Metropolis-Hastings

Jonathan Navarrete

July 1, 2017

General Metropolis-Hastings

Given a target density f, we build a Markov kernel K with stationary distribution f and then generate a Markov chain X_t using this kernel so that the limiting distribution of X_t is f and integrals can be approximated according to the Ergodic Theorem.

The **Metropolis-Hastings algorithm** is a general purpose MCMC method for approximating a f. Given the target density f and a conditional density $q(y|x)$ that is easy to simulate from. In addition, q can be almost arbitrary in that the only theoretical requirements are that the ration $\frac{f(y)}{q(y|x)}$ is known up to a constant *independent* of x and that $q(\cdot|x)$ has enough dispertion to lead to an exploration of the entire support of f

We can rely on the feature of Metropolis-Hatings algorithm that for every given q, we can then construct a Metropolis-Hastings kernel such that f is its stationary distribution.

The Metropolis-Hastings algorithm as described Robert & Casella goes as follows Given $x^{(t)}$

1. Generate $Y_t \sim q(y|x_t)$
2. Take
$$X_{t+1} = \begin{cases} Y_t & \text{with probability } \rho(x^{(t)}, Y_t) \\ x^{(t)} & \text{with probability } 1 - \rho(x^{(t)}, Y_t) \end{cases}$$
where
$$\rho(x^{(t)}, Y_t) = \min\{\frac{f(Y_t)}{f(x^{(t)})}\frac{q(x^{(t)}|Y_t)}{q(Y_t|x^{(t)})}, 1\}$$

In simpler terms, as we want to generate $X \sim f$, we first take an initial value $x^{(0)}$ (which can almost be any artibrary value in the support of f).

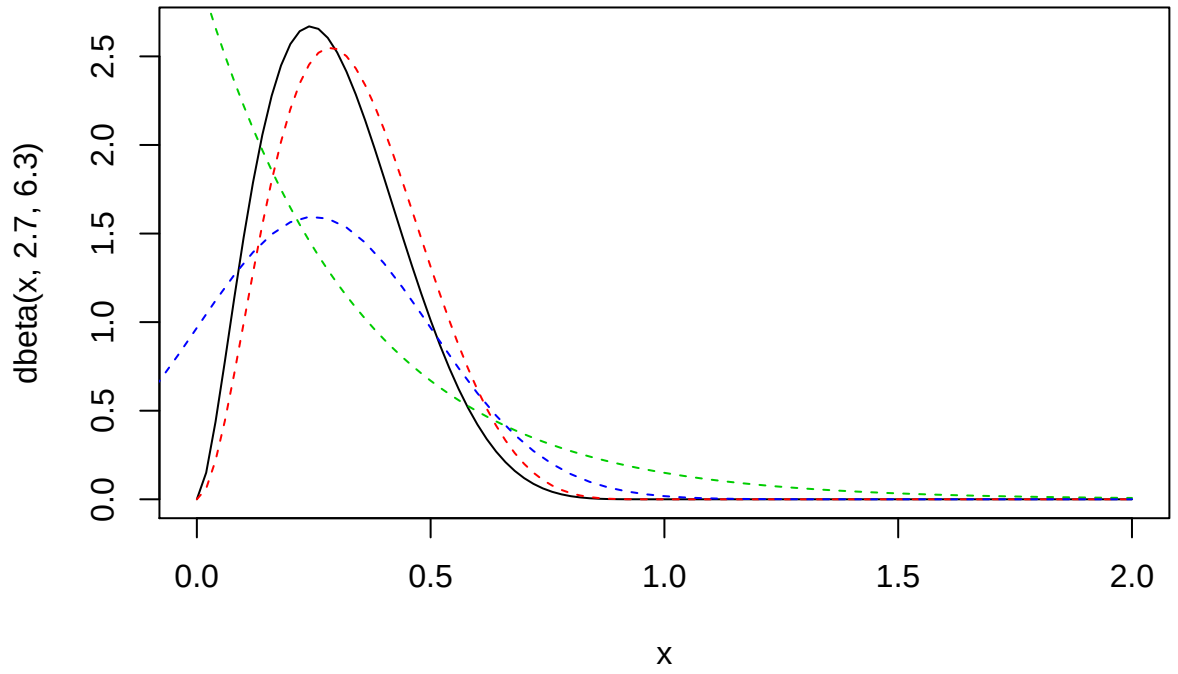
1. We generate a value $Y_0 \sim q(y|x^{(0)})$.
2. We calculate $\rho(x^{(t)}, Y_t)$
3. Generate a random value $U \sim Unif(0, 1)$
4. If $U < \rho(x^{(t)}, Y_t)$, then we accept $X^{(1)} = Y_t$; else we take $X^{(1)} = X^{(0)}$
5. Repeate steps 1-4 until you've satisfied the number of samples needed

Example 6.1: Beta(2.7, 6.3)

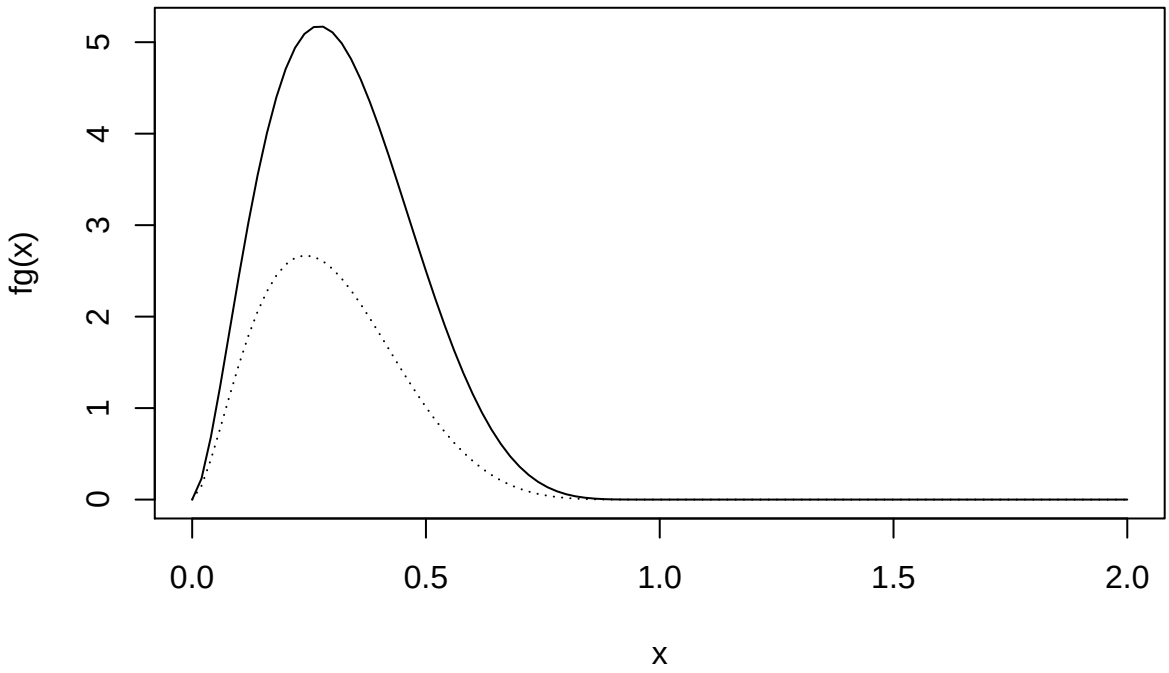
As of now, we've covered multiple ways of generating random samples from a target density. Let us compare the accept-reject algorithm once more with the Metropolis-Hastings algorithm. Generate N samples from $Beta(2.7, 6.3)$

In order to use the accept-reject algorithm, we need a candidate distribution to sample from. Below are a set of potential candidate distributions.

```
## potential instumential distributions
curve(expr = dbeta(x, 2.7, 6.3), from = 0, to = 2)
curve(expr = dbeta(x, 3, 6), from = 0, to = 2, add = TRUE, col = 2, lty = 2)
curve(expr = dexp(x, rate = 3), from = 0, to = 2, add = TRUE, col = 3, lty = 2)
#curve(expr = dnorm(x, mean = 1, sd = 1), from = 0, to = 2, add = TRUE, col = 4, lty = 2)
curve(expr = dnorm(x, mean = 0.25, 0.25), from = -2, to = 2, add = TRUE, col = 4, lty = 2)
```

