Implementing the Metropolis-Hastings algorithm in R

Jasper Ginn

April 8, 2019

In a previous article (LINK), we derived the conditional posterior distributions for the regression coefficients with normal priors. The posteriors turned out to be conjugate, and thanks to a some mathematical manipulation we were able to derive the posterior mean and variance such that we could sample straight from the posterior (normal) distribution.

Now consider the case where this is not possible. If we do not pick a conjugate prior, then the issue is that we can only derive the posterior distribution up to a proportionality constant. Fortunately, this situation is sufficient to use another MC sampler: the Metropolis-Hastings (MH) algorithm.

Example: estimating the correlation between two variables

The following example is taken from Lynch (2007), pp. 118-120. Suppose that we have two variables $x, y \sim N(0,1)$. We wish to estimate the correlation between these two variables. The conditional posterior distribution that we want to estimate then becomes

$$\begin{aligned} f(\rho|x,y) &\propto \frac{1}{(1-\rho^2)^{(3/2)}} \prod_{i=1}^{n} \frac{1}{2\pi\sqrt{1-\rho^2}} e^{\left(-\frac{1}{2(1-\rho^2)}[x_i^2 - 2\rho x_i y_i + y_i^2]\right)} \\ &\propto \frac{1}{(1-\rho^2)^{(n+3)/2}} e^{\left(-\frac{1}{2(1-\rho^2)}[\sum x_i^2 - 2\rho \sum x_i y_i + \sum y_i^2]\right)} \end{aligned}$$

This posterior density is not of a known form, and hence we cannot sample from it directly. However, we can use $\mathbb{R}$ and MC sampling.

First, we generate a data set under the conditions specified above. That is, we generate two random data sets with mean 0 and standard deviation 1.

```r
# Draw random x and make y a variation on x
x <- scale(rnorm(1200, mean=0, sd=1), center=TRUE, scale=TRUE)[,1]
y <- scale(rnorm(1200, mean=0, sd=1), center=TRUE, scale=TRUE)[,1]
y <- scale(x + runif(1200, -4, 4), center=TRUE, scale=TRUE)[,1]
```

For x and y above, $\text{cor}(x, y) = .65$

Next, we create a function for the posterior distribution. Notice, however, that we log the posterior to prevent overflow and underflow problems in the exponent.

$$\begin{aligned}
\ln f(\rho|x, y) &\propto \ln\left(\frac{1}{(1-\rho^2)^{(n+3)/2}} e^{\left(-\frac{1}{2(1-\rho^2)}\left[\sum x_i^2 - 2\rho\sum x_i y_i + \sum y_i^2\right]\right)}\right) \\
&\propto \ln\left((1-\rho^2)^{-(n+3)/2} e^{\left(-\frac{1}{2(1-\rho^2)}\left[\sum x_i^2 - 2\rho\sum x_i y_i + \sum y_i^2\right]\right)}\right) \\
&\propto -\left(\frac{n-3}{2}\right)\ln\left(1-\rho^2\right) - \frac{1}{2}\frac{\left[\sum x_i^2 - 2\rho\sum x_i y_i + \sum_i^2\right]}{(1-\rho^2)}
\end{aligned}$$

In $\mathbb{R}$, this yields

```
# Posterior
lnposterior <- function(r, n, x, y) {


  return(
    - ((n-3)/2) * log(1-r^2) - (1/2) *
      (sum(x^2)-2*r*sum(x*y) + sum(y^2)) / (1-r^2)
  )


}
```

Next, we set up the number of iterations and pre-allocate the results matrix.

```
# Results matrix
k <- 10000
corr <- matrix(0, k)
# Accepted draws
accepted <- 0
```

