Math 459 Lecture 16

Todd Kuffner

Comments on Mixing of Chains

The mixing time of a Markov chain is the number of steps (amount of time) until the chain is close to the stationary distribution. Mixing measures how well the chain moves around around the parameter space.

Fast mixing means the the chain will reach the stationary distribution quickly, regardless of the starting values.

With perfect mixing, MC samples can move from one region of the state space to any other in one step.

Formal definition requires measure theory.

- we can do some visual inspections in practice, and these must be done for *every parameter* (e.g. traceplots, running means plots)

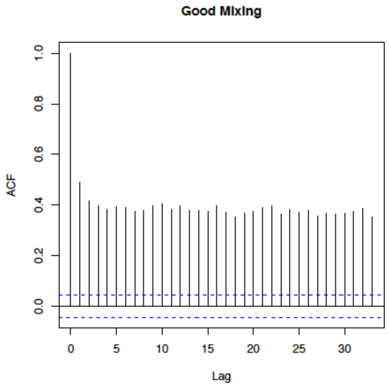
Autocorrelation

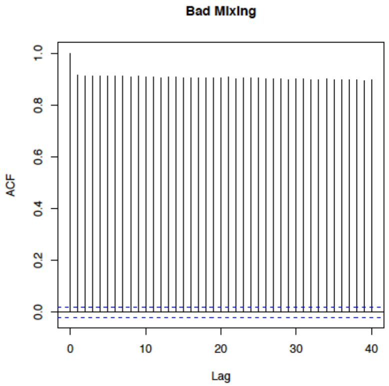
The lag k autocorrelation, denoted ρ_k, is the correlation between each draw and its kth lag:

$$\rho_k = \frac{\sum_{i=1}^{n-k}(x_i - \bar{x})(x_{i+k} - \bar{x})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

We expect the kth lag autocorrelation to be smaller as k increases (i.e. the 3rd and 25th draws should be less correlated than the 3rd and 4th draws).

- if autocorrelation is high for large values of k, this indicates high correlation between the draws and slow mixing.


Good Mixing
ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30
Lag


Bad Mixing
ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
Lag

Comments on Autocorrelation

Any Markov chain will have autocorrelation, since the current value depends on the previous one.

The Ergodic Theorem tells us that inference from *correlated* samples will still be valid, i.e. the Monte Carlo approximation of the posterior quantity of interest will converge to the true value (in probability).

Autocorrelation can still be present in chains that have converged.

- Monte Carlo approximation still works, but you may need a very large sample (long chain) for the approximation to be accurate.

Effective Sample Size

Since not all samples are independent, we have to adjust the sample size when computing the mean.

- since the samples are autocorrelated, and hence dependent, there is less posterior sampled information
- this reduces the precision in estimating the posterior distribution (and hence any posterior quantity)
- proposed by Radford Neal in dsucssion of Kass, Carlin, Gelman & Neal (1998)
- the same effective sample size is used for each marginal posterior (it is computed tkaing this into account)
- there is no ‘convergence’ rule, but in practice it is recommended to have an effective sample size of *at least* 1000

Multi-Parameter Models and Marginal Posteriors

When θ is a vector, obviously MCMC produces a Markov chain for each one. Each chain corresponds to a sample from the *joint posterior*.

Proposal: let's just treat the chain for each parameter as draws from the marginal posterior for that parameter

Problem: We should worry about <u>dependence</u> between the chains, not just within each chain. In general there is non-zero covariance between the chains for each parameter.

Suggestions:

1. run a separate MCMC algorithm for each parameter (very inefficient)
2. re-order the chains (permute within a chain)

In practice: many people just ignore the other chains, and think of this as averaging over them. Why does this work?