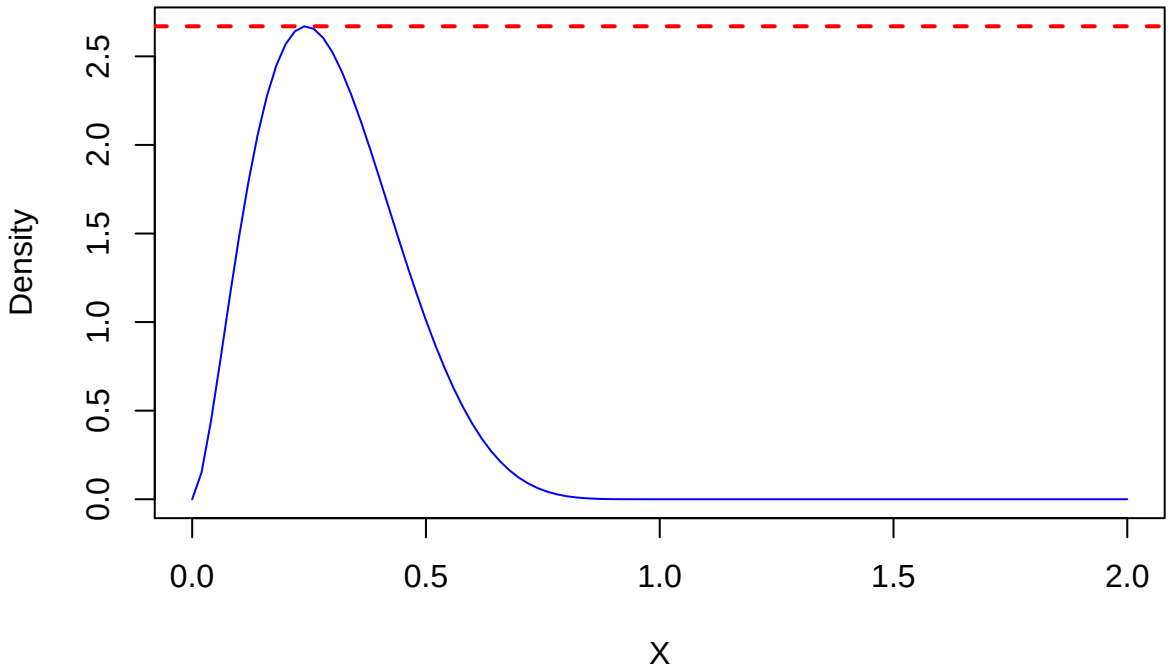



```
fg = function(x){
  1.5*dbeta(x, 2.7, 6.3)/dexp(x, rate = 1)
}

curve(expr = fg, from = 0, to = 2)
curve(expr = dbeta(x, 2.7, 6.3), from = 0, to = 2, add = TRUE, lty = 3)
```



We will use $Exp(3)$ as our candidate distribution, g.

```
N = 10000

## For accept-reject, we need to find a value for M
## we can use `optimize` to find the maximum of our target density
maximum  = optimize(f = function(x){ dbeta(x, 2.7, 6.3) },
                    interval = c(0, 2), maximum = TRUE ) ## obtain maximum

## M here is basically random guess
M = maximum$objective  ## something smaller like 1.5 works too!
curve(expr = dbeta(x, 2.7, 6.3),
      from = 0, to = 2, col = "blue",
      main = "Beta(2.7, 6.3)", xlab = "X", ylab = "Density")
abline(h = M, lty = 2, lwd = 2, col = "red")
```




```
f = function(x){
  dbeta(x, 2.7, 6.3)
}

g = function(x){
  #dnorm(x, mean = maximum$maximum, sd = 0.25)
  dexp(x, rate = 1)
}


#N = 100000
X = numeric(N)
i = 0
while(i < N){
  #Y = abs(rnorm(n = 1, mean = maximum$maximum, sd = 0.25))
  Y = rexp(n = 1, rate = 1)
  U = runif(1)
  if(U*M <= f(Y)/g(Y)){
    i = i + 1
    X[i] = Y
  }
}



print("sample mean from direct samples")
```

```
## [1] "sample mean from direct samples"
```

```r
mean(rbeta(n = 1000, shape1 = 2.7, shape2 = 6.3))
```

```
## [1] 0.3056742
```

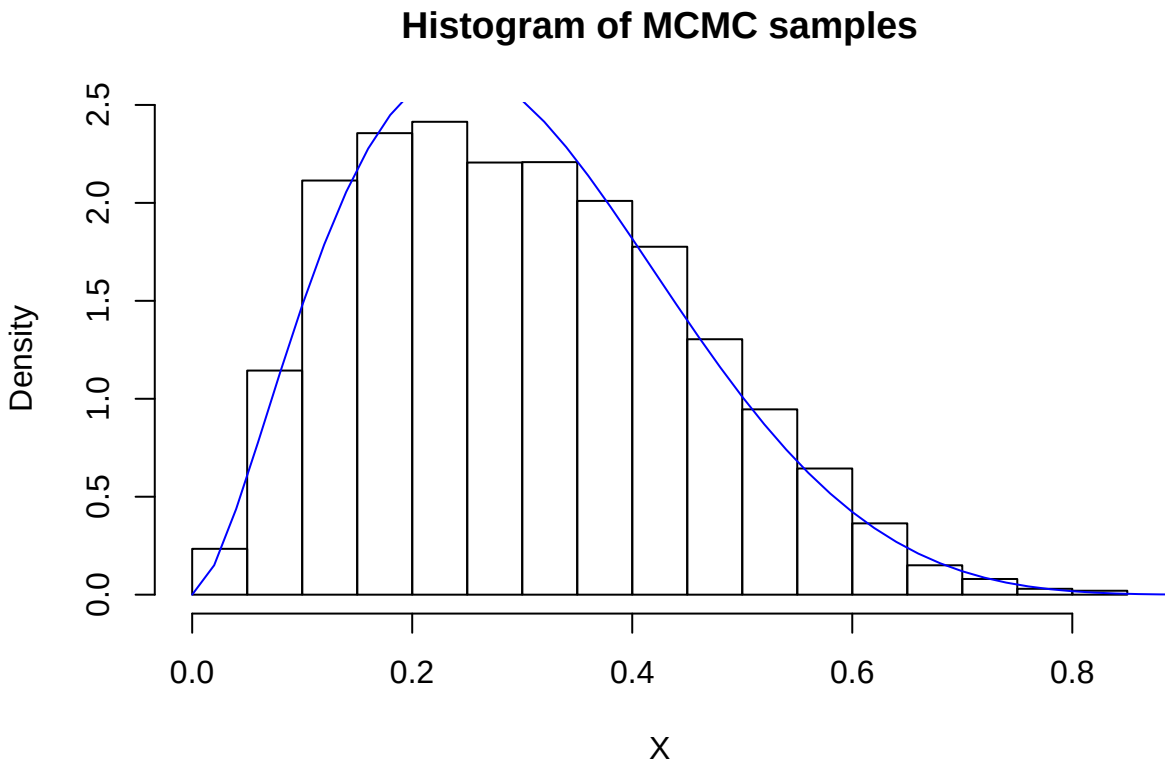
```r
print("sample mean from Accept-Reject samples")
```

```
## [1] "sample mean from Accept-Reject samples"
```

```r
mean(X)
```

```
## [1] 0.3028933
```

```r
## see how samples from chain compare to Beta(2.7, 6.3) density
hist(X, main = "Histogram of MCMC samples", prob = TRUE)
curve(expr = dbeta(x, 2.7, 6.3),
      from = 0, to = 2, add = TRUE, col = "blue")
```



Below is the Metropolis-Hastings implementation for this problem.

```r
## Metropolis Hastings
## now compare results with Beta(2.7, 6.3)

N = 10000
X = numeric(N)
X[1] = rbeta(n = 1, shape1 = 2.7, shape2 = 6.3) ## initial value
for(i in 1:N){
  Y = rexp(n = 1, rate = X[i])
  rho = (dbeta(x = Y, 2.7, 6.3) * dexp(x = X[i], rate = Y)) /
    (dbeta(x = X[i], 2.7, 6.3) * dexp(x = Y, rate = X[i]))

  if(runif(1) < rho){
    X[i+1] = Y
```

```
  } else{
    X[i+1] = X[i]
  }

}

print("sample mean from direct samples")
```

```
## [1] "sample mean from direct samples"
```

```
mean(rbeta(n = 1000, shape1 = 2.7, shape2 = 6.3))
```

```
## [1] 0.2961242
```