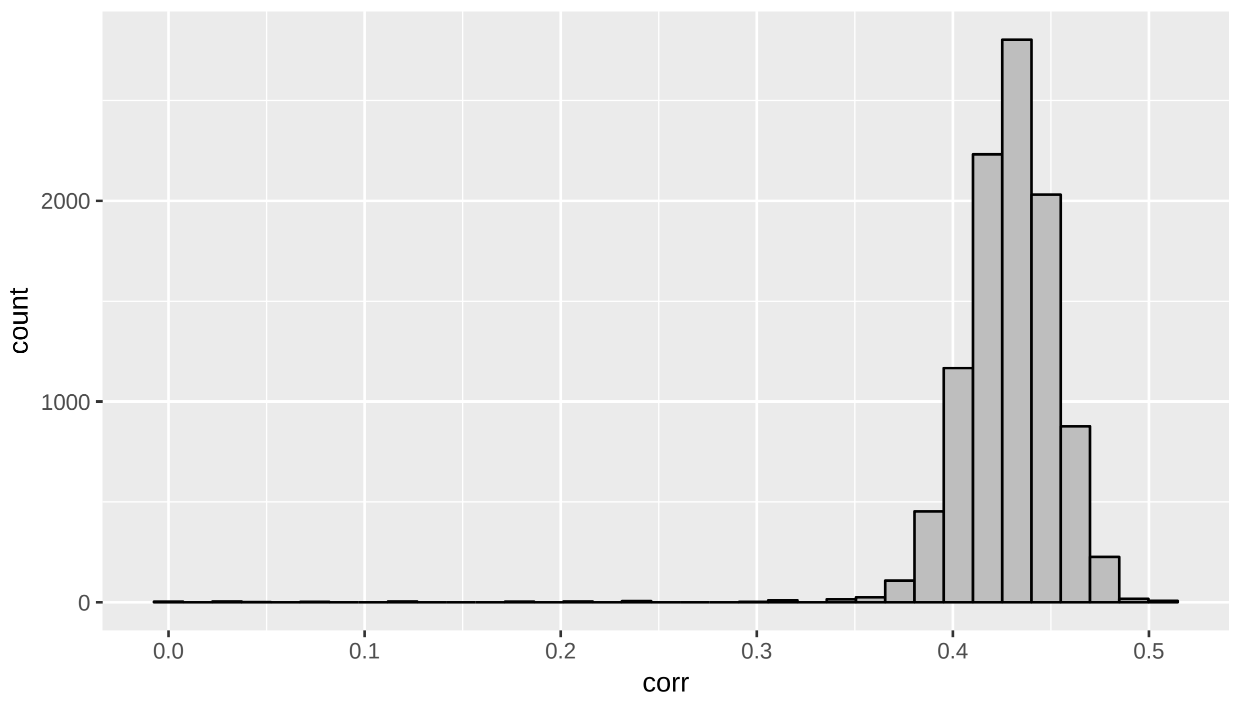
Finally, we loop over the iterations and estimate r

```
# Loop over the k draws
# (note that we start at 2 [==>] this means that the starting value is
# implicitly set to 0)
```

```r
for(i in 2:k) {

  # Proposal distribution
  # Update the previous value with a uniform proposal density
  corr[i] <- corr[i-1] + runif(1, min = -.07, max=.07)

  # If the absolute value of the current draw is larger than one,
  # reject the sample (this is a check to see whether the current draw
  # is bounded by -1 and 1)
  if( abs(corr[i]) > 1 ) {

    corr[i] <- corr[i-1]

  }

  # Evaluate the log-likelihood of the function
  ll_corr_current <- lnposterior(corr[i], length(x), x, y)
  ll_corr_previous <- lnposterior(corr[i-1], length(x), x, y)

  # Draw a uniform and log (since the likelihood is logged also)
  u <- log(runif(1, min=0, max=1))

  # Accept or reject based on the difference between corr_current and
  # corr_previous (why difference? Because we logged the values ==>
  #  log(a/b) = log(a) - log(b))
  ll_diff <- ll_corr_current - ll_corr_previous

  # Reject the sample if ll_diff < u
  if( ll_diff < u ) {

    # Set current value to previous value
    corr[i] <- corr[i-1]
    # Continue
    next

  }

  # Set accepted + 1
  accepted <- accepted + 1

  # Every 100 draws, print progress to console
  if ( (i %% 100) == 0 ) {
    #print(c(i,corr[i],accepted/i))
  }

}
```

This gives us

```
library(ggplot2)
ggplot(data.frame(corr=corr), aes(x=corr)) +
  geom_histogram(color="black", fill="grey", bins=35)
```



Or an estimate of 0.429 with an acceptance rate of 0.46.

Implementing a simple Gibbs sampler

Our next objective is to implement a Gibbs sampler for simple linear regression. The next step is then to replace the Gibbs sampler we use for one of the coefficients with a Metropolis-Hastings sampler. The model is given by:

$$\vec{y} = \mathbf{X}\vec{w} + \vec{\epsilon}, \quad \vec{\epsilon} \sim N(0, \sigma^2)$$

In total, we estimate three parameters:

1. Two regression coefficients (β_0 and β_1). Both are assumed to be normally distributed with some mean and standard deviation.
2. One parameter for the residual (unexplained) variance.

We derived the conditional posteriors in an earlier document (LINK). Here, we just list their implementation in $\mathbb{R}$

```r
# Posterior mean helper function
gibbs_posterior_mu <- function(X, xj, y, w, sigma, mu0, tau0) {

  # Numerator
  num1 <- sum(xj * (y - X %*% w)) / sigma
  num2 <- mu0 / tau0

  # Denominator
  den1 <- t(xj) %*% xj / sigma
  den2 <- 1 / tau0

  # Return
  return( ( num1 + num2 ) / ( den1 + den2 ) )

}

# Posterior tau helper function
gibbs_posterior_tau <- function(xj, sigma, tau0) {

  return( 1 / ( (t(xj) %*% xj/sigma) + (1/tau0) ) )

}

# Posterior rate for IG gibbs
gibbs_posterior_rate <- function(N, prior_rate) {

  return( (N / 2) + prior_rate )

}

# Posterior scale for IG gibbs
gibbs_posterior_scale <- function(X, y, w, prior_scale) {
  return(
    (sum((y - X %*% w)^2) / 2) + prior_scale
  )

}
```

We set the following (uninformative priors)

```
# Priors
pr_mu_b0 <- 0
pr_tau_b0 <- 1000
pr_mu_b1 <- 0
pr_tau_b1 <- 1000
pr_rate_sig <- 0.001
pr_scale_sig <- 0.001
```

The data are generated as follows

```
## Plan: model b[0] + b[1]*x + e

# Draw x
x <- rnorm(500, mean=0, sd=3)
# Coef (intercept)
b0 <- 4.2
b1 <- 1.87
# Generate y
res_variance <- 2.1^2
y <- rnorm(500, mean = (b0 + b1*x), sd = sqrt(res_variance))

# Create data frame for data
dat <- data.frame(x=x, y=y)

# Create a model matrix
```

```r
X <- model.matrix(y ~ x, data=dat)
N <- nrow(X)
```

Next, we specify the initial values and initialize the result matrix

```r
# Number of replications
k <- 20000
# Results
posterior_sampled <- matrix(0, nrow=k, ncol=3)

# First row of the posterior_sampled matrix are the initial values
posterior_sampled[1,] <- c(2, -2, 0.01)
```