Method of Composition

Recall that for any joint density $p(\theta_1, \theta_2)$, we can write

$$p(\theta_1, \theta_2) = p(\theta_1)p(\theta_2|\theta_1)$$

Often each density on the RHS is easy to sample from. To get draws from joint density

1. draw $\theta_1^{(j)}$ from $p(\theta_1)$ (marginal density), then
2. draw $\theta_2^{(j)}$ from $p(\theta_2|\theta_1^{(j)})$

Since $(\theta_1^{(j)}, \theta_2^{(j)})$ is a draw from joint density, then the second component is a draw from the marginal for θ_2

$$\theta_2^{(j)} \sim p(\theta_2) = \int p(\theta_2|\theta_1)p(\theta_1)d\theta_1$$

$\Rightarrow$ to draw $\theta_2^{(j)}$ from $p(\theta_2)$, it suffices to draw from the joint density and retain the second component

Tuning MCMC algorithms

Generally you want to tune the algorithms to get better performance in terms of:

- reducing correlation within each chain, i.e. autocorrelation or serial correlation
- reducing correlation between each chain
- acceptance rate

too high	can increase correlation
too low	takes forever to explore the state space (the support of the posterior)

Burn-In and Thinning

Burn-In discard some portion at the start of the chain; motivations

1. ignore samples before convergence to stationary distribution
2. reduce the effect of the starting value
3. use the burn-in period to tune the algorithm

Thinning thin the chains by keeping only every kth value, e.g. every 20th value; motivations

1. reduce autocorrelation
2. storing only a small portion of draws reduces memory requirements

Thinning has been largely abandoned: (i) unnecessary due to ergodic theorem; (ii) too inefficient; (iii) typically increases variance of Monte Carlo estimates.

Old School Software

`BUGS`: **B**ayesian inference **U**sing **G**ibbs **S**ampling (1989, MRC Biostatistics Unit, University of Cambridge)

- became `WinBUGS` (Microsoft Windows)
- now `OpenBUGS`; interfaces with `R`

Insprired `Stan` and `NIMBLE` which can do Bayesian inference for very general models using MCMC; both interface with `R`.

Slightly different is `JAGS`: **J**ust **A**nother **G**ibbs **S**ampler

Some R packages (see CRAN Task View: Bayesian)

MCMCpack is a general purpose R package for MCMC

rstan is also general purpose and up-to-date

Output is a coda mcmc object; convenient for convergence diagnostics

coda: Convergence Diagnosis and Output Analysis; \ boa: Bayesian Output Analysis is another option

RWMH: univariate gamma density (with coda analysis)

```r
library(coda)
mh.gamma <- function(n.sims, start, burnin, cand.sd, shape, rate){
  theta.cur <- start
  draws <- c()
    theta.update <- function(theta.cur, shape, rate){
    theta.can <- rnorm(1, mean=theta.cur, sd=cand.sd)
    accept.prob <- dgamma(theta.can, shape=shape,
      rate=rate)/dgamma(theta.cur, shape=shape, rate=rate)
    if(runif(1) <= accept.prob)
      theta.can
    else
      theta.cur
  }
  for(i in 1:n.sims){
    draws[i] <- theta.cur <- theta.update(theta.cur, shape = shape,
                                            rate = rate)
  }
  res <- mcmc(draws[(burnin + 1):n.sims])
  cat("Acceptance Rate:", 1-rejectionRate(res), "\n")
  return(res)
}
```

Some comments

`mcmc` coerces the output from our RWMH code into a `coda` object of class `mcmc`

`rejectionRate` calculates the fraction of iterations that a M-H-type chain rejected the candidate/proposed move

- this 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 returned values should be the same

```r
mh.draws <- mh.gamma(10000, start = 1, burnin = 1000,
                    cand.sd = 1, shape = 1.7, rate = 4.4)
```