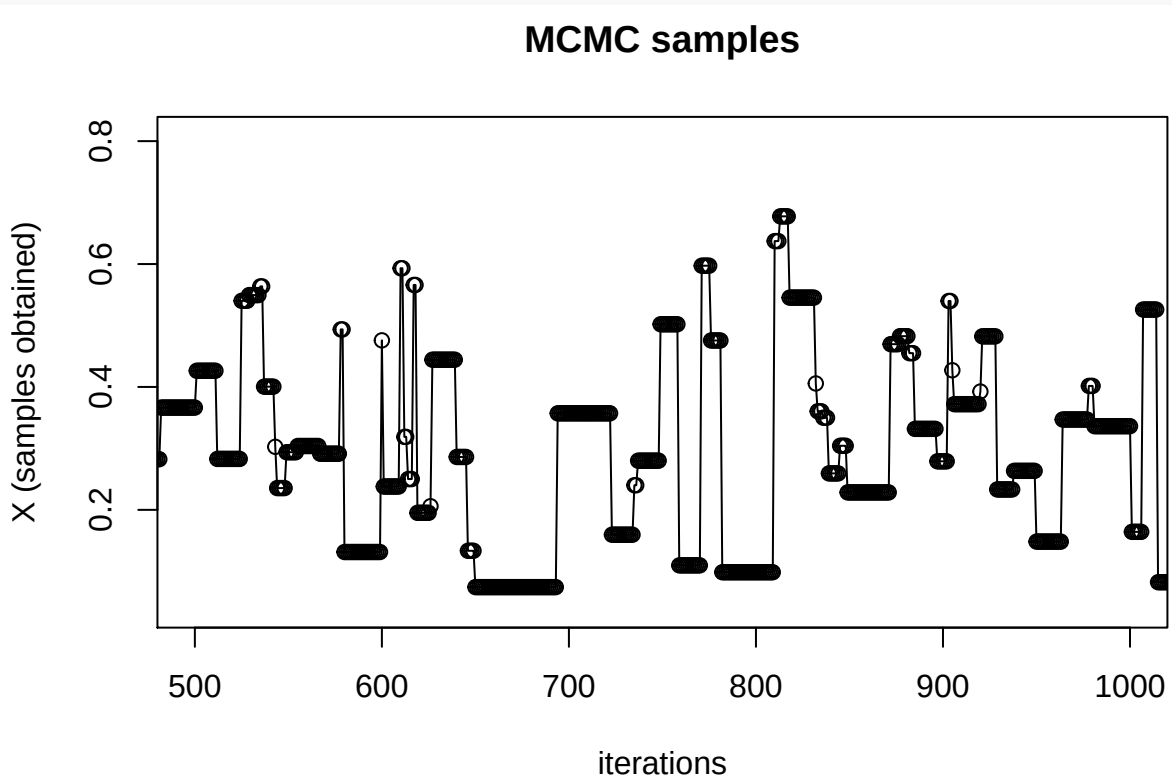
```
print("sample mean from M-H samples")
```

```
## [1] "sample mean from M-H samples"
```

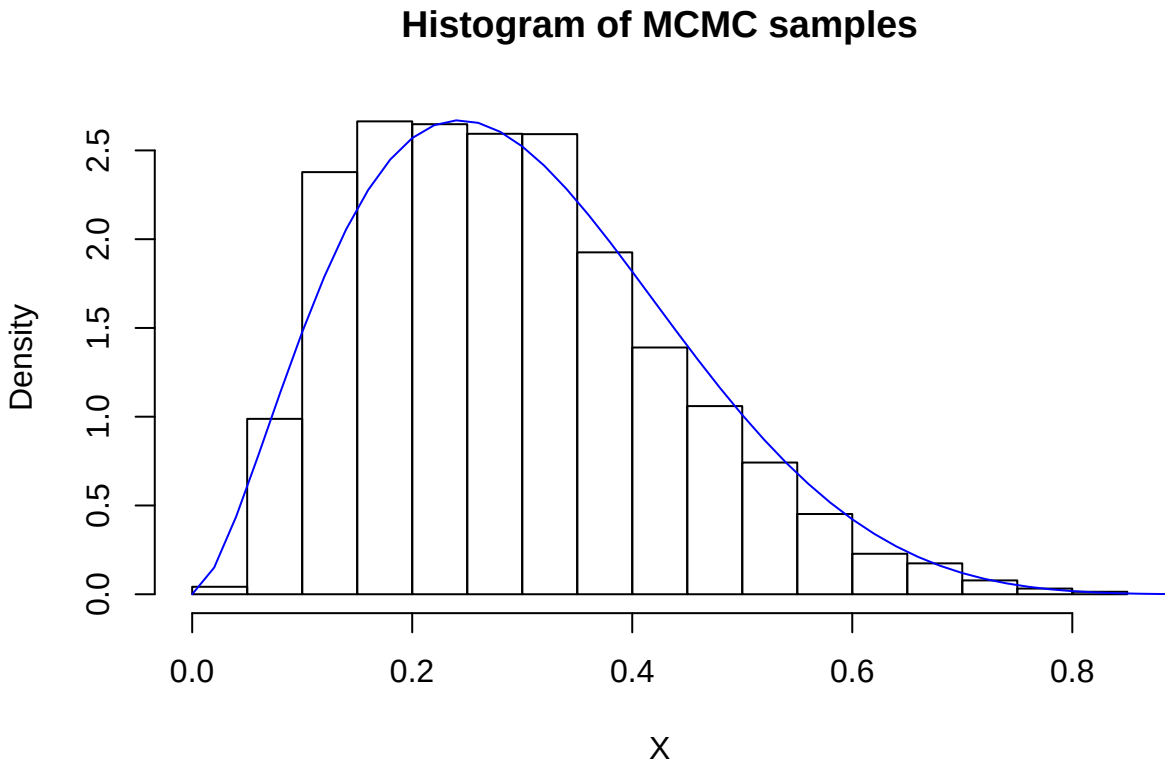
```
mean(X)
```

```
## [1] 0.289545
```

```
## see chain transitions
plot(X, type = "o", main = "MCMC samples",
     xlim = c(500,1000),
     xlab = "iterations", ylab = "X (samples obtained)")
```




```
## see how samples from chain compare to Beta(2.7, 6.3) density
hist(X, main = "Histogram of MCMC samples", prob = TRUE)
curve(expr = dbeta(x, 2.7, 6.3),
      from = 0, to = 2, add = TRUE, col = "blue")
```



Here is a varition of the M-H algorithm used previously, except we do not let the candidate distribution depend on previous values of the chain. The candidate distribution depends only on present values of the chain, in effect $q(y|x) = q(y)$.

```
## Metropolis Hastings
## now compare results with Beta(2.7, 6.3)

N = 10000
X = numeric(N)
X[1] = rbeta(n = 1, shape1 = 2.7, shape2 = 6.3)
for(i in 1:N){
  Y = rexp(n = 1, rate = 1)
  rho = (dbeta(x = Y, 2.7, 6.3) * dexp(x = X[i], rate = 1)) /
    (dbeta(x = X[i], 2.7, 6.3) * dexp(x = Y, rate = 1))

  if(runif(1) < rho){
    X[i+1] = Y
  } else{
    X[i+1] = X[i]
  }

}



print("sample mean from direct samples")
```

```
## [1] "sample mean from direct samples"
```

```r
mean(rbeta(n = 1000, shape1 = 2.7, shape2 = 6.3))
```

```
## [1] 0.3072212
```