We are now ready to sample the posterior

```r
# Iterate
for( i in 2:k ) {

  # Retrieve w (coefficients)
  # Ignore the last value ==> this is for sigma
  w <- posterior_sampled[i-1, -3]
  sigma <- posterior_sampled[i-1, 3]

  # For each parameter, retrieve posterior samples using gibbs only

  #### b[0]

  w[1] <- rnorm(1,
              mean = gibbs_posterior_mu(matrix(X[,-1], ncol=1), matrix(X[,1], ncol=1),
                                         y, w[-1], sigma, pr_mu_b0, pr_tau_b0),
              sd = sqrt(gibbs_posterior_tau(X[,1], sigma, pr_tau_b0)))

  #### b[1]
  w[2] <- rnorm(1,
                mean = gibbs_posterior_mu(matrix(X[,-2], ncol=1), matrix(X[,2], ncol=1),
                                          y, w[-2], sigma, pr_mu_b0, pr_tau_b0),
                sd = sqrt(gibbs_posterior_tau(X[,2], sigma, pr_tau_b0)))

  #### sigma

  sigma <- 1 / rgamma(1,
                      gibbs_posterior_rate(N, pr_rate_sig),
                      gibbs_posterior_scale(X, y, w, pr_scale_sig))

  # Update results matrix
  posterior_sampled[i,] <- c(w, sigma)

}
```

Finally, we square-root the residual variance (to obtain an estimate of the standard deviation) and obtain the MAP values

```
# square-root the posterior sigma
posterior_sampled[,3] <- sqrt(posterior_sampled[,3])

# MAP values
apply(posterior_sampled, 2, mean)
```

```
## [1] 4.107319 1.857211 2.214792
```

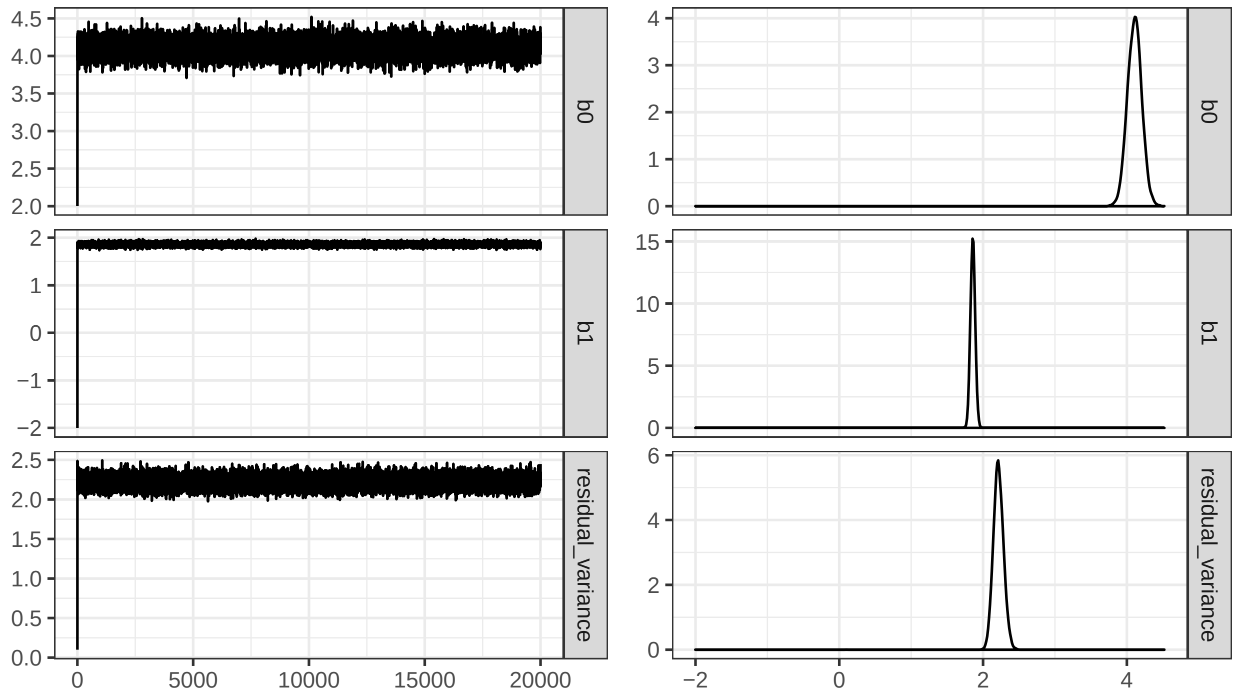
These values are close to the maximum likelihood estimates

```
c(coef(lm("y ~ x", data=dat)), sd(resid(lm("y~x", data=dat))))
```

```
## (Intercept)           x
##    4.107710    1.857418    2.208903
```