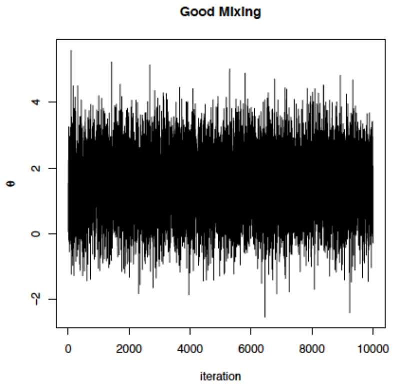
```
Acceptance Rate: 0.2752528
```

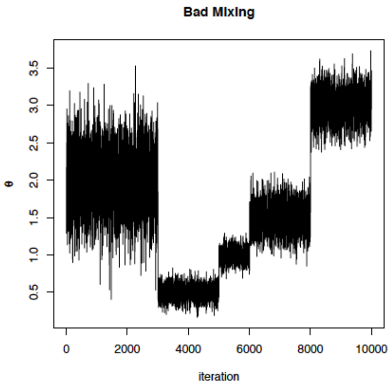
```r
gamma.samp <- rgamma(10000, shape = 1.7, rate = 4.4)
gamma.draws <- as.mcmc(gamma.samp)
# to convert the gamma sample to an mcmc object
```

Traceplots

Plots iteration number against the value of the draw of the parameter for each iteration.

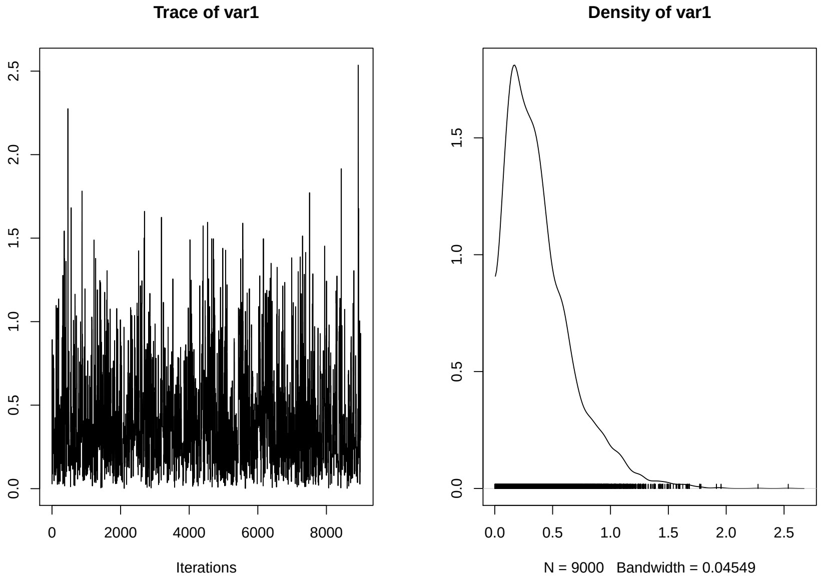
- helps us see if the chain gets stuck in a particular region of the parameter space, which indicates bad mixing





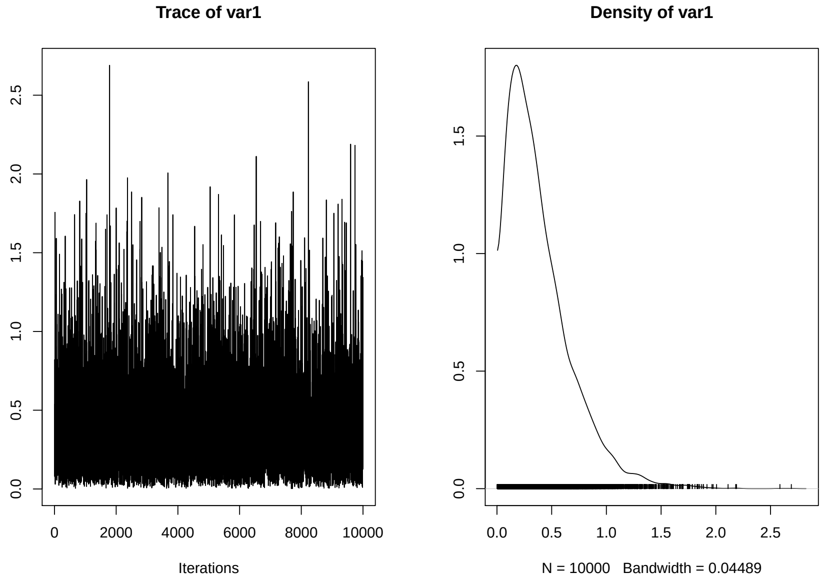
Plot of M-H Draws

```r
plot(mh.draws)   # calls both traceplot(), densplot()
```