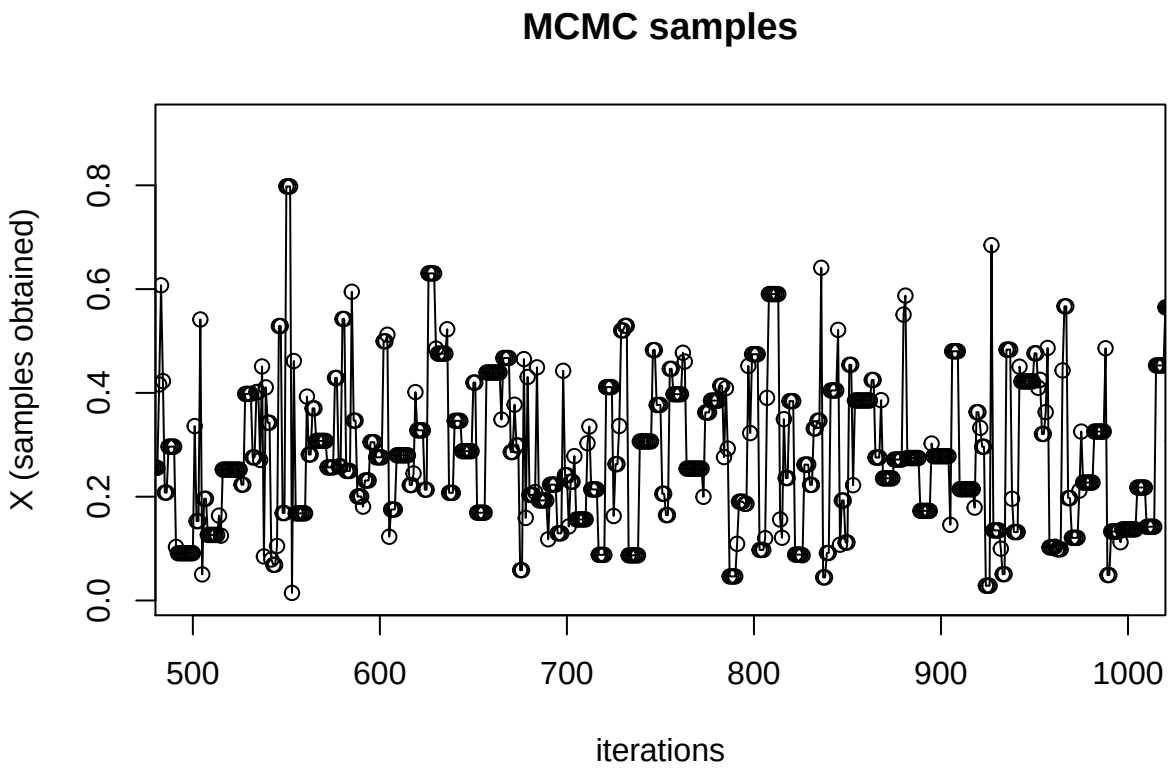
```r
print("sample mean from M-H samples")
```

```
## [1] "sample mean from M-H samples"
```

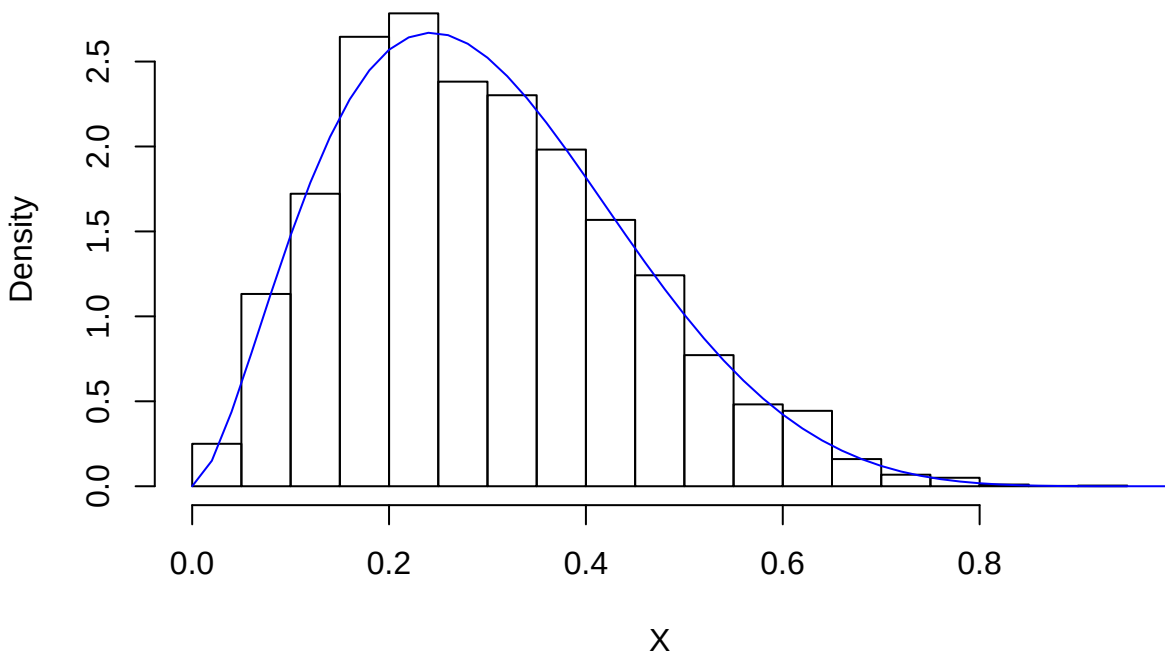
```r
mean(X)
```

```
## [1] 0.2982815
```

```r
## see chain transitions
plot(X, type = "o", main = "MCMC samples",
     xlim = c(500,1000),
     xlab = "iterations", ylab = "X (samples obtained)")
```




```r
## see how samples from chain compare to Beta(2.7, 6.3) density
hist(X, main = "Histogram of MCMC samples", prob = TRUE)
curve(expr = dbeta(x, 2.7, 6.3),
      from = 0, to = 2, add = TRUE, col = "blue")
```



This version of the M-H algorithm is known as the **Independent Metropolis-Hastings**. This method appears a generalization of the accept-reject algorithm in the sense that the instrumental distribution is the same density g as in the accept-reject algorithm. Thus, the proposed values Y_i are the same, if not the accepted ones.

The **Independent Metropolis-Hastings algorithm** as described Robert & Casella goes as follows Given $x^{(t)}$

1. Generate $Y_t \sim g(y)$
2. Take
$$X_{t+1} = \begin{cases} Y_t & \text{with probability } \rho(x^{(t)}, Y_t) \\ x^{(t)} & \text{with probability } 1 - \rho(x^{(t)}, Y_t) \end{cases}$$
where
$$\rho(x^{(t)}, Y_t) = \min\{\frac{f(Y_t)}{f(x^{(t)})}\frac{g(x^{(t)})}{g(Y_t)}, 1\}$$

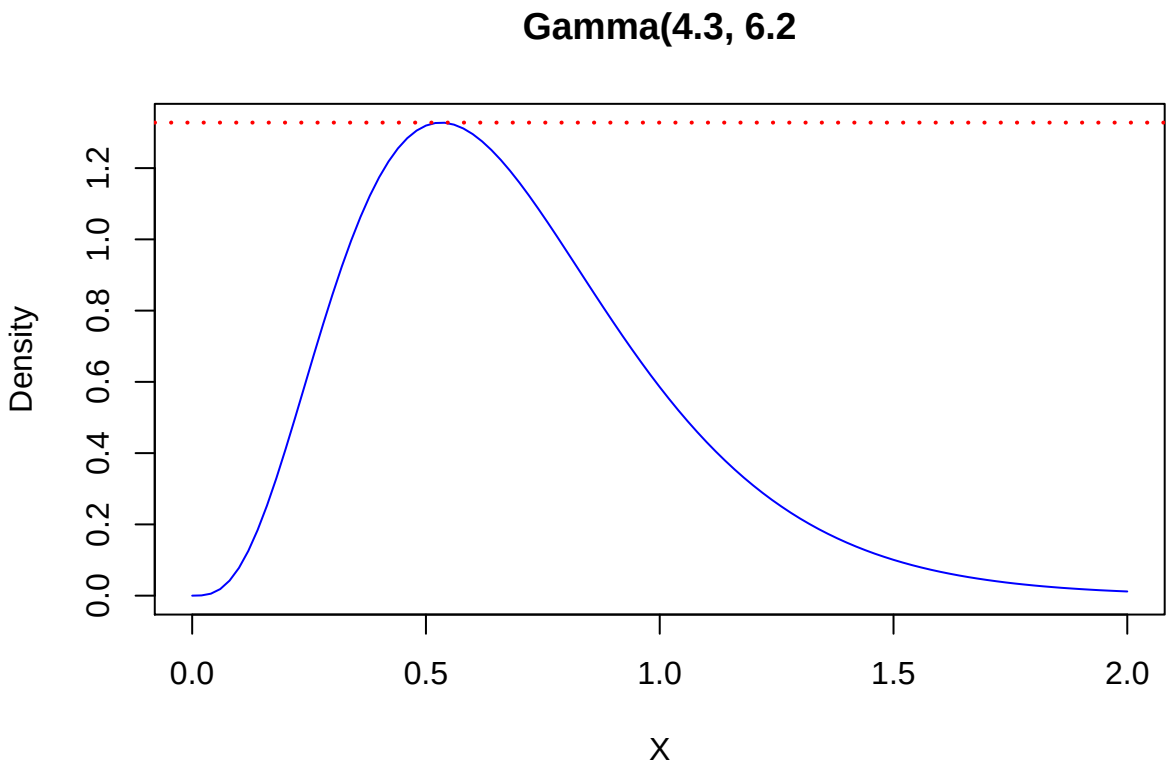
In simpler terms, as we want to generate $X \sim f$, we first take an initial value $x^{(0)}$ (which can almost be any arbitrary value in the support of f).