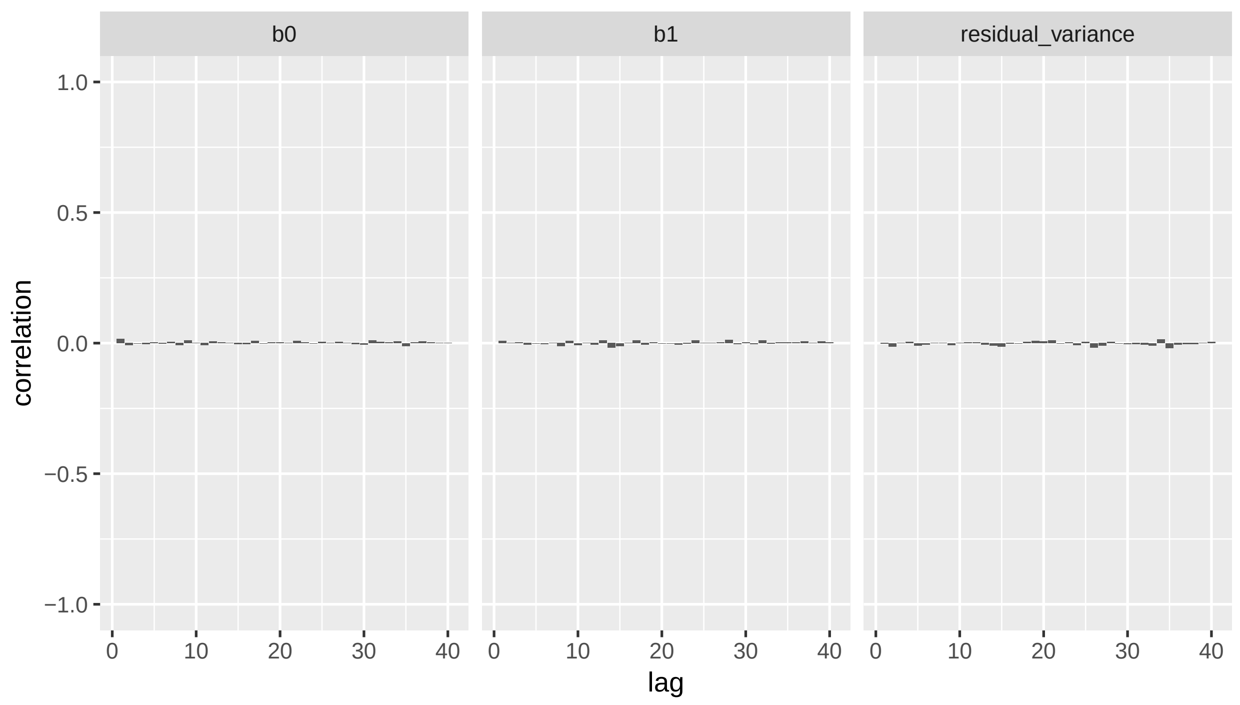
The trace plot and the density plots show us that the parameters have likely converged to their stationary distributions

```r
library(grid)
library(gridExtra)
p1 <- trace_plot(posterior_sampled, k)
p2 <- density_plot(posterior_sampled)
grid.arrange(p1, p2, ncol=2)
```



Autocorrelation is not an issue for these data and model.

```r
autocor_plot(posterior_sampled)
```



Mixing MH and Gibbs samplers

In this final step, we will sample β_0 using the MH algorithm while sampling the other parameters using the Gibbs sampler.

To sample the intercept using the MH algorithm, we need the density of the posterior distribution up to a proportionality constant. We can find the equation we need from the document (LINK) on deriving conditional posterior distributions. (equation 4.2.5 on page 8).

$$\begin{aligned}
f(\beta_j| \dots) &\propto e^{\left[-\frac{\sum_i (\beta_j x_{ji})^2}{2\sigma^2} + \beta_j x_{ji} \frac{\sum_i (y_i - \beta_0 - \left[\sum_{k=1}^{j-1} \beta_k x_{ki}\right])}{\sigma^2}\right]} \times e^{\left[-\frac{\beta_j^2}{2\tau_{j,0}^2} + \frac{\beta_j \mu_{j,0}}{\tau_{j,0}^2}\right]} \\
&\propto e^{\left[\left\{-\frac{\sum_i (\beta_j)^2 (x_{ji})^2}{2\sigma^2} - \frac{\beta_j^2}{2\tau_{j,0}^2}\right\} + \left\{\beta_j x_{ji} \frac{\sum_i (y_i - \beta_0 - \left[\sum_{k=1}^{j-1} \beta_k x_{ki}\right])}{\sigma^2} + \frac{\beta_j \mu_{j,0}}{\tau_{j,0}^2}\right\}\right]} \\
&\propto e^{\left[-\beta_j^2 \left\{\frac{\sum_i x_{ji}^2}{2\sigma^2} + \frac{1}{2\tau_{j,0}^2}\right\} + \beta_j \left\{\frac{x_{ij} \sum_i (y_i - \beta_0 - \left[\sum_{k=1}^{j-1} \beta_k x_{ki}\right])}{\sigma^2} + \frac{u_{j,0}}{\tau_{j,0}^2}\right\}\right]}
\end{aligned} \tag{4.2.5}$$

If we vectorize this result and generalize it so it holds for all j[1], we get the following equation:

[1] In its current form, it holds only for $j = \max(j)$. See p.9 in the document on deriving conditional posterior distributions.

$$
\begin{aligned}
f(\beta_j|\dots) &\propto e^{\left[-\beta_j^2\left\{\frac{\vec{x_j}^T\cdot\vec{x_j}}{2\sigma^2}+\frac{1}{2\tau_{j,0}^2}\right\}+\beta_j\left\{\frac{\sum\vec{x_j}\circ(\vec{y}-\mathbf{X}^{-j}\vec{w}_{-j})}{\sigma^2}+\frac{\mu_{j,0}}{\tau_{j,0}^2}\right\}\right]} \\
&= e^{[A+B]}
\end{aligned}
$$

Where:

- $\vec{x_j}^T\vec{x_j}$ is the sum of the squared elements of $\vec{x_j}$
- X^{-j} is the design matrix without the j^{th} column corresponding to β_j
- w_{-j} is the weight matrix without the j^{th} coefficient

Instead of evaluating the posterior in this form, we take the logarithm of $f(\beta_j|\dots)$ to avoid computational issues. Hence, the posterior becomes:

$$
\begin{aligned}
\ln\left[f(\beta_j|\dots)\right] &\propto \ln\left(e^{A+B}\right) \\
&\propto \ln\left(e^{A}\right)\ln\left(e^{B}\right) \\
&\propto A+B \\
&\propto -\beta_j^2\left\{\frac{\vec{x_j}^T\cdot\vec{x_j}}{2\sigma^2}+\frac{1}{2\tau_{j,0}^2}\right\}+\beta_j\left\{\frac{\sum\vec{x_j}\circ(\vec{y}-\mathbf{X}^{-j}\vec{w}_{-j})}{\sigma^2}+\frac{\mu_{j,0}}{\tau_{j,0}^2}\right\}
\end{aligned}
$$