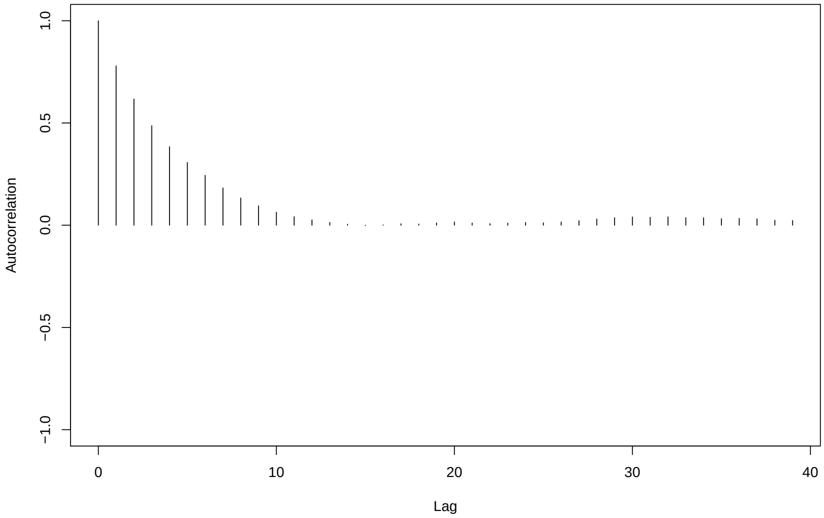
Plot of `rgamma` Draws

```r
plot(gamma.draws)
```



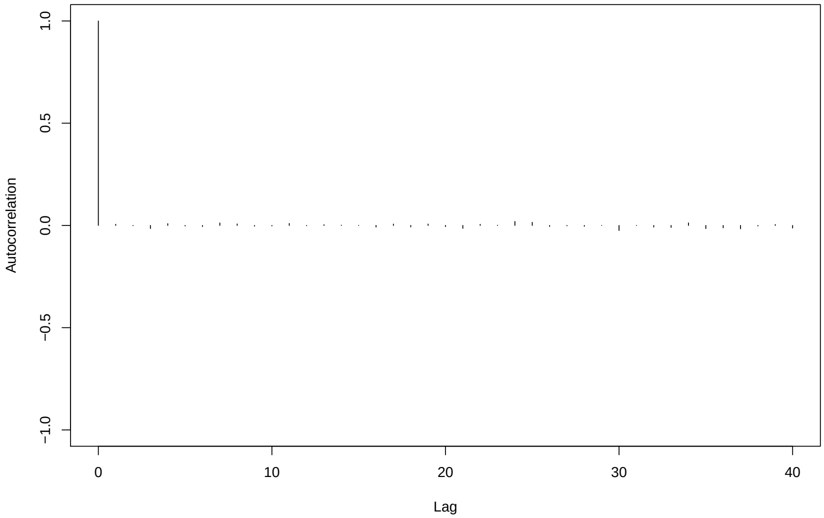
Autocorrelation Plot of M-H Draws

```
autocorr.plot(mh.draws)
```



