Convergence Diagnostics

Patrick Lam

Outline

Outline

Running Example

Running Example

As a running example, we will use the bivariate normal Metropolis sampler from class.

Running Example

As a running example, we will use the bivariate normal Metropolis sampler from class.

```
> 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

Convergence

From our theory of Markov chains, we expect our chains to eventually converge to the stationary distribution, which is also our target distribution.

Convergence

From our theory of Markov chains, we expect our chains to eventually converge to the stationary distribution, which is also our target distribution.

However, there is no guarantee that our chain has converged after M draws.

Convergence

From our theory of Markov chains, we expect our chains to eventually converge to the stationary distribution, which is also our target distribution.

However, there is no guarantee that our chain has converged after M draws.

How do we know whether our chain has actually converged?

Convergence

From our theory of Markov chains, we expect our chains to eventually converge to the stationary distribution, which is also our target distribution.

However, there is no guarantee that our chain has converged after M draws.

How do we know whether our chain has actually converged?

We can never be sure, but there are several tests we can do, both visual and statistical, to see if the chain appears to be converged.

Convergence Diagnostics in R

All the diagnostics we will use are in the `coda` package in R.

Convergence Diagnostics in R

All the diagnostics we will use are in the `coda` package in R.

```
> library(coda)
```

Convergence Diagnostics in R

All the diagnostics we will use are in the `coda` package in R.

```
> library(coda)
```

Before we use the diagnostics, we should turn our chains into `mcmc` objects.

Convergence Diagnostics in R

All the diagnostics we will use are in the `coda` package in R.

```
> library(coda)
```

Before we use the diagnostics, we should turn our chains into `mcmc` objects.

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

Convergence Diagnostics in R

All the diagnostics we will use are in the `coda` package in R.

```
> library(coda)
```

Before we use the diagnostics, we should turn our chains into `mcmc` objects.

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

We can tell the `mcmc()` function to burn-in or drop draws with the `start` and `end` arguments.

Convergence Diagnostics in R

All the diagnostics we will use are in the `coda` package in R.

```
> library(coda)
```

Before we use the diagnostics, we should turn our chains into `mcmc` objects.

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

We can tell the `mcmc()` function to burn-in or drop draws with the `start` and `end` arguments.

`mcmc()` also has a `thin` argument, which only tells it the thinning interval that was used (it does not actually thin for us).

Posterior Summary

Posterior Summary

We can do `summary()` of an `mcmc` object to get summary statistics for the posterior.

Posterior Summary

We can do `summary()` of an `mcmc` object to get summary statistics for the posterior.

```
> 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,] 3.0282 1.027  0.01453        0.03859
[2,] 0.9997 1.424  0.02014        0.04109

2. Quantiles for each variable:

       2.5%     25%    50%   75% 97.5%
var1  0.864 2.37214 3.0489 3.733 5.016
var2 -1.622 0.03447 0.9984 1.927 3.777
```

Posterior Summary

We can do `summary()` of an `mcmc` object to get summary statistics for the posterior.

```
> 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,] 3.0282 1.027  0.01453        0.03859
[2,] 0.9997 1.424  0.02014        0.04109

