

# Package ‘mcgibbsit’

April 17, 2009

**Title** Warnes and Raftery’s MCGibbsit MCMC diagnostic

**Version** 1.0.5

**Date** 2005-05-23

**Author** Gregory R. Warnes <Gregory.R.Warnes@Pfizer.com>

**Depends** coda

**Description** mcgibbsit provides an implementation of Warnes & Raftery’s MCGibbsit run-length diagnostic for a set of (not-necessarily independent) MCMC samplers. It combines the estimate error-bounding approach of Raftery and Lewis with evaluate between verses within chain approach of Gelman and Rubin.

**Maintainer** Gregory R. Warnes <Gregory.R.Warnes@Pfizer.com>

**License** GPL

**Repository** CRAN

**Date/Publication** 2005-06-22 16:22:49

## R topics documented:

mcgibbsit . . . . .	1
read.mcmc . . . . .	4

<b>Index</b>	<b>6</b>
--------------	----------

## Description

mcgibbsit provides an implementation of Warnes & Raftery's MCGibbsit run-length diagnostic for a set of (not-necessarily independent) MCMC samplers. It combines the estimate error-bounding approach of Raftery and Lewis with the between chain variance verses within chain variance approach of Gelman and Rubin.

## Usage

```
mcgibbsit(data, q=0.025, r=0.0125, s=0.95, converge.eps=0.001,
          correct.cor=TRUE)
```

## Arguments

<code>data</code>	an 'mcmc' object.
<code>q</code>	quantile(s) 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
<code>converge.eps</code>	Precision required for estimate of time to convergence.
<code>correct.cor</code>	should the between-chain correlation correction (R) be computed and applied. Set to false for independent MCMC chains.

## Details

mcgibbsit computes the minimum run length  $N_{min}$ , required burn in  $M$ , total run length  $N$ , run length inflation due to *auto-correlation*,  $I$ , and the run length inflation due to *between-chain* correlation,  $R$  for a set of exchangeable MCMC simulations which need not be independent.

The normal usage is to perform an initial MCMC run of some pre-determined length (e.g., 300 iterations) for each of a set of  $k$  (e.g.,  $k = 20$ ) MCMC samplers. The output from these samplers is then read in to create an `mcmc.list` object and `mcgibbsit` is run specifying the desired accuracy of estimation for quantiles of interest. This will return the minimum number of iterations to achieve the specified error bound. The set of MCMC samplers is now run so that the total number of iterations exceeds this minimum, and `mcgibbsit` is again called. This should continue until the number of iterations already complete is less than the minimum number computed by `mcgibbsit`.

If the initial number of iterations in `data` is too small to perform the calculations, an error message is printed indicating the minimum pilot run length.

The parameters `q`, `r`, `s`, `converge.eps`, and `correct.cor` can be supplied as vectors. This will cause `mcgibbsit` to produce a list of results, with one element produced for each set of values. I.e., setting `q=(0.025,0.975)`, `r=(0.0125,0.005)` will yield a list containing two `mcgibbsit` objects, one computed with parameters `q=0.025`, `r=0.0125`, and the other with `q=0.975`, `r=0.005`.

**Value**

An `mcgibbsit` object with components

<code>call</code>	parameters used to call 'mcgibbsit'
<code>params</code>	values of <code>r</code> , <code>s</code> , and <code>q</code> used
<code>resmatrix</code>	a matrix with 6 columns: <b>Nmin</b> The minimum required sample size for a chain with no correlation between consecutive samples. Positive autocorrelation will increase the required sample size above this minimum value. <b>M</b> The number of 'burn in' iterations to be discarded (total over all chains). <b>N</b> The number of iterations after burn in required to estimate the quantile <code>q</code> to within an accuracy of $\pm r$ with probability <code>p</code> (total over all chains). <b>Total</b> Overall number of iterations required ( $M + N$ ). <b>I</b> An estimate (the 'dependence factor') of the extent to which auto-correlation inflates the required sample size. 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. <b>R</b> An estimate of the extent to which between-chain correlation inflates the required sample size. Large values of 'R' indicate that there is significant correlation between the chains and may be indicative of a lack of convergence or a poor multi-chain algorithm.
<code>nchains</code>	the number of MCMC chains in the data
<code>len</code>	the length of each chain