Autocorrelation Plot of `rgamma` Draws

```r
autocorr.plot(gamma.draws)
```



1-D Gibbs example for normal model, unknown mean and variance

```r
# Conjugate priors: mu~N(m0,1/t0), sigma2<-Inv-chisquare(a, b)
# Generate data
n <- 50
mu <- 3
sigma <- 3 ; sigma2 <- sigma^2
y <- rnorm(n,mu,sigma)
# Prior hyperparameters
m0 <- 0
t0 <- 0       # Noninformative
a <- b <- 0   # Noninformative
# Initial values
tau <- 1
mtau <- a+n/2
# Create empty vectors to store the results
nsim <- 5000
Tau <- Mu <- Sigma <- rep(0,nsim)
```

```r
for (i in 1:nsim) {
  v<-1/(tau*n + t0)
  m<-v*(tau*sum(y)+t0*m0)
  mu<- Mu[i]<-rnorm(1,m,sqrt(v))

  tau<-rgamma(1,mtau,b+sum((y-mu)^2)/2)
  Sigma[i]<-1/tau
}
```

2-D Random Walk Metropolis-Hastings

```r
NormalMHExample <- function(n.sim, n.burnin) {
library(mvtnorm)
mu.vector <- c(3, 1)
variance.matrix <- cbind(c(1, -0.5), c(-0.5, 2))
theta.mh <- matrix(NA, nrow = n.sim, ncol = 2)
theta.current <- rnorm(n = 2, mean = 0, sd = 4)
theta.update <- function(index, theta.current, ...) {
theta.star <- rnorm(n = 1, mean = theta.current[index], sd = 2)
if (index == 1)
theta.temp <- c(theta.star, theta.current[2])
else theta.temp <- c(theta.current[1], theta.star)
r <- dmvnorm(theta.temp, mu.vector, variance.matrix)/dmvnorm(theta.current,
mu.vector, variance.matrix)
r <- min(r, 1, na.rm = T)
if (runif(1) < r)
theta.star
else theta.current[index]
}
for (i in 1:n.sim) {
theta.current[1] <- theta.mh[i, 1] <- theta.update(1, theta.current,
mu.vector, variance.matrix)
theta.current[2] <- theta.mh[i, 2] <- theta.update(2, theta.current,
mu.vector, variance.matrix)
}
theta.mh <- theta.mh[(n.burnin + 1):n.sim, ]
}
mh.draws <- NormalMHExample(n.sim = 5000, n.burnin = 0)
```

Convergence

The theory of Markov chains tells us that eventually (under some assumptions) the chain converges to its stationary distribution, which is also the target distribution.

- However there is no guarantee that the chain has converged after M draws, even for a very large M

How do we know that the chain has converged?

We can never be certain, but we have some tools that help us determine if the chain *appears* to have converged.

Convergence Diagnostics using `coda` in `R`

First need to turn the chain into an `mcmc` object:

```
mh.draws <- mcmc(mh.draws)
```

The `mcmc()` function has several other options:

- `start` and `end` allow us to select burn-in (to drop those values)
- `thin` specifies the thinning interval that was used (it doesn't actually thin the chain, that would have been specified when the chain was produced)

Summarizing the Posterior Density

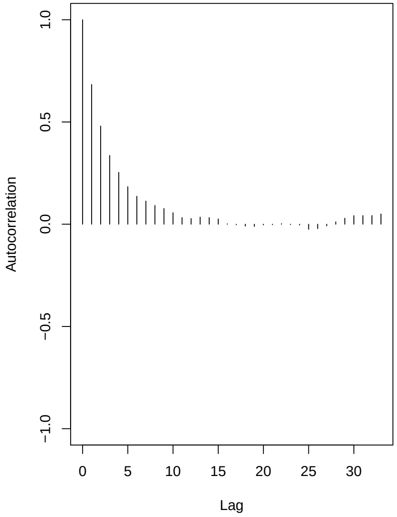
Using `summary()` for an `mcmc` object produces summary statistics.