In $\mathbb{R}$, this yields:

```r
# Posterior density
# i.e. the posterior density proportional to a normal up to a constant
# It will be logged to simplify working with the exponents
MH_posterior_coef <- function(bj, xj, X, y, w, sigma, prior_mu, prior_tau) {

  ## Left side
  left <- -(bj^2) * ( ((t(xj) %*% xj)/(2*sigma)) + (1 / (2*prior_tau)))

  ## Right side
  right <- bj * ( ( (sum(xj * (y - X %*% w))) / (sigma) ) + (prior_mu / prior_tau) )

  ## Add left and right
  return(left + right)
}
```

The Metropolis-Hastings algorithm proceeds as follows:

Metropolis-Hastings Algorithm

1. Select an initial value $\theta^{(0)}$.
2. For $i = 1, \ldots, m$, repeat:

   a. draw a candidate value from a proposal distribution $q(\theta)$ s.t. $\theta^* \sim q(\theta^*|\theta^{(i-1)})$.

   b. Set $\alpha = \frac{g(\theta^*)/q(\theta^*|\theta^{(i-1)})}{g(\theta^{(i-1)})/q(\theta^*|\theta^{(i-1)})} = \frac{g(\theta^*)q(\theta^{(i-1)}|\theta^*)}{g(\theta^{(i-1)})q(\theta^*|\theta^{(i-1)})}$

   c. If $\alpha \geq 1$: Accept θ^* and set $\theta^{(i)} \leftarrow \theta^*$

   d. If $\alpha < 1$: Accept θ^* and set $\theta^{(i)} \leftarrow \theta^*$ with probability α

We will use a random-walk MH algorithm. In a random walk MH, the proposal distribution is centered on the value of theta of the previous iteration ($\theta^{(i-1)}$). If we choose $q(\theta)$ to be a symmetric distribution (e.g. a normal), we have a neat additional feature; it means that in step $2b$, we only have to calculate the ratio $\alpha = \frac{g(\theta^*)}{g(\theta^{(i-1)})}$. This works because, for symmetric distributions, $q(\theta^{(i-1)}|\theta^*) = q(\theta^*|\theta^{(i-1)})$.

The MH algorithm has an **acceptance rate** which is basically the proportion of accepted candidates. High acceptance rates means that we are stepping through the posterior space too slow (slow mixing). Typically, we want an acceptance rate between $20\% \sim 50\%$. Similarly, we want low autocorrelation, but in practice the autocorrelation will be quite high because samples generated from an MH are not i.i.d. (in this case we can always use longer chains).

In our case, the acceptance rate is controlled by a parameter ζ. This parameter is the standard deviation we will use for the normal proposal density in the MH algorithm. **The lower the value of this parameter, the higher the acceptance rate will be**.

The MH algorithm as described above is implented as follows (this is for a single iteration):

```
# One step of MH algorithm
MH_one_step <- function(b_previous, xj, X, y, w, sigma, prior_mu, prior_tau, zeta) {

  ## Proposal density
  b_proposed <- rnorm(1, mean=b_previous, sd=zeta)
  ## Draw from the posterior
  g_proposed <- MH_posterior_coef(b_proposed, xj, X, y, w, sigma, prior_mu, prior_tau)
  g_previous <- MH_posterior_coef(b_previous, xj, X, y, w, sigma, prior_mu, prior_tau)

  ## Difference (because logged)
  g_diff <- g_proposed - g_previous

  ## Draw random variate from uniform
  u <- log(runif(1, min=0, max=1))

  ## If gdiff < u ==> set b_current to b_previous
  if( g_diff < u ) {

    b_current <- b_previous

  } else { # Accepted the proposed value

    b_current <- b_proposed

  }

  # Return b
  return(b_current)

}
```

Next, we set up the results matrix, the tuning parameter ζ and a scalar to keep track of the number of accepted draws.

```
# We are not resetting results matrix etc. but making a new results matrix
posterior_sampled_mh <- matrix(0, nrow=k, ncol=3)
posterior_sampled_mh[1,] <- posterior_sampled[1,]
```

```r
# Keep track of accepted draws for MH
accepted <- 0

# Tuning parameter for proposal density
zeta <- 0.25
```

Finally, we iterate over the desired number of iterations and obtain samples for the posterior distribution.

```r
## Iterate
for( i in 2:k ) {

  ## Retrieve w (coefficients)
  # Ignore the last value ==> this is for sigma
  w <- posterior_sampled_mh[i-1, -3]
  sigma <- posterior_sampled_mh[i-1, 3]

  ## For each parameter, retrieve posterior samples using a mix of gibbs / MH only

  #### b[0] ==> Metropolis-Hastings

  b0_prev <- w[1]
  w[1] <- MH_one_step(w[1], matrix(X[,1], ncol=1), matrix(X[,-1], ncol=1),
                      y, w[-1], sigma, pr_mu_b0, pr_tau_b0, zeta)

  # If accepted, increment
  if(w[1] != b0_prev) {
    accepted <- accepted+1
  }

  #### b[1] ==> Gibbs

  w[2] <- rnorm(1,
                mean = gibbs_posterior_mu(matrix(X[,-2], ncol=1), matrix(X[,2], ncol=1),
                                          y, w[-2], sigma, pr_mu_b0, pr_tau_b0),
                sd = sqrt(gibbs_posterior_tau(X[,2], sigma, pr_tau_b0)))

  #### sigma ==> Gibbs

  sigma <- 1 / rgamma(1,
                      gibbs_posterior_rate(N, pr_rate_sig),
                      gibbs_posterior_scale(X, y, w, pr_scale_sig))

  # Update results matrix
  posterior_sampled_mh[i,] <- c(w, sigma)

}

# square-root the posterior sigma
posterior_sampled_mh[,3] <- sqrt(posterior_sampled_mh[,3])
```