1. We generate a value $Y_0 \sim q(y|x^{(0)})$.
2. We calculate $\rho(x^{(t)}, Y_t)$
3. Generate a random value $U \sim Unif(0, 1)$
4. If $U < \rho(x^{(t)}, Y_t)$, then we accept $X^{(1)} = Y_t$; else we take $X^{(1)} = X^{(0)}$
5. Repeate steps 1-4 until you've satisfied the number of samples needed

Example: Gamma(4.3, 6.2)

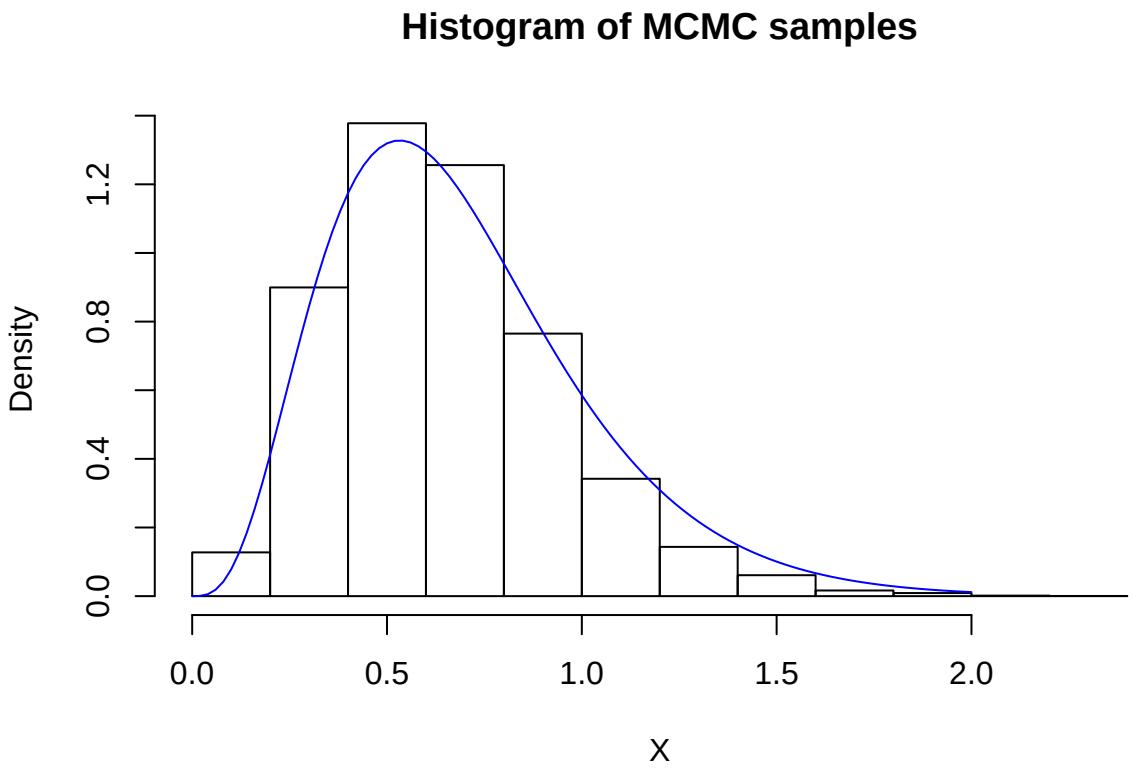
Here we will compare again the Accept-Reject algorithm against the Metropolis-Hastings. Generate N random variables $X \sim Gamma(4.3, 6.2)$.

```
N = 10000

## For accept-reject, we need to find a value for M
## we can use `optimize` to find the maximum of our target density
maximum  = optimize(f = function(x){ dgamma(x = x, shape = 4.3, rate = 6.2)},
                    interval = c(0, 2), maximum = TRUE ) ## obtain maximum

M = maximum$objective
curve(expr = dgamma(x = x, shape = 4.3, rate = 6.2),
      from = 0, to = 2, col = "blue",
      main = "Gamma(4.3, 6.2", xlab = "X", ylab = "Density")
abline(h = M, lty = 3, lwd = 2, col = "red")
```




```
f = function(x){
  dgamma(x = x, shape = 4.3, rate = 6.2)
}

g = function(x){
  dgamma(x = x, shape = 4, rate = 7)
}


#N = 10
X = numeric(N)
i = 0
```

```r
while(i < N){
  Y = rgamma(n = 1, shape = 4, rate = 7)
  U = runif(1)
  if(U*M <= f(Y)/g(Y)){
    i = i + 1
    X[i] = Y
  }
}

## see how samples from chain compare to Gamma density
hist(X, main = "Histogram of MCMC samples", prob = TRUE)
curve(expr = dgamma(x = x, shape = 4.3, rate = 6.2),
      from = 0, to = 2, add = TRUE, col = "blue")
```



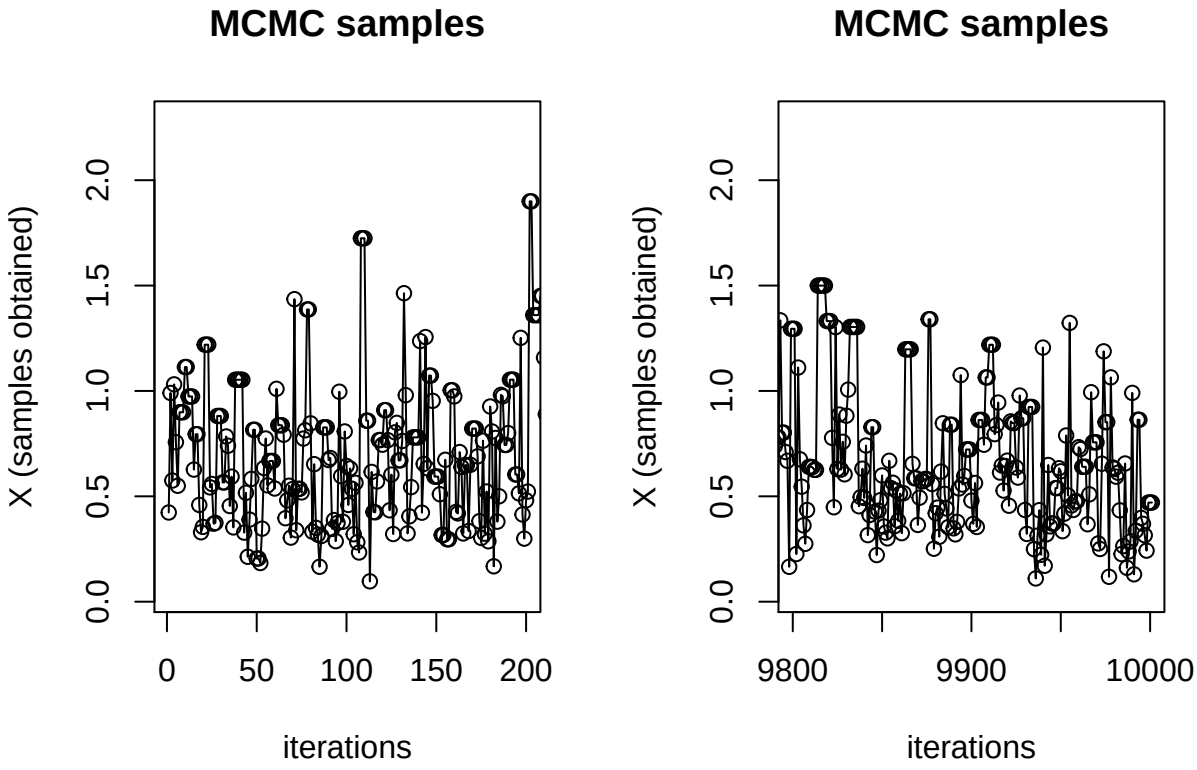

```r
## Metropolis Hastings

N = 10000
X = numeric(N)
X[1] = rgamma(n = 1, shape = 4.3, rate = 6.2)
for(i in 1:N){
  Y = rgamma(n = 1, shape = 4, rate = 7)
  rho = (dgamma(x = Y, shape = 4.3, rate = 6.2) * dgamma(x = X[i], shape = 4, rate = 7)) /
    (dgamma(x = X[i], shape = 4.3, rate = 6.2) * dgamma(x = Y, shape = 4, rate = 7))
  #X[i+1] = X[i] + (Y - X[i])*(runif(1) < rho) ## equivalent to if-else statement below
  if(runif(1) < rho){
    X[i+1] = Y
  } else{
    X[i+1] = X[i]
  }
```

```
}
```

```
mean(X)
```