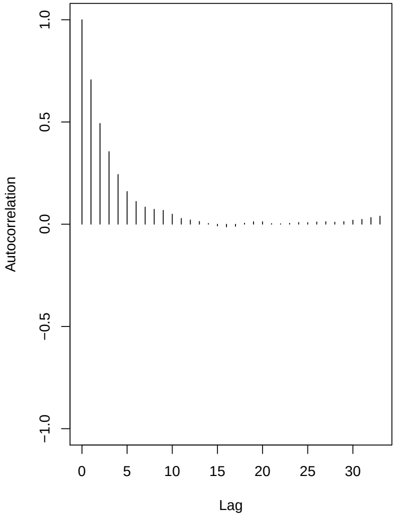
```
summary(mh.draws)
```

```
Iterations = 1:5000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 5000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

      Mean    SD Naive SE Time-series SE
[1,] 2.985 1.033  0.01460        0.03566
[2,] 1.006 1.402  0.01983        0.04710

2. Quantiles for each variable:

       2.5%     25%   50%   75% 97.5%
var1  0.965 2.31153 2.956 3.694 4.954
var2 -1.717 0.04265 0.995 1.936 3.713
```

Autocorrelation plots

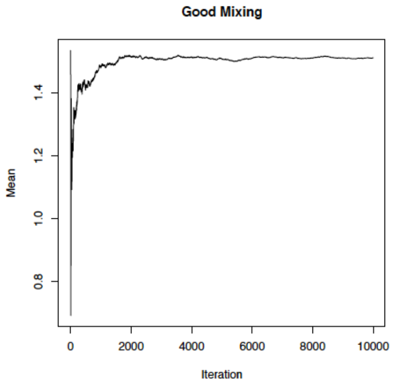
```r
autocorr.plot(mh.draws)
```

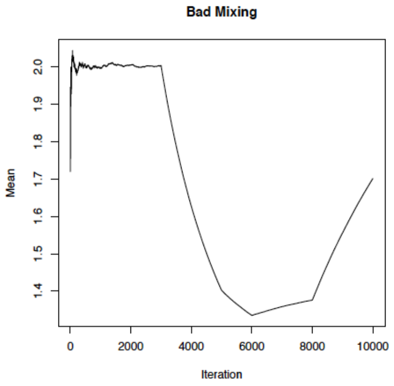




Running Means Plot with `ggmcmc`

Plot of iterations against mean of the draws up to that iteration.





Comments on M-H Tuning

The proposal variance (or covariance matrix) is very important.

- a small proposal variance gives a high acceptance rate, but the chain ‘mixes’ slowly: the chain moves very slowly around the parameter space and takes longer to converge
- a large proposal variance gives a lower acceptance rate, which means more iterations are required to produce a desired number of samples from the posterior

You may come across *optimal acceptance rates*, e.g. 23.4%: while these arise from great ideas, they are not helpful in practical problems, since they assume posterior independence of the parameters (i.e. the target factors into a product of marginal densities).

Gelman and Rubin Multiple Sequence Diagnostic

Follow these steps for each parameter:

1. Run $m \geq 2$ chains of length $2n$ from overdispersed starting values.
2. Discard the first n draws in each chain.
3. Compute the within-chain and between-chain variance.
4. Compute the estimated variation of the parameter as a *weighted sum* of the within-chain and between-chain variance.
5. Compute the potential scale reduction factor.

Within-Chain Variance

Define

$$W = \frac{1}{m} \sum_{j=1}^{m} s_j^2,$$

with

$$s_j^2 = \frac{1}{n-1} \sum_{i=1}^{n} (\theta_{ij} - \bar{\theta}_j)^2$$

- s_j^2 is the variance of the jth chain
- W is the average of the variances of each chain
- W often underestimates the true variance of the stationary distribution, since the chains have typically not reached *all* points of the stationary distribution

Between-Chain Variance

Define

$$B = \frac{n}{m-1} \sum_{j=1}^{m} (\bar{\theta}_j - \bar{\bar{\theta}})^2,$$

with

$$\bar{\bar{\theta}} = \frac{1}{m} \sum_{j=1}^{m} \bar{\theta}_j$$

- this is the variance of the means of the chains multiplied by n because each chain is based on n draws