**Author(s)**

Gregory R. Warnes ([gregory\\_r\\_warnes@groton.pfizer.com](mailto:gregory_r_warnes@groton.pfizer.com)) based on the the R function `raftery.diag` which is part of the 'CODA' library. `raftery.diag`, in turn, is based on the FORTRAN program 'gibbsit' written by Steven Lewis which is available from the Statlib archive.

**References**

- Warnes, G.W. (2004). The Normal Kernel Coupler: An adaptive MCMC method for efficiently sampling from multi-modal distributions (web site), <http://www.analytics.washington.edu/Zope/projects/MCMC/NKC/index.html>.
- Warnes, G.W. (2000). Multi-Chain and Parallel Algorithms for Markov Chain Monte Carlo. Dissertation, Department of Biostatistics, University of Washington,
- 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.

**See Also**

[read.mcmc](#)

## Examples

```
# this is a totally useless example, but it does exercise the code
for(i in 1:20){
  x <- matrix(rnorm(1000),ncol=4)
  x[,4] <- x[,4] + 1/3 * (x[,1] + x[,2] + x[,3])
  colnames(x) <- c("alpha","beta","gamma", "nu")
  write.table(x, file=paste("mcmc",i,"csv",sep="."), sep=",")
}

data <- read.mcmc(20, "mcmc.#.csv", sep=",")

mcgibbsit(data)
```

---

read.mcmc

*Read in data from a set of MCMC runs*

---

## Description

Read in data from a set of MCMC runs and create an `mcmc.list` object.

## Usage

```
read.mcmc( nc, sourcepattern, ...,
           col.names, start = 1,
           end = nrow(tmp)/numComponents*thin,
           thin = 1, numComponents=1)
```

## Arguments

<code>nc</code>	Number of MCMC sampler files to read
<code>sourcepattern</code>	MCMC data file name pattern.
<code>...</code>	Arguments to be passed to <code>read.table</code> when loading MCMC sampler data.
<code>col.names</code>	Data file column names (optional)
<code>start, end, thin</code>	See documentation for <code>mcmc</code>
<code>numComponents</code>	Number of component samplers.

## Details

This function reads in the states output from one or more MCMC samplers and creates a single `mcmc.list` object. `sourcepattern` will be used as a filename pattern with `#` replaced by the sampler number. EG, `sourcepattern="MCMC.#.csv"` will be converted to `"MCMC.1.csv"`, `"MCMC.2.csv"`, etc.

The function `read.table` is used to read in the data. Options for `read.table` may be included as part of the call to `read.mcmc`.

The `start`, `end`, and `thin` arguments can be used to annotate the MCMC samplers with additional information.

**Value**

An `mcmc.list` object containing `nc` component `mcmc` objects.

**Author(s)**

Gregory R. Warnes ([gregory\\_r\\_warnes@groton.pfizer.com](mailto:gregory_r_warnes@groton.pfizer.com))

**See Also**

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

**Examples**

```
# this is a totally useless example, but it does exercise the code
for(i in 1:20){
  x <- matrix(rnorm(1000), ncol=4)
  x[,4] <- x[,4] + 1/3 * (x[,1] + x[,2] + x[,3])
  colnames(x) <- c("alpha", "beta", "gamma", "nu")
  write.table(x, file=paste("mcmc", i, "csv", sep="."), sep=",")
}

data <- read.mcmc(20, "mcmc.\#.csv", sep=",")
```

# Index

\*Topic **file**

read.mcmc, 4

\*Topic **models**

mcgibbsit, 1

mcgibbsit, 1

mcmc, 5

mcmc.list, 5

print.mcgibbsit (*mcgibbsit*), 1

read.mcmc, 3, 4

read.table, 5