2. Quantiles for each variable:

       2.5%     25%    50%   75% 97.5%
var1  0.864 2.37214 3.0489 3.733 5.016
var2 -1.622 0.03447 0.9984 1.927 3.777
```

The results give the posterior means, posterior standard deviations, and posterior quantiles for each variable.

The “naïve” standard error is the **standard error of the mean**, which captures *simulation error* of the mean rather than posterior uncertainty.

The “naïve” standard error is the **standard error of the mean**, which captures *simulation error* of the mean rather than posterior uncertainty.

$$\text{naive SE} = \frac{\text{posterior SD}}{\sqrt{n}}$$

The “naïve” standard error is the **standard error of the mean**, which captures *simulation error* of the mean rather than posterior uncertainty.

$$\text{naive SE} = \frac{\text{posterior SD}}{\sqrt{n}}$$

The time-series standard error adjusts the “naïve” standard error for autocorrelation.

Outline

Mixing

Mixing

One way to see if our chain has converged is to see how well our chain is **mixing**, or moving around the parameter space.

Mixing

One way to see if our chain has converged is to see how well our chain is **mixing**, or moving around the parameter space.

If our chain is taking a long time to move around the parameter space, then it will take longer to converge.

Mixing

One way to see if our chain has converged is to see how well our chain is **mixing**, or moving around the parameter space.

If our chain is taking a long time to move around the parameter space, then it will take longer to converge.

We can see how well our chain is mixing through visual inspection.

Mixing

One way to see if our chain has converged is to see how well our chain is **mixing**, or moving around the parameter space.

If our chain is taking a long time to move around the parameter space, then it will take longer to converge.

We can see how well our chain is mixing through visual inspection.

We need to do the inspections for every parameter.

Traceplots

Traceplots

A **traceplot** is a plot of the iteration number against the value of the draw of the parameter at each iteration.

Traceplots

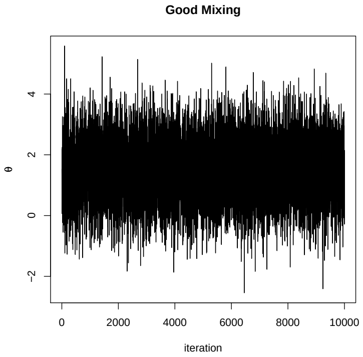
A **traceplot** is a plot of the iteration number against the value of the draw of the parameter at each iteration.

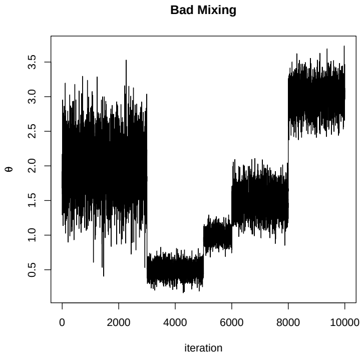
We can see whether our chain gets stuck in certain areas of the parameter space, which indicates bad mixing.

Traceplots

A **traceplot** is a plot of the iteration number against the value of the draw of the parameter at each iteration.

We can see whether our chain gets stuck in certain areas of the parameter space, which indicates bad mixing.





Traceplots and Density Plots

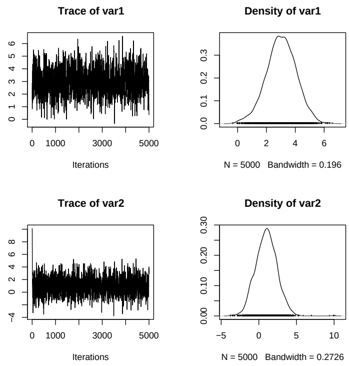
Traceplots and Density Plots

We can do traceplots and density plots by plotting an `mcmc` object or by calling the `traceplot()` and `densplot()` functions.

Traceplots and Density Plots

We can do traceplots and density plots by plotting an `mcmc` object or by calling the `traceplot()` and `densplot()` functions.