Estimated Variance

We then estimate the variance of the stationary distribution using a *weighted average* of W and B:

$$\widehat{\mathrm{Var}(\theta)} = (1 - \frac{1}{n})W + \frac{1}{n}B$$

- the overdispersion of the starting values makes this *overestimate* the true variance, but it is unbiased if the starting distribution $\underline{\text{equals}}$ the stationary distribution (if the starting values were not overdispersed)

Potential Scale Reduction Factor

The potential scale reduction factor is defined as:

$$\hat{R} = \sqrt{\frac{\widehat{\mathrm{Var}(\theta)}}{W}}$$

- for large $\hat{R}$, e.g. $\hat{R} > 1.1$, we should run our chains longer to improve convergence to the stationary distribution
- for more than one parameter, we compute the potential scale reduction factor for each parameter
- we should run chains longe enough so that <u>all</u> potential scale reduction factors are small enough
- then combine the mn total draws from our chains to produce one chain from the stationary distribution

Gelman and Rubin diagnostic in R

Run M chains at different (overdispersed) starting values (here $M = 3$), convert them to `mcmc` objects and combine them in an `mcmc.list`.

```r
mh.draws1 <- mcmc(NormalMHExample(n.sim = 5000, n.burnin = 0))
mh.draws2 <- mcmc(NormalMHExample(n.sim = 5000, n.burnin = 0))
mh.draws3 <- mcmc(NormalMHExample(n.sim = 5000, n.burnin = 0))
mh.draws4 <- mcmc(NormalMHExample(n.sim = 5000, n.burnin = 0))
mh.draws5 <- mcmc(NormalMHExample(n.sim = 5000, n.burnin = 0))
mh.list <- mcmc.list(list(mh.draws1, mh.draws2, mh.draws3,
                          mh.draws4, mh.draws5))
```

```r
gelman.diag(mh.list)
```

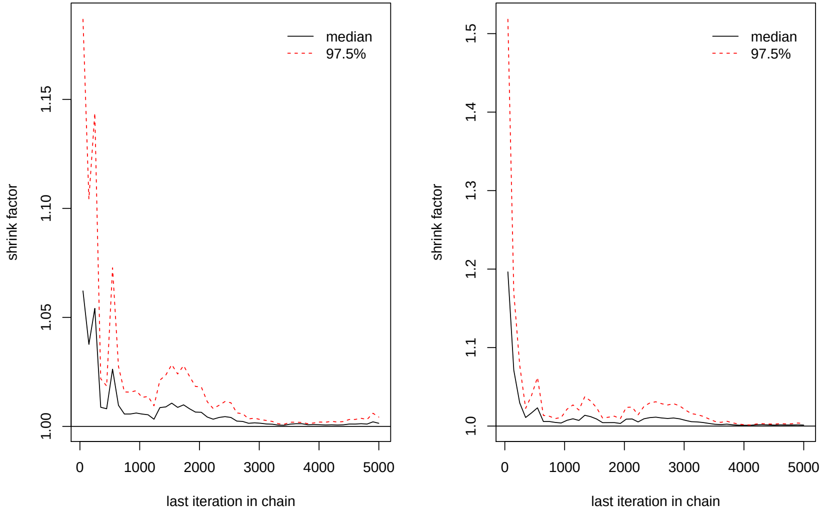
```
Potential scale reduction factors:

     Point est. Upper C.I.
[1,]          1          1
[2,]          1          1

Multivariate psrf

1
```

This gives the median scale reduction factor and its 97.5% quantile (the PSRF is estimated with uncertainty since the chain lengths are finite).

Also reports multivariate scale reduction factor (Gelman & Brooks, 1997).

Plotting how PSRF Changes through Iteration

```r
gelman.plot(mh.list)
```



Geweke Diagnostic

Take two non-overlapping parts (typically the first 0.1 and last 0.5 proportions) of the Markov chain and compares the means of both parts.