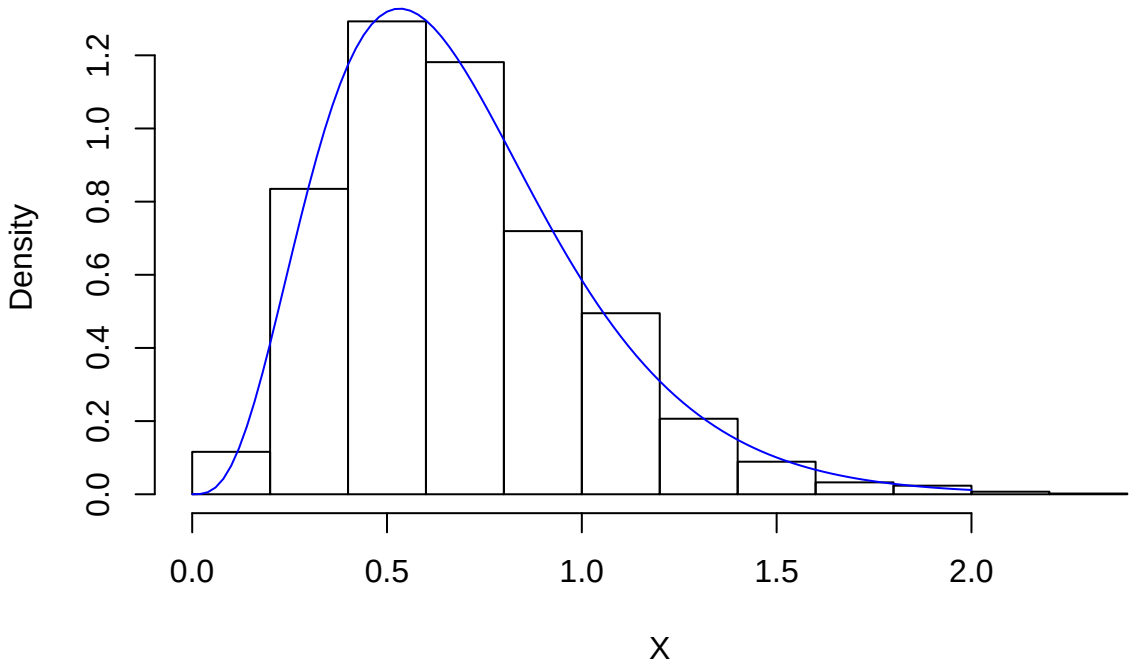
```
## [1] 0.690021
```

```
## see chain transitions
par(mfrow = c(1,2))
plot(X, type = "o", main = "MCMC samples",
     xlim = c(1,200),
     xlab = "iterations", ylab = "X (samples obtained)")

plot(X, type = "o", main = "MCMC samples",
     xlim = c(N-200,N),
     xlab = "iterations", ylab = "X (samples obtained)")
```




```
par(mfrow = c(1,1))
```

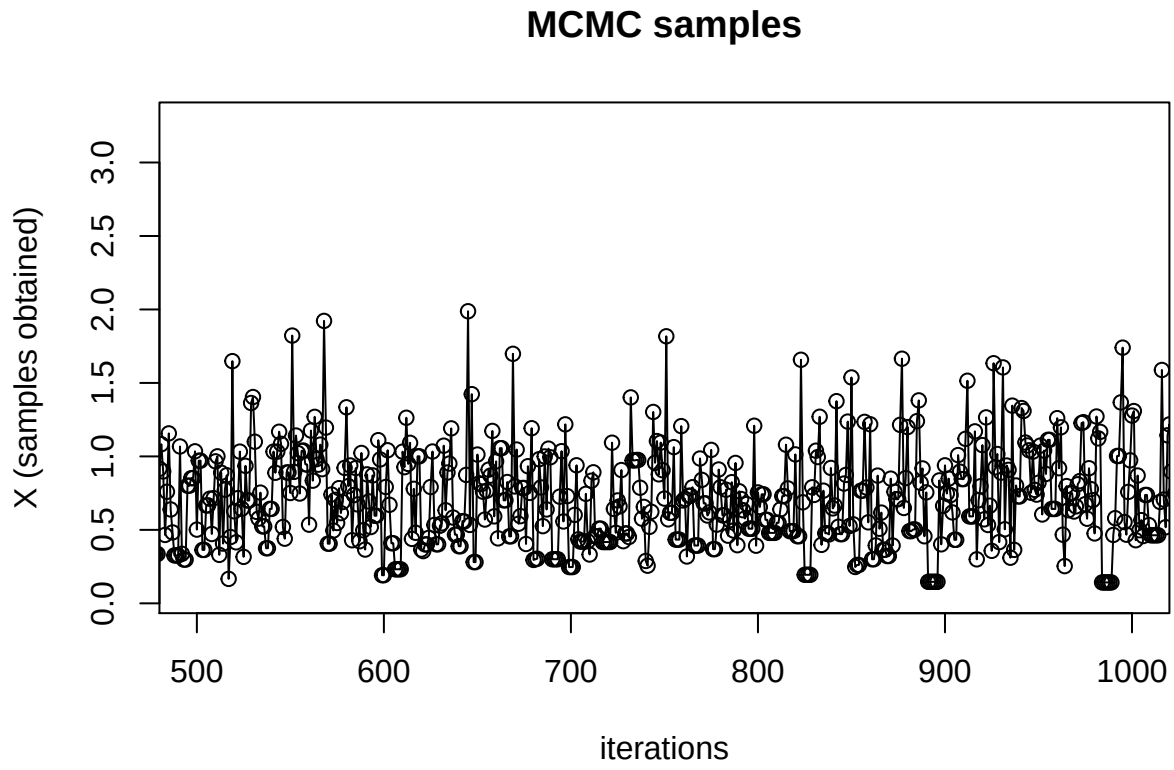
```
## see how samples from chain compare to Gamma density
hist(X, main = "Histogram of MCMC samples", prob = TRUE)
curve(expr = dgamma(x = x, shape = 4.3, rate = 6.2),
      from = 0, to = 2, add = TRUE, col = "blue")
```



