aspect = "xy",
        start = 1, thin = 1,
        lag.max = NULL,
        ylab = "Autocorrelation",
        xlab = "Lag",
        main = attr(x, "title"),
        ...,
        subset)
## S3 method for class 'mcmc.list':
acfplot(x, data, outer = FALSE, groups = !outer,
        prepanel, panel,
        type = if (groups) 'b' else 'h',
        aspect = "xy",
        start = 1, thin = 1,
        lag.max = NULL,
        ylab = "Autocorrelation",
        xlab = "Lag",
        main = attr(x, "title"),
        ...,

```

subset)

## Arguments

<code>x</code>	an "mcmc" or "mcmc.list" object.
<code>data</code>	ignored, present for consistency with generic.
<code>outer</code>	for the "mcmc.list" methods, a logical flag to control whether multiple runs of a series are displayed in the same panel (they are if FALSE, not if TRUE). If specified in the "mcmc" methods, this argument is ignored with a warning.
<code>groups</code>	for the "mcmc.list" methods, a logical flag to control whether the underlying lattice call will be supplied a <code>groups</code> arguments indicating which run a data point originated from. The panel function is responsible for handling such an argument, and will usually differentiate runs within a panel by using different graphical parameters. When <code>outer=FALSE</code> , the default of <code>groups</code> is TRUE if the corresponding default panel function is able to make use of such information. When <code>outer=FALSE</code> , <code>groups=TRUE</code> will be ignored with a warning.
<code>aspect</code>	controls the physical aspect ratio of the panel. See <code>xyplot</code> for details. The default for these methods is chosen carefully - check what the default plot looks like before changing this parameter.
<code>default.scales</code>	this parameter provides a reasonable default value of the <code>scales</code> parameter for the method. It is unlikely that a user will wish to change this parameter. Pass a value for <code>scales</code> (see <code>xyplot</code> ) instead, which will override values specified here.
<code>type</code>	a character vector that determines if lines, points, etc. are drawn on the panel. The default values for the methods are carefully chosen. See <code>panel.xyplot</code> for possible values.
<code>thin</code>	an optional thinning interval that is applied before the plot is drawn.
<code>start</code>	an optional value for the starting point within the series. Values before the starting point are considered part of the "burn-in" of the series and dropped.
<code>plot.points</code>	character argument giving the style in which points are added to the plot. See <code>panel.densityplot</code> for details.
<code>layout</code>	a method-specific default for the <code>layout</code> argument to the lattice functions.
<code>xlab, ylab, main</code>	Used to provide default axis annotations and plot labels.
<code>cuts, at</code>	defines number and location of values where colors change
<code>col.regions</code>	color palette used
<code>lag.max</code>	maximum lag for which autocorrelation is computed. By default, the value chosen by <code>acf</code> is used
<code>prepanel, panel</code>	suitable prepanel and panel functions for <code>acfplot</code> . The prepanel function omits the lag-0 auto-correlation (which is always 1) from the range calculations.
<code>...</code>	other arguments, passed to the lattice function. Documentation of the corresponding generics in the <code>lattice</code> package should be consulted for possible arguments.

subset indices of the subset of the series to plot. The default is constructed from the `start` and `thin` arguments.

### Value

An object of class "trellis". The relevant `update` method can be used to update components of the object and the `print` method (usually called by default) will plot it on an appropriate plotting device.

### Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

### See Also

[Lattice](#) for a brief introduction to lattice displays and links to further documentation.

### Examples

```
data(line)

## Not run:
xyplot(line)
xyplot(line[[1]], start = 10)
densityplot(line, start = 10)
qqmath(line, start = 10)
levelplot(line[[2]])
acfplot(line, outer = TRUE)
## End(Not run)
```

---

varnames

*Named dimensions of MCMC objects*

---

### Description

`varnames()` returns the variable names and `chanames` returns the chain names, or NULL if these are not set.

If `allow.null = FALSE` then NULL values will be replaced with canonical names.

### Usage

```
varnames(x, allow.null=TRUE)
chanames(x, allow.null=TRUE)
varnames(x) <- value
chanames(x) <- value
```

**Arguments**

<code>x</code>	an <code>mcmc</code> or <code>mcmc.list</code> object
<code>allow.null</code>	Logical argument that determines whether the function may return <code>NULL</code>
<code>value</code>	A character vector, or <code>NULL</code>

**Value**

A character vector, or `NULL`.

**See Also**

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

---

`window.mcmc`

*Time windows for mcmc objects*

---

**Description**

`window.mcmc` is a method for `mcmc` objects which is normally called by the generic function `window`

In addition to the generic parameters, `start` and `end` the additional parameter `thin` may be used to thin out the Markov chain. Setting `thin=k` selects every `k`th iteration starting with the first. Note that the value of `thin` is *absolute* not relative. The value supplied given to the parameter `thin` must be a multiple of `thin(x)`.

Values of `start`, `end` and `thin` which are inconsistent with `x` are ignored, but a warning message is issued.

**Usage**