Given 0.25, the acceptance rate is 0.429

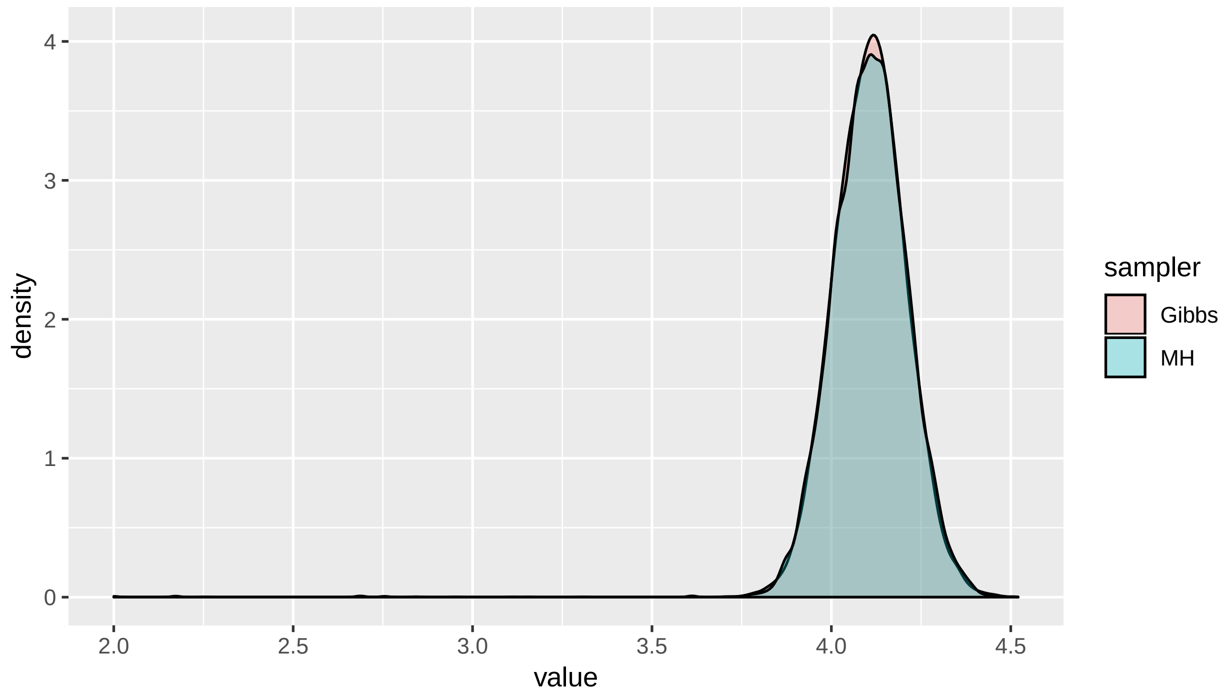
When we compare the estimates of the above procedure with the all-Gibbs procedure, we get:

```
# MAP values
maps <- matrix(0, nrow=3, ncol=3,
               dimnames = list(
                 c("Gibbs", "Mixed", "ML"),
                 c("b[0]", "b[1]", "Residual_variance")
               ))
maps[1,] <- round(apply(posterior_sampled_mh, 2, mean), digits=3)
maps[2,] <- round(apply(posterior_sampled, 2, mean), digits=3)
maps[3,] <- round(c(unname(coef(lm("y~.", data=dat))),
                    sd(resid(lm("y~.", data=dat)))), digits=3)

knitr::kable(maps)
```

	b[0]	b[1]	Residual_variance
Gibbs	4.107	1.857	2.215
Mixed	4.107	1.857	2.215
ML	4.108	1.857	2.209

These estimates are more or less equivalent. When we compare the posterior densities of β_0 using different sampling methods, we see:

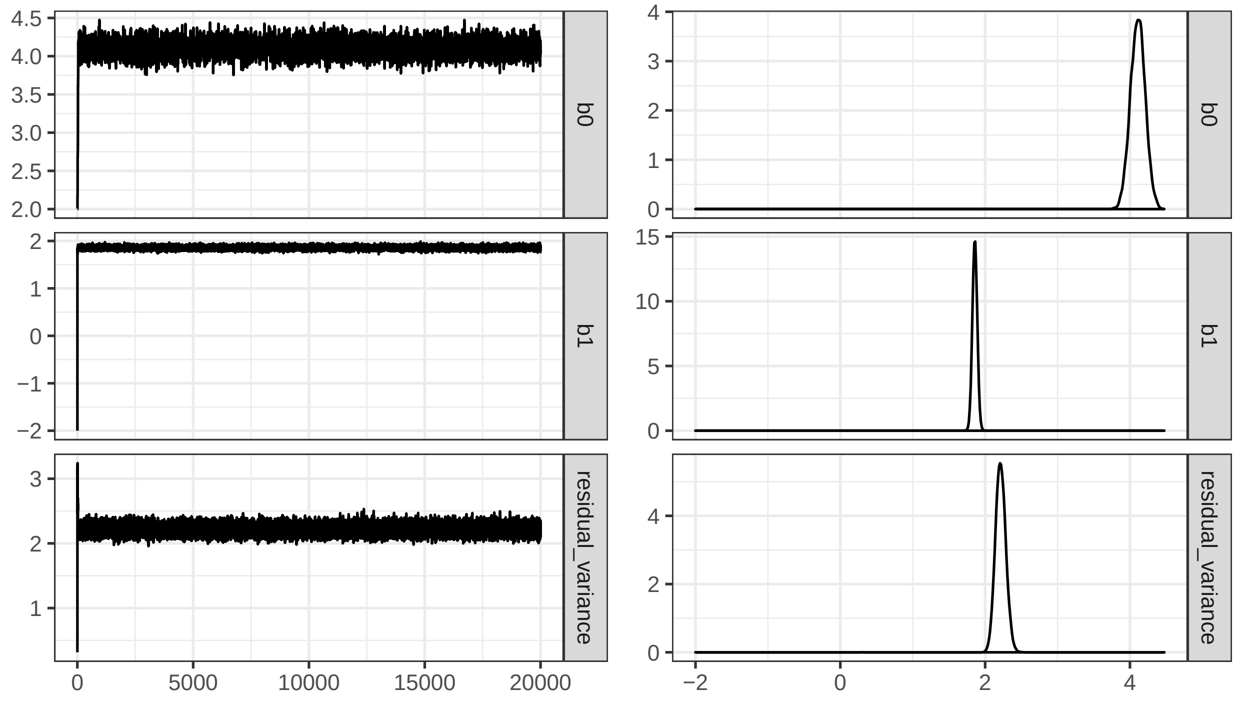
```r
# Compare densities
data_frame(
  "Gibbs" = posterior_sampled[,1],
  "MH" = posterior_sampled_mh[,1]
) %>%
  gather(sampler, value) %>%
  ggplot(., aes(x=value, fill=sampler)) +
    geom_density(alpha=0.3)
```