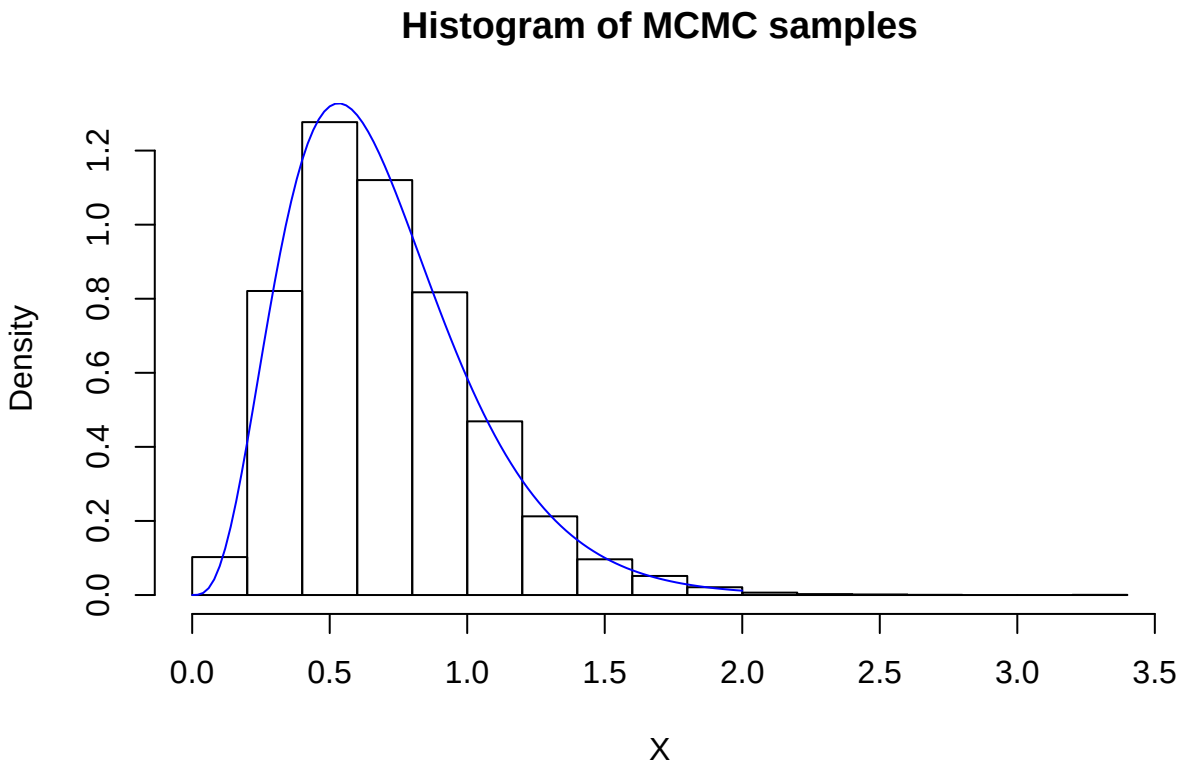

```
## Metropolis Hastings
## now compare results with Gamma(5,6)

N = 10000
X = numeric(N)
X[1] = rgamma(n = 1, shape = 4.3, rate = 6.2)
for(i in 1:N){
  Y = rgamma(n = 1, shape = 5, rate = 6)
  rho = (dgamma(x = Y, shape = 4.3, rate = 6.2) * dgamma(x = X[i], shape = 5, rate = 6)) /
    (dgamma(x = X[i], shape = 4.3, rate = 6.2) * dgamma(x = Y, shape = 5, rate = 6))
  #X[i+1] = X[i] + (Y - X[i])*(runif(1) < rho) ## equivalent to if-else statement below
  if(runif(1) < rho){
    X[i+1] = Y
  } else{
    X[i+1] = X[i]
  }

}



mean(X)
```

```
## [1] 0.6991385
```

```
## see chain transitions
plot(X, type = "o", main = "MCMC samples",
     xlim = c(500,1000),
     xlab = "iterations", ylab = "X (samples obtained)")
```




```
## see how samples from chain compare to Gamma density
hist(X, main = "Histogram of MCMC samples", prob = TRUE)
curve(expr = dgamma(x = x, shape = 4.3, rate = 6.2),
      from = 0, to = 2, add = TRUE, col = "blue")
```



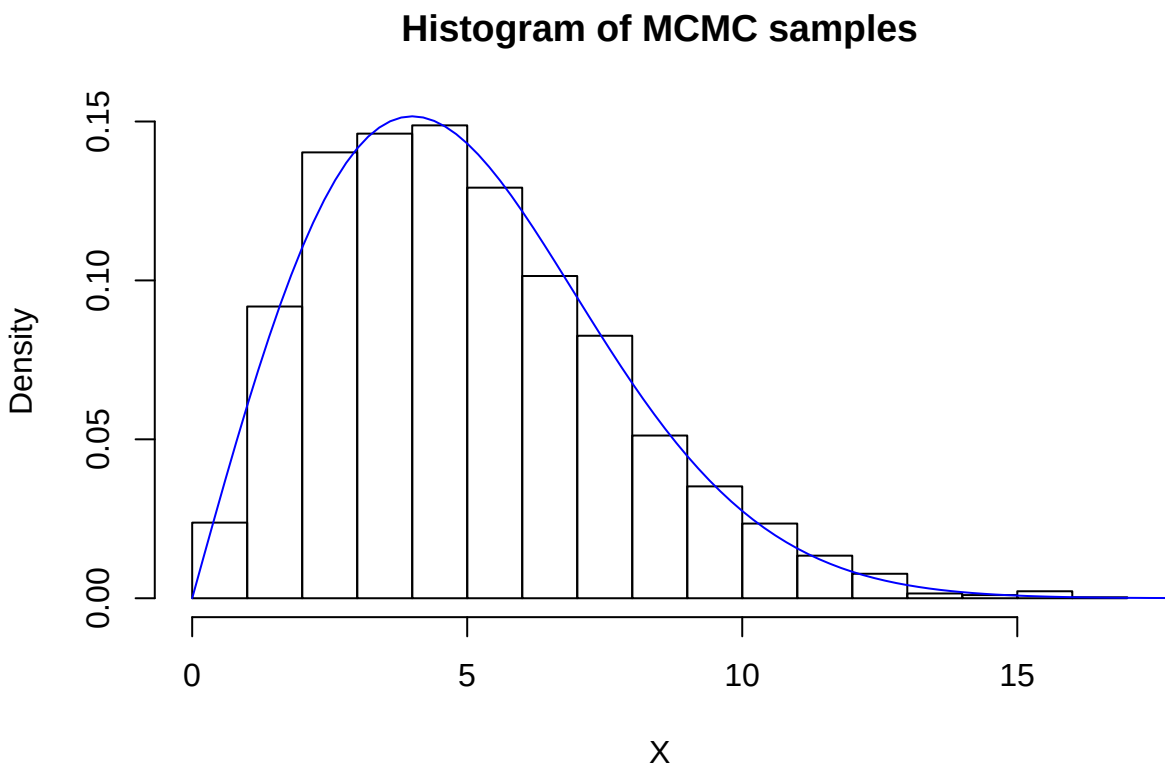
Example 3: Rayleigh distribution

Use the M-H algorithm to generate samples from a Rayleigh distrubition. The Rayleigh density is

$$f(x) = \frac{x}{\sigma^2} e^{-x^2/(2\sigma^2)} \text{ , given that } \; x \geq 0, \; \sigma > 0$$

We will use the χ^2 distribution as a candidate distribution to sample from.

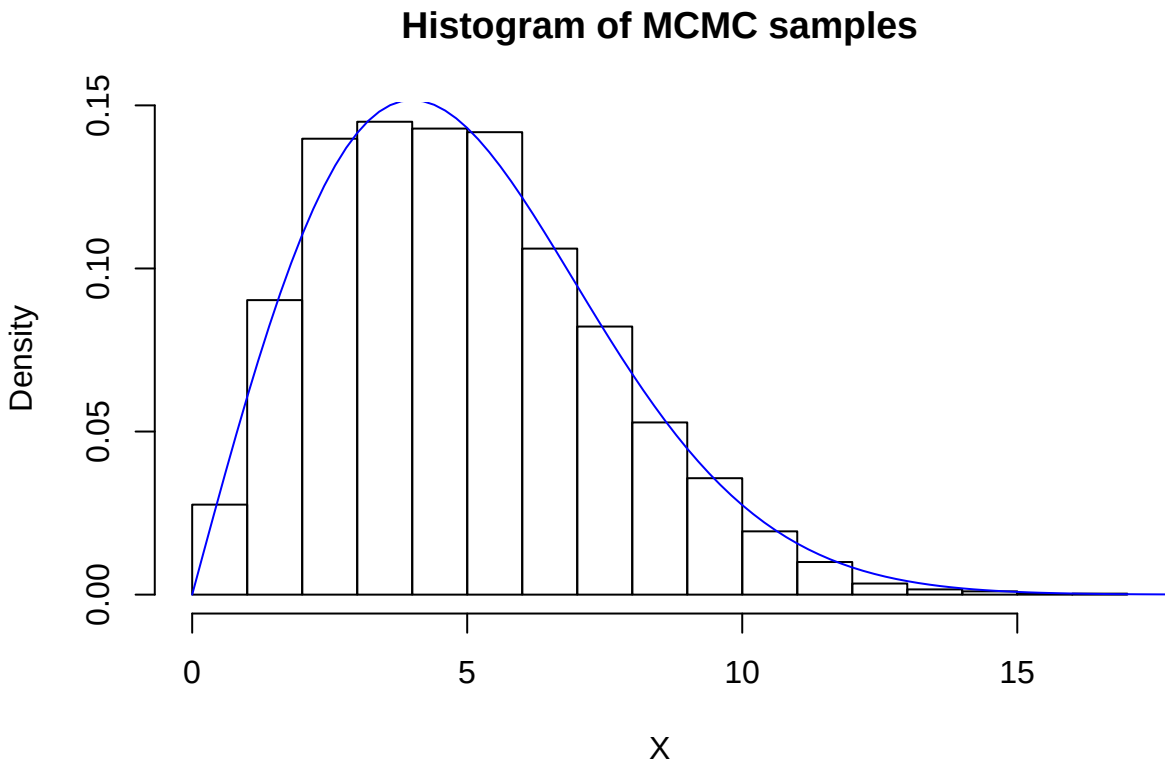
```
dRayleigh = function(x, sigma = 4){
  ## any needed for `curve`
  if(any(x < 0)){
    out = 0
  } else{
  out = (x/sigma^2) * exp(-x^2 / (2 * sigma^2))
  }
  return(out)
}

N = 10000

X = numeric(N)
X[1] = 2 ## some random value

for(i in 1:N){
  Y = rchisq(n = 1, df = X[i])
  U = runif(1)
  rho = (dRayleigh(Y) * dchisq(x = X[i], df = Y)) /
        (dRayleigh(X[i]) * dchisq(x = Y, df = X[i]))
  if(U < rho){
    X[i+1] = Y
  } else {
    X[i+1] = X[i]
  }
}



## see how samples from chain compare to Gamma density
hist(X, main = "Histogram of MCMC samples", prob = TRUE)
curve(expr = dRayleigh,
      from = 0, to = 20, add = TRUE, col = "blue")
```