```
## S3 method for class 'mcmc':
window(x, start, end, thin, ...)
```

**Arguments**

<code>x</code>	an <code>mcmc</code> object
<code>start</code>	the first iteration of interest
<code>end</code>	the last iteration of interest
<code>thin</code>	the required interval between successive samples
<code>...</code>	further arguments for future methods

**See Also**

[window](#), [thin](#).

# Index

- \*Topic **array**
  - crosscorr, 8
  - mcmc.convert, 20
- \*Topic **datasets**
  - line, 19
- \*Topic **distribution**
  - Cramer, 7
- \*Topic **file**
  - bugs2jags, 5
  - read.coda, 28
  - read.coda.interactive, 29
  - read.openbugs, 30
- \*Topic **hplot**
  - autocorr.plot, 3
  - crosscorr.plot, 9
  - cumuplot, 9
  - densplot, 10
  - gelman.plot, 13
  - geweke.plot, 16
  - plot.mcmc, 25
  - traceplot, 36
  - trellisplots, 36
- \*Topic **htest**
  - gelman.diag, 11
  - geweke.diag, 15
  - heidel.diag, 17
  - HPDinterval, 18
  - raftery.diag, 26
- \*Topic **manip**
  - varnames, 40
- \*Topic **multivariate**
  - crosscorr, 8
- \*Topic **ts**
  - as.ts.mcmc, 1
  - autocorr, 2
  - autocorr.diag, 3
  - batchSE, 4
  - effectiveSize, 11
  - mcmc, 19
  - mcmc.list, 21
  - mcmc.subset, 22
  - mcmcUpgrade, 23
  - mcpair, 23
  - nchain, 24
  - rejectionRate, 31
  - spectrum0, 31
  - spectrum0.ar, 33
  - thin, 34
  - time.mcmc, 35
  - window.mcmc, 41
- \*Topic **univar**
  - HPDinterval, 18
  - summary.mcmc, 34
- \*Topic **utilities**
  - as.ts.mcmc, 1
  - coda.options, 6
  - codamenu, 7
  - multi.menu, 24
  - read.and.check, 27
  - .Coda.Options(*coda.options*), 6
  - [, 22
  - [.mcmc(*mcmc.subset*), 22
  - acf, 2, 3, 39
  - acfplot(*trellisplots*), 36
  - acfplot.mcmc.list(*trellisplots*), 36
  - as.array, 21
  - as.array.mcmc.list(*mcmc.convert*), 20
  - as.matrix, 21
  - as.matrix.mcmc(*mcmc.convert*), 20
  - as.mcmc, 21
  - as.mcmc(*mcmc*), 19
  - as.mcmc.list(*mcmc.list*), 21
  - as.mcmc.mcmc.list(*mcmc.convert*), 20
  - as.ts, 2
  - as.ts.mcmc, 1

- autocorr, [2, 3, 4, 8](#)
- autocorr.diag, [3](#)
- autocorr.diag.mcmc.list  
(*autocorr.diag*), [3](#)
- autocorr.plot, [2, 3, 3](#)
- batchSE, [4](#)
- bugs2jags, [5](#)
- chanames (*varnames*), [40](#)
- chanames<- (*varnames*), [40](#)
- coda.options, [6](#)
- codamenu, [7, 30](#)
- Cramer, [7](#)
- crosscorr, [8, 9](#)
- crosscorr.plot, [8, 9](#)
- cumuplot, [9](#)
- density, [10](#)
- densityplot.mcmc (*trellisplots*),  
[36](#)
- densplot, [10, 26, 36](#)
- display.coda.options  
(*coda.options*), [6](#)
- dput, [5](#)
- dump, [5](#)
- effectiveSize, [5, 11](#)
- end.mcmc (*time.mcmc*), [35](#)
- frequency, [35](#)
- frequency.mcmc (*time.mcmc*), [35](#)
- gelman.diag, [11, 14](#)
- gelman.plot, [13, 13](#)
- geweke.diag, [15, 16](#)
- geweke.plot, [15, 16](#)
- glm, [32, 33](#)
- heidel.diag, [17](#)
- HPDinterval, [18](#)
- HPDinterval.mcmc.list  
(*HPDinterval*), [18](#)
- image, [9](#)
- is.mcmc (*mcmc*), [19](#)
- is.mcmc.list (*mcmc.list*), [21](#)
- Lattice, [40](#)
- levelplot.mcmc (*trellisplots*), [36](#)
- line, [19](#)
- mcmc, [19, 22, 23, 25, 29, 30, 34, 41](#)
- mcmc.convert, [20](#)
- mcmc.list, [20, 21, 23, 25, 30, 34, 41](#)
- mcmc.subset, [22](#)
- mcmcUpgrade, [20, 23](#)
- mcpair, [23](#)
- menu, [24](#)
- multi.menu, [24](#)
- nchain, [24](#)
- niter (*nchain*), [24](#)
- nvar (*nchain*), [24](#)
- options, [7](#)
- panel.densityplot, [39](#)
- panel.xyplot, [39](#)
- pcramer (*Cramer*), [7](#)
- plot.mcmc, [10, 20, 25, 36](#)
- plot.mcmc.list (*mcmc.list*), [21](#)
- print, [40](#)
- print.mcmc (*mcmc*), [19](#)
- qqmath.mcmc (*trellisplots*), [36](#)
- raftery.diag, [26](#)
- read.and.check, [27](#)
- read.coda, [28, 30](#)
- read.coda.interactive, [29, 29](#)
- read.jags (*read.coda*), [28](#)
- read.openbugs, [29, 30](#)
- rejectionRate, [31](#)
- rejectionRate.mcmc.list  
(*rejectionRate*), [31](#)
- spectrum, [32, 33](#)
- spectrum0, [31, 33](#)
- spectrum0.ar, [5, 11, 32, 33](#)
- start, [35](#)
- start.mcmc (*time.mcmc*), [35](#)
- summary.mcmc, [5, 20, 34](#)
- thin, [20, 34, 35, 41](#)
- thin.mcmc (*time.mcmc*), [35](#)
- time, [35](#)
- time.mcmc, [35](#)
- topo.colors, [9](#)
- traceplot, [26, 36](#)

trellisplots, 36  
ts, 23  
  
update, 40  
  
varnames, 40  
varnames<- (varnames), 40  
  
window, 41  
window.mcmc, 20, 22, 41  
  
xyplot, 39  
xyplot.mcmc (trellisplots), 36