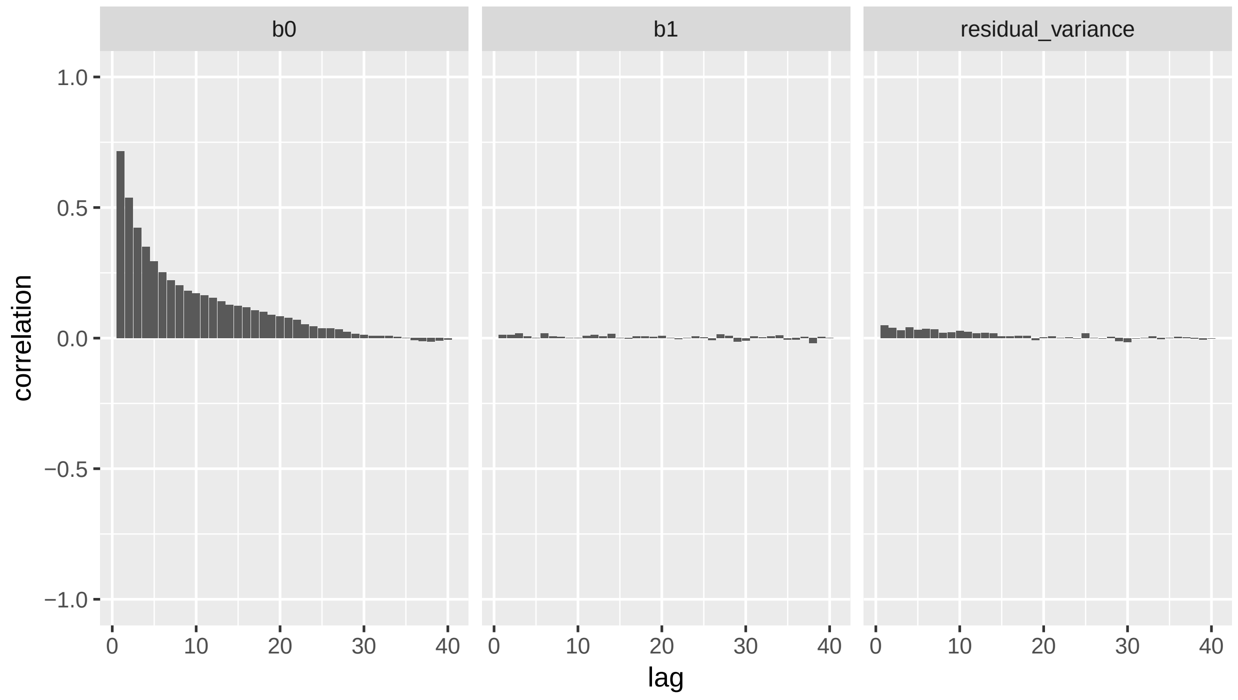
Again, these are very similar. This is not strange or unexpected since we are approximating a normal distribution using a normal proposal density.

We expect the MH sampler to exhibit higher levels of autocorrelation.

```r
library(grid)

library(gridExtra)

p1 <- trace_plot(posterior_sampled_mh, k)

p2 <- density_plot(posterior_sampled_mh)

grid.arrange(p1, p2, ncol=2)
```