```
> plot(mh.draws)
```



Running Mean Plots

Running Mean Plots

We can also use **running mean plots** to check how well our chains are mixing.

Running Mean Plots

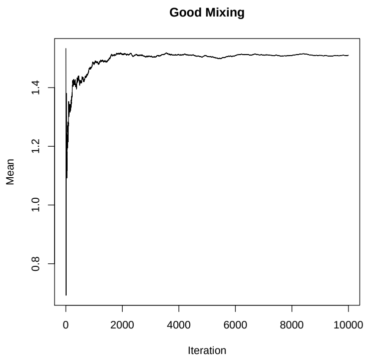
We can also use **running mean plots** to check how well our chains are mixing.

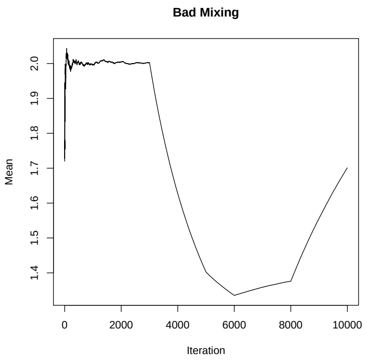
A running mean plot is a plot of the iterations against the mean of the draws up to each iteration.

Running Mean Plots

We can also use **running mean plots** to check how well our chains are mixing.

A running mean plot is a plot of the iterations against the mean of the draws up to each iteration.





Autocorrelation

Autocorrelation

Another way to assess convergence is to assess the autocorrelations between the draws of our Markov chain.

Autocorrelation

Another way to assess convergence is to assess the autocorrelations between the draws of our Markov chain.

The lag k autocorrelation ρ_k is the correlation between every draw and its kth lag:

Autocorrelation

Another way to assess convergence is to assess the autocorrelations between the draws of our Markov chain.

The lag k autocorrelation ρ_k is the correlation between every draw and its kth lag:

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

Autocorrelation

Another way to assess convergence is to assess the autocorrelations between the draws of our Markov chain.

The lag k autocorrelation ρ_k is the correlation between every 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 would expect the kth lag autocorrelation to be smaller as k increases

Autocorrelation

Another way to assess convergence is to assess the autocorrelations between the draws of our Markov chain.

The lag k autocorrelation ρ_k is the correlation between every 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 would expect the kth lag autocorrelation to be smaller as k increases (our 2nd and 50th draws should be less correlated than our 2nd and 4th draws).

Autocorrelation

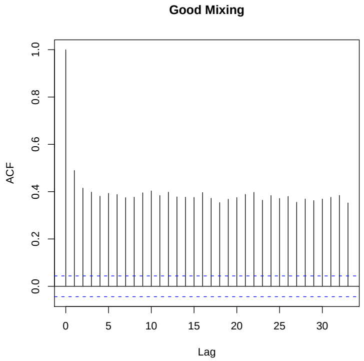
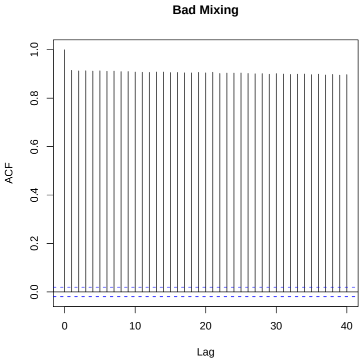
Another way to assess convergence is to assess the autocorrelations between the draws of our Markov chain.

The lag k autocorrelation ρ_k is the correlation between every 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 would expect the kth lag autocorrelation to be smaller as k increases (our 2nd and 50th draws should be less correlated than our 2nd and 4th draws).

If autocorrelation is still relatively high for higher values of k, this indicates high degree of correlation between our draws and slow mixing.


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

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

Autocorrelation Plots

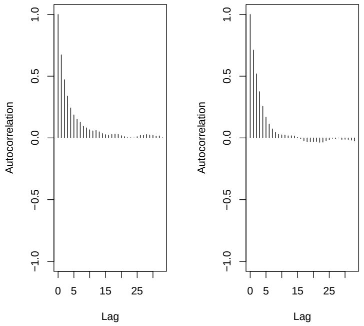
Autocorrelation Plots

We can get autocorrelation plots using the `autocorr.plot()` function.

Autocorrelation Plots

We can get autocorrelation plots using the `autocorr.plot()` function.

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



Rejection Rate for Metropolis-Hastings Algorithm

Rejection Rate for Metropolis-Hastings Algorithm

We can also get a rejection rate for the Metropolis-Hastings algorithm using the `rejectionRate()` function.

Rejection Rate for Metropolis-Hastings Algorithm

We can also get a rejection rate for the Metropolis-Hastings algorithm using the `rejectionRate()` function.

```
> rejectionRate(mh.draws)

     var1      var2
0.5277055 0.4094819
```

Rejection Rate for Metropolis-Hastings Algorithm

We can also get a rejection rate for the Metropolis-Hastings algorithm using the `rejectionRate()` function.

```
> rejectionRate(mh.draws)

     var1      var2
0.5277055 0.4094819
```

To get the acceptance rate, we just want $1-$ rejection rate.

Rejection Rate for Metropolis-Hastings Algorithm

We can also get a rejection rate for the Metropolis-Hastings algorithm using the `rejectionRate()` function.

```
> rejectionRate(mh.draws)

     var1      var2
0.5277055 0.4094819
```

To get the acceptance rate, we just want $1-$ rejection rate.