- uses a difference-of-means test with null hypothesis that they are from the same distribution
- standard Z-score test with standard error adjusted for autocorrelation

```
geweke.diag(mh.draws)
```

```
Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

  var1   var2 
 1.992 -1.419 
```

Raftery and Lewis Diagnostic

Suppose we wish to measure a posterior quantile of interest, q.

- We define an acceptable tolerance r for q and a probability s of being within that tolerance.
- the Raftery and Lewis diagnostic computes the *number of iterations* N and the *number of burn-ins* M necessary to satisfy the specified conditions.

This diagnostic was designed to test the number of iterations and burn-in needed by first running and testing a shorter 'pilot' chain.

- in practice, we just test the usual chain and see if it satisfies the results suggested by the diagnostic

Inputs

1. Select a posterior quantile of interest q (e.g. the 0.025 quantile)
2. Select an acceptable tolerance r for this quantile; e.g. $r = 0.005$ means we want to measure the 0.025 quantile with accuracy of ± 0.005
3. Select a probability s, which is the desired probability of being within $(q - r, q + r)$
4. Run a 'pilot' sampler to generate a Markov chain of minimum length given by *rounding up*

$$n_{\min} = \left[\Phi^{-1} \left(\frac{s+1}{2} \right) \frac{\sqrt{q(1-q)}}{r} \right]^2$$

with Φ^{-1} the inverse of the standard normal CDF (i.e. the standard normal quantile function).

```
raftery.diag(mh.draws,q=0.025,r=0.005,s=0.95)
```

```
Quantile (q) = 0.025
Accuracy (r) = +/- 0.005
Probability (s) = 0.95

 Burn-in   Total Lower bound   Dependence
 (M)       (N)   (Nmin)        factor (I)
 15        16073 3746          4.29
 12        13176 3746          3.52
```

- M: number of burn-ins necessary
- N: number of iterations necessary in the Markov chain
- N_{min}: minimum number of iterations for the 'pilot' sampler
- I: dependence factor, interpreted as the proportional increase in the number of iterations attributable to autocorrelation (serial dependence)
- high dependence factors (e.g. > 5) cause concern: may be due to influential starting values, high correlations between parameters, or poor mixing

Comments

The Raftery-Lewis diagnostic differs by the quantile q chosen.

Estimates are typically conservative in the sense of suggesting more iterations than are really necessary.

Only test the *marginal* convergence on each parameter.

Heidelberg and Welch Diagnostic

Calculates a test statistic (based on Cramer-von Mises test statistic) to reject or fail to reject the null hypothesis that the Markov chain is from a stationary distribution.
Two-parts to this diagnostic.

Part One

1. Generate a chain of N iterations and define an α level.
2. Calculate the test statistic on the whole chain. Reject or fail to reject the null that the chain is from a stationary distribution.
3. If null is rejected, discard the first 10% of the chain. Calculate the test statistic and again reject or fail to reject the new null.
4. If null is rejected, discard the the next 10% and test again on the new null.
5. Repeat until the null hypothesis is not rejected at level α, or until 50% is discarded. If the test still rejects the null, then the chain must be run longer.

Part Two

If the chain 'passes' the first part, then keep the part of the chain which was not discarded for use in the second part.

The halfwidth test calculates half the width of the $(1 - \alpha)\%$ credible interval around the mean.

If the ratio of the halfwidth and the mean is lower than some ϵ, then the chain 'passes' the test. Otherwise the chain must be run longer.

Default ϵ is 0.1 and default α is 0.05.

```r
heidel.diag(mh.draws)
```

```
     Stationarity start     p-value
     test         iteration
var1 passed       1         0.117
var2 passed       1         0.165

     Halfwidth Mean Halfwidth
     test
var1 passed    2.98 0.0699
var2 passed    1.01 0.0923
```

Coming Up

Bayesian linear regression models.

Bayesian model comparison and selection.

Reversible Jump MCMC