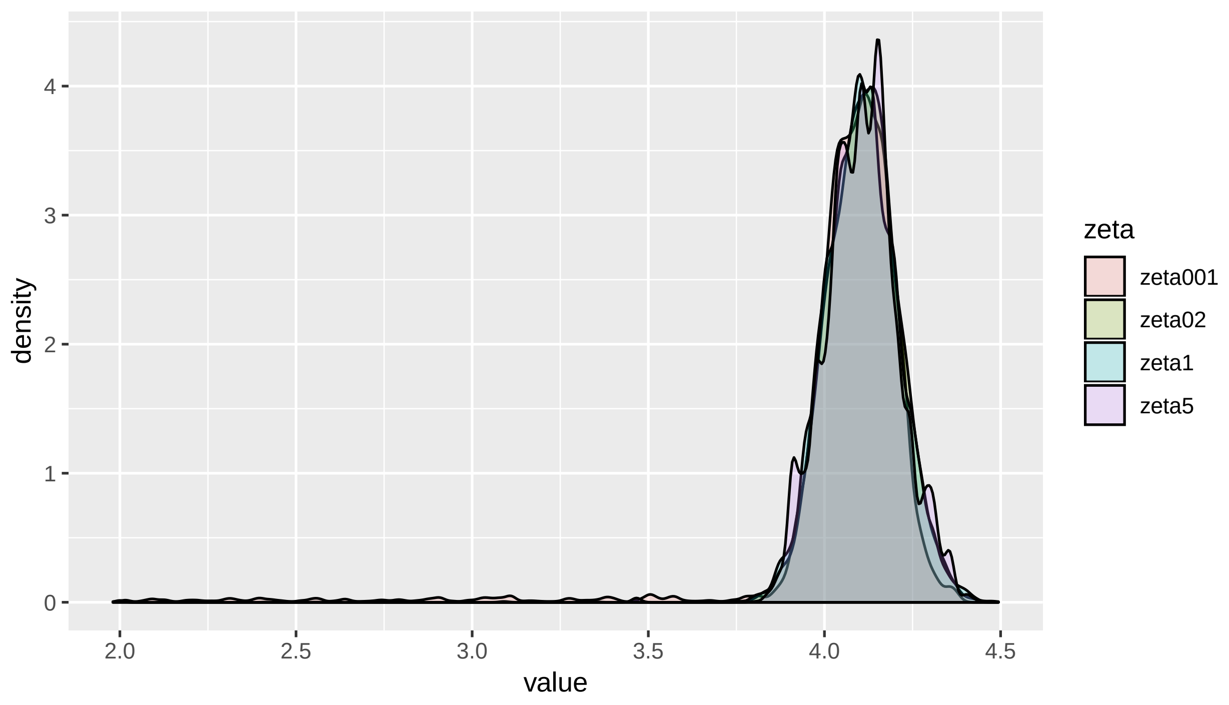


```r
autocor_plot(posterior_sampled_mh)
```



Finally, we can examine the effect of changing the tuning parameter ζ (i.e. the variance for the proposal density).

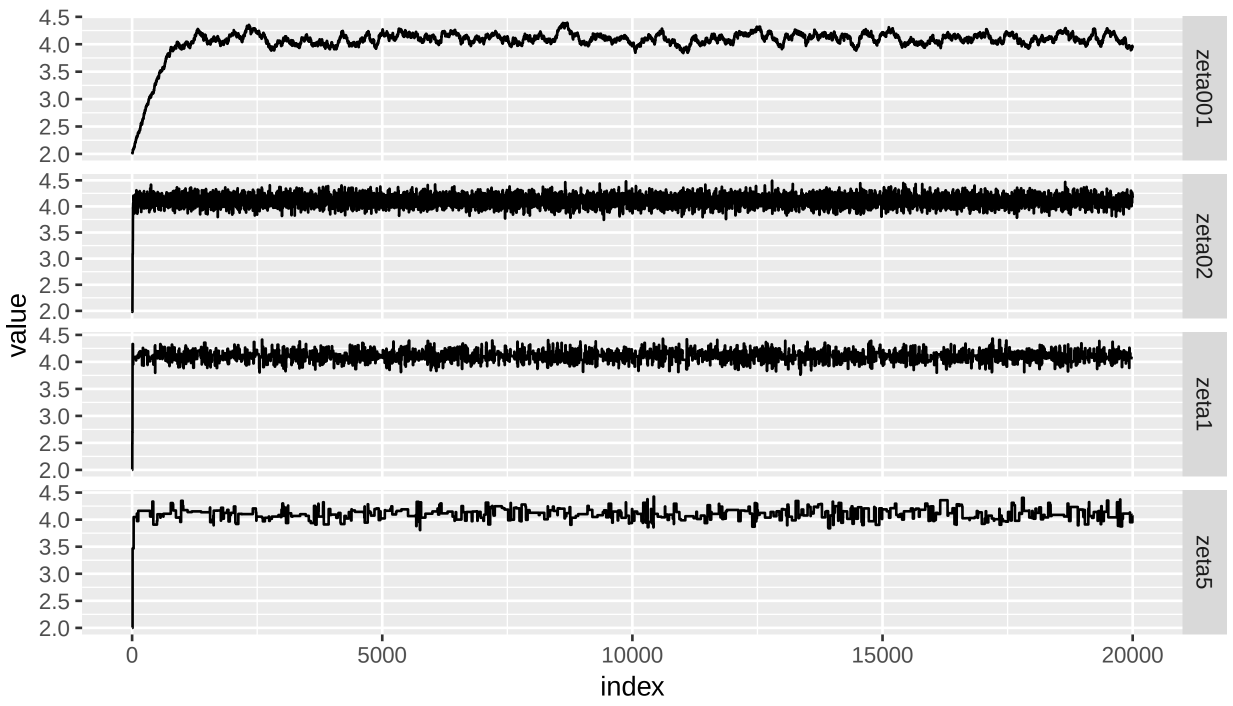
```r
zvals <- data.frame(
  "zeta5" = pz5,
  "zeta1" = pz1,
  "zeta02" = pz02,
  "zeta001" = pz001
) %>%
  gather(zeta, value)

ggplot(zvals, aes(x=value, fill=zeta)) +
  geom_density(alpha=0.2)
```



We observe that, for high or small values of ζ, the distribution does not resemble a normal distribution. Rather, it looks ‘jagged’. The best fit is at $\zeta = .2$.

The same is indicated by the trace plots below.

```
zvals <- data.frame(
  "zeta5" = pz5,
  "zeta1" = pz1,
  "zeta02" = pz02,
  "zeta001" = pz001
) %>%
  mutate(index = 1:length(pz5)) %>%
  gather(zeta, value, -index)

ggplot(zvals, aes(x=index, y=value, group=zeta)) +
  geom_line() +
  facet_grid("zeta ~ .", scales="free_y")
```



References

Lynch, S. M. (2007). Introduction to applied Bayesian statistics and estimation for social scientists. Springer Science & Business Media.