```
> acceptance.rate <- 1 - rejectionRate(mh.draws)
> acceptance.rate

     var1      var2
0.4722945 0.5905181
```

Outline

Gelman and Rubin Multiple Sequence Diagnostic

Gelman and Rubin Multiple Sequence Diagnostic

Steps (for each parameter):

Gelman and Rubin Multiple Sequence Diagnostic

Steps (for each parameter):

1. Run $m \geq 2$ chains of length $2n$ from overdispersed starting values.

Gelman and Rubin Multiple Sequence Diagnostic

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.

Gelman and Rubin Multiple Sequence Diagnostic

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. Calculate the within-chain and between-chain variance.

Gelman and Rubin Multiple Sequence Diagnostic

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. Calculate the within-chain and between-chain variance.
4. Calculate the estimated variance of the parameter as a weighted sum of the within-chain and between-chain variance.

Gelman and Rubin Multiple Sequence Diagnostic

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. Calculate the within-chain and between-chain variance.
4. Calculate the estimated variance of the parameter as a weighted sum of the within-chain and between-chain variance.
5. Calculate the potential scale reduction factor.

Within Chain Variance

Within Chain Variance

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

where

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

Within Chain Variance

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

where

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

s_j^2 is just the formula for the variance of the jth chain.

Within Chain Variance

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

where

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

s_j^2 is just the formula for the variance of the jth chain. W is then just the mean of the variances of each chain.

Within Chain Variance

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

where

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

s_j^2 is just the formula for the variance of the jth chain. W is then just the mean of the variances of each chain.

W likely underestimates the true variance of the stationary distribution since our chains have probably not reached all the points of the stationary distribution.

Between Chain Variance

Between Chain Variance

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

where

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

Between Chain Variance

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

where

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

This is the variance of the chain means multiplied by n because each chain is based on n draws.

Estimated Variance

Estimated Variance

We can then estimate the variance of the stationary distribution as a weighted average of W and B.

Estimated Variance

We can then estimate the variance of the stationary distribution as a weighted average of W and B.

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

Estimated Variance

We can then estimate the variance of the stationary distribution as a weighted average of W and B.

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

Because of overdispersion of the starting values, this overestimates the true variance, but is unbiased if the starting distribution equals the stationary distribution (if starting values were not overdispersed).

Potential Scale Reduction Factor

Potential Scale Reduction Factor

The potential scale reduction factor is

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

Potential Scale Reduction Factor

The potential scale reduction factor is

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

When $\hat{R}$ is high (perhaps greater than 1.1 or 1.2), then we should run our chains out longer to improve convergence to the stationary distribution.

If we have more than one parameter, then we need to calculate the potential scale reduction factor for each parameter.

If we have more than one parameter, then we need to calculate the potential scale reduction factor for each parameter.

We should run our chains out long enough so that all the potential scale reduction factors are small enough.

If we have more than one parameter, then we need to calculate the potential scale reduction factor for each parameter.

We should run our chains out long enough so that all the potential scale reduction factors are small enough.

We can then combine the mn total draws from our chains to produce one chain from the stationary distribution.

We can do the **Gelman and Rubin diagnostic** in R.

We can do the **Gelman and Rubin diagnostic** in R.

First, we run M chains at different (should be overdispersed) starting values ($M = 5$ here) and convert them to `mcmc` objects.

We can do the **Gelman and Rubin diagnostic** in R.

First, we run M chains at different (should be overdispersed) starting values ($M = 5$ here) and convert them to `mcmc` objects.

```
> 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))
```

We can do the **Gelman and Rubin diagnostic** in R.

First, we run M chains at different (should be overdispersed) starting values ($M = 5$ here) and convert them to `mcmc` objects.

```
> 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))
```

We then put the M chains together into an `mcmc.list`.

We can do the **Gelman and Rubin diagnostic** in R.

First, we run *M* chains at different (should be overdispersed) starting values ($M = 5$ here) and convert them to `mcmc` objects.

```
> 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))
```

We then put the *M* chains together into an `mcmc.list`.

```
> mh.list <- mcmc.list(list(mh.draws1, mh.draws2, mh.draws3, mh.draws4,
+     mh.draws5))
```

We then run the diagnostic with the `gelman.diag()` function.

[1] Brooks, Stephen P. and Andrew Gelman. 1997. "General methods for monitoring convergence of iterative simulations." *Journal of Computational and Graphical Statistics* 7: 434-455.

We then run the diagnostic with the `gelman.diag()` function.

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

Potential scale reduction factors:

     Point est. 97.5% quantile
[1,]       1.00           1.00
[2,]       1.00           1.00

Multivariate psrf

1.00
```

[1] Brooks, Stephen P. and Andrew Gelman. 1997. "General methods for monitoring convergence of iterative simulations." *Journal of Computational and Graphical Statistics* 7: 434-455.

We then run the diagnostic with the `gelman.diag()` function.

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

Potential scale reduction factors:

     Point est. 97.5% quantile
[1,]       1.00           1.00
[2,]       1.00           1.00

Multivariate psrf

1.00
```

The results give us the median potential scale reduction factor and its 97.5% quantile (the psrf is estimated with uncertainty because our chain lengths are finite).

[1] Brooks, Stephen P. and Andrew Gelman. 1997. "General methods for monitoring convergence of iterative simulations." *Journal of Computational and Graphical Statistics* 7: 434-455.

We then run the diagnostic with the `gelman.diag()` function.

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

Potential scale reduction factors:

     Point est. 97.5% quantile
[1,]       1.00           1.00
[2,]       1.00           1.00

Multivariate psrf

1.00
```

The results give us the median potential scale reduction factor and its 97.5% quantile (the psrf is estimated with uncertainty because our chain lengths are finite).