We will use the $Gamma(X_t, 1)$ distribution as a candidate distribution to sample from.

```
dRayleigh = function(x, sigma = 4){
  ## any needed for `curve`
  if(any(x < 0)){
    out = 0
  } else{
  out = (x/sigma^2) * exp(-x^2 / (2 * sigma^2))
  }
  return(out)
}

N = 10000

X = numeric(N)
X[1] = 2 ## some random value

for(i in 1:N){
  Y = rgamma(n = 1, shape = X[i], rate = 1)
  U = runif(1)
  rho = (dRayleigh(Y) * dgamma(x = X[i], shape = Y, rate = 1)) /
        (dRayleigh(X[i]) * dgamma(x = Y, shape = X[i], rate = 1))
  if(U < rho){
    X[i+1] = Y
  } else {
    X[i+1] = X[i]
  }
}
```

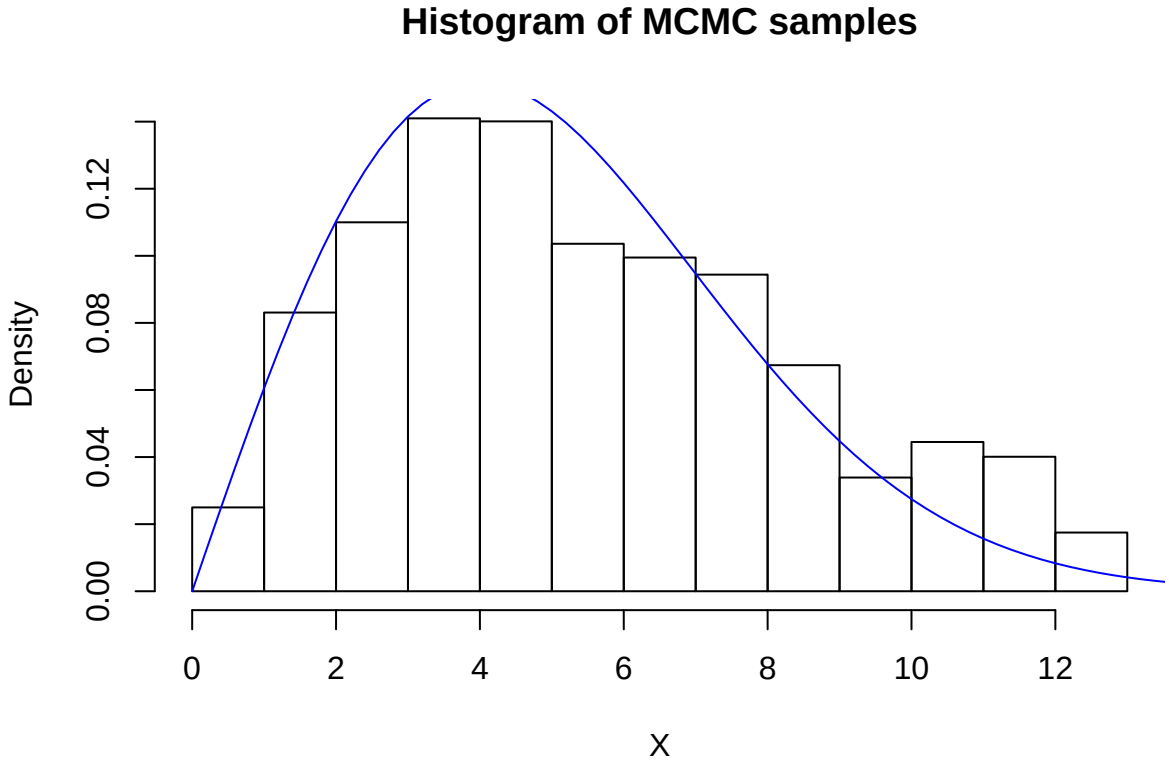
```r
## see how samples from chain compare to Gamma density
hist(X, main = "Histogram of MCMC samples", prob = TRUE)
curve(expr = dRayleigh,
      from = 0, to = 20, add = TRUE, col = "blue")
```



Now we implement an independent M-H version. We will use the $Gamma(2, 1)$ distribution as a candidate distribution to sample from.

```r
dRayleigh = function(x, sigma = 4){
  ## any needed for `curve`
  if(any(x < 0)){
    out = 0
  } else{
  out = (x/sigma^2) * exp(-x^2 / (2 * sigma^2))
  }
  return(out)
}


N = 10000

X = numeric(N)
X[1] = 2 ## some random value

for(i in 1:N){
  Y = rgamma(n = 1, shape = 2, rate = 1)
  U = runif(1)
  rho = (dRayleigh(Y) * dgamma(x = X[i], shape = 2, rate = 1)) /
        (dRayleigh(X[i]) * dgamma(x = Y, shape = 2, rate = 1))
  if(U < rho){
```

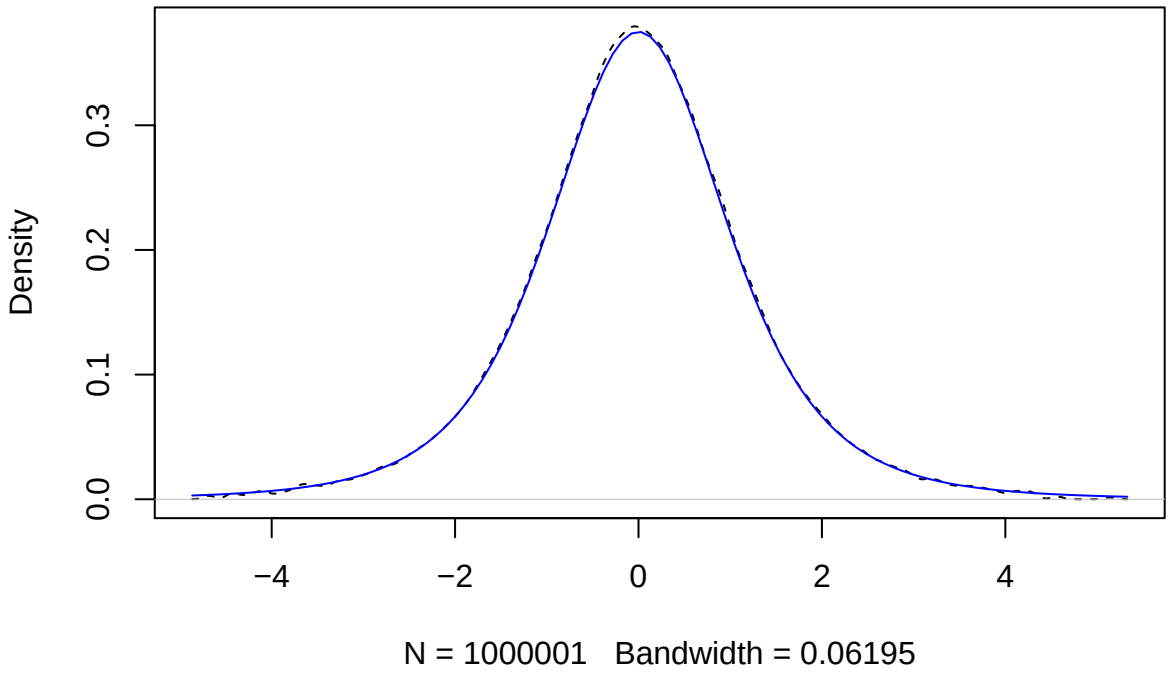
```
    X[i+1] = Y
  } else {
    X[i+1] = X[i]
  }
}



## see how samples from chain compare to Gamma density
hist(X, main = "Histogram of MCMC samples", prob = TRUE)
curve(expr = dRayleigh,
      from = 0, to = 20, add = TRUE, col = "blue")
```