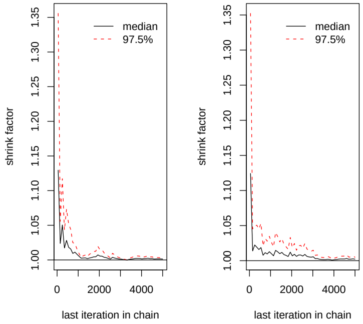
We also get a multivariate potential scale reduction factor that was proposed by Gelman and Brooks.[1]

[1] Brooks, Stephen P. and Andrew Gelman. 1997. "General methods for monitoring convergence of iterative simulations." *Journal of Computational and Graphical Statistics* 7: 434-455.

We can see how the potential scale reduction factor changes through the iterations using the `gelman.plot()` function.

We can see how the potential scale reduction factor changes through the iterations using the `gelman.plot()` function.

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



Outline

Geweke Diagnostic

Geweke Diagnostic

The **Geweke diagnostic** takes two nonoverlapping parts (usually the first 0.1 and last 0.5 proportions) of the Markov chain and compares the means of both parts, using a difference of means test to see if the two parts of the chain are from the same distribution (null hypothesis).

Geweke Diagnostic

The **Geweke diagnostic** takes two nonoverlapping parts (usually the first 0.1 and last 0.5 proportions) of the Markov chain and compares the means of both parts, using a difference of means test to see if the two parts of the chain are from the same distribution (null hypothesis).

The test statistic is a standard Z-score with the standard errors adjusted for autocorrelation.

Geweke Diagnostic

The **Geweke diagnostic** takes two nonoverlapping parts (usually the first 0.1 and last 0.5 proportions) of the Markov chain and compares the means of both parts, using a difference of means test to see if the two parts of the chain are from the same distribution (null hypothesis).

The test statistic is a standard Z-score with the standard errors adjusted for autocorrelation.

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

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

  var1   var2
0.6149 0.1035
```

Outline

Raftery and Lewis Diagnostic

Raftery and Lewis Diagnostic

Suppose we want to measure some posterior quantile of interest q.

Raftery and Lewis Diagnostic

Suppose we want to measure some posterior quantile of interest q.

If we define some acceptable tolerance r for q and a probability s of being within that tolerance, the **Raftery and Lewis diagnostic** will calculate the number of iterations N and the number of burn-ins M necessary to satisfy the specified conditions.

Raftery and Lewis Diagnostic

Suppose we want to measure some posterior quantile of interest q.

If we define some acceptable tolerance r for q and a probability s of being within that tolerance, the **Raftery and Lewis diagnostic** will calculate the number of iterations N and the number of burn-ins M necessary to satisfy the specified conditions.

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

Raftery and Lewis Diagnostic

Suppose we want to measure some posterior quantile of interest q.

If we define some acceptable tolerance r for q and a probability s of being within that tolerance, the **Raftery and Lewis diagnostic** will calculate the number of iterations N and the number of burn-ins M necessary to satisfy the specified conditions.

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

In practice, we can also just test our normal chain to see if it satisfies the results that the diagnostic suggests.

Inputs:

Inputs:

1. Select a posterior quantile of interest q (for example, the 0.025 quantile).

Inputs:

1. Select a posterior quantile of interest q (for example, the 0.025 quantile).
2. Select an acceptable tolerance r for this quantile

Inputs:

1. Select a posterior quantile of interest q (for example, the 0.025 quantile).
2. Select an acceptable tolerance r for this quantile (for example, if $r = 0.005$, then that means we want to measure the 0.025 quantile with an accuracy of ± 0.005).

Inputs:

1. Select a posterior quantile of interest q (for example, the 0.025 quantile).
2. Select an acceptable tolerance r for this quantile (for example, if $r = 0.005$, then that means we want to measure the 0.025 quantile with an accuracy of ± 0.005).
3. Select a probability s, which is the desired probability of being within (q-r, q+r).

Inputs:

1. Select a posterior quantile of interest q (for example, the 0.025 quantile).
2. Select an acceptable tolerance r for this quantile (for example, if $r = 0.005$, then that means we want to measure the 0.025 quantile with an 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

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

where $\Phi^{-1}(\cdot)$ is the inverse of the normal CDF.

Results:

Results:

```
> 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)
 18       20149 3746         5.38
 13       14570 3746         3.89
```

Results:

```
> 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)
 18       20149 3746         5.38
 13       14570 3746         3.89
```

- M: number of burn-ins necessary

Results:

```
> 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)
 18       20149 3746         5.38
 13       14570 3746         3.89
```

- M: number of burn-ins necessary
- N: number of iterations necessary in the Markov chain

Results:

```
> 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)
 18       20149 3746         5.38
 13       14570 3746         3.89
```

- 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

Results:

```
> 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)
 18       20149 3746         5.38
 13       14570 3746         3.89
```

- 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 serial dependence.

Results:

```
> 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)
 18       20149 3746         5.38
 13       14570 3746         3.89
```

- 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 serial dependence.

  High dependence factors (> 5) are worrisome and may be due to influential starting values, high correlations between coefficients, or poor mixing.

The Raftery-Lewis diagnostic will differ depending on which quantile q you choose.

The Raftery-Lewis diagnostic will differ depending on which quantile q you choose.

Estimates tend to be conservative in that it will suggest more iterations than necessary.

The Raftery-Lewis diagnostic will differ depending on which quantile q you choose.

Estimates tend to be conservative in that it will suggest more iterations than necessary.

It only tests marginal convergence on each parameter.

The Raftery-Lewis diagnostic will differ depending on which quantile q you choose.

Estimates tend to be conservative in that it will suggest more iterations than necessary.

It only tests marginal convergence on each parameter.

Nevertheless, it often works well with simple models.

Outline

Heidelberg and Welch Diagnostic

Heidelberg and Welch Diagnostic

The **Heidelberg and Welch diagnostic** calculates a test statistic (based on the Cramer-von Mises test statistic) to accept or reject the null hypothesis that the Markov chain is from a stationary distribution.

Heidelberg and Welch Diagnostic

The **Heidelberg and Welch diagnostic** calculates a test statistic (based on the Cramer-von Mises test statistic) to accept or reject the null hypothesis that the Markov chain is from a stationary distribution.

The diagnostic consists of two parts.

First Part:

First Part:

1. Generate a chain of N iterations and define an α level.

First Part:

1. Generate a chain of N iterations and define an α level.
2. Calculate the test statistic on the whole chain. Accept or reject null hypothesis that the chain is from a stationary distribution.

First Part:

1. Generate a chain of N iterations and define an α level.
2. Calculate the test statistic on the whole chain. Accept or reject null hypothesis that the chain is from a stationary distribution.
3. If null hypothesis is rejected, discard the first 10% of the chain. Calculate the test statistic and accept or reject null.

First Part:

1. Generate a chain of N iterations and define an α level.
2. Calculate the test statistic on the whole chain. Accept or reject null hypothesis that the chain is from a stationary distribution.
3. If null hypothesis is rejected, discard the first 10% of the chain. Calculate the test statistic and accept or reject null.
4. If null hypothesis is rejected, discard the next 10% and calculate the test statistic.

First Part:

1. Generate a chain of N iterations and define an α level.
2. Calculate the test statistic on the whole chain. Accept or reject null hypothesis that the chain is from a stationary distribution.
3. If null hypothesis is rejected, discard the first 10% of the chain. Calculate the test statistic and accept or reject null.
4. If null hypothesis is rejected, discard the next 10% and calculate the test statistic.
5. Repeat until null hypothesis is accepted or 50% of the chain is discarded. If test still rejects null hypothesis, then the chain fails the test and needs to be run longer.

Second Part:

Second Part:

If the chain passes the first part of the diagnostic, then it takes the part of the chain not discarded from the first part to test the second part.

Second Part:

If the chain passes the first part of the diagnostic, then it takes the part of the chain not discarded from the first part to test the second part.

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

Second Part:

If the chain passes the first part of the diagnostic, then it takes the part of the chain not discarded from the first part to test 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 out longer.

Second Part:

If the chain passes the first part of the diagnostic, then it takes the part of the chain not discarded from the first part to test 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 out longer.

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

     Stationarity start     p-value
     test         iteration
var1 passed       1         0.834
var2 passed       1         0.693

     Halfwidth Mean Halfwidth
     test
var1 passed    3.03 0.0756
var2 passed    1.00 0.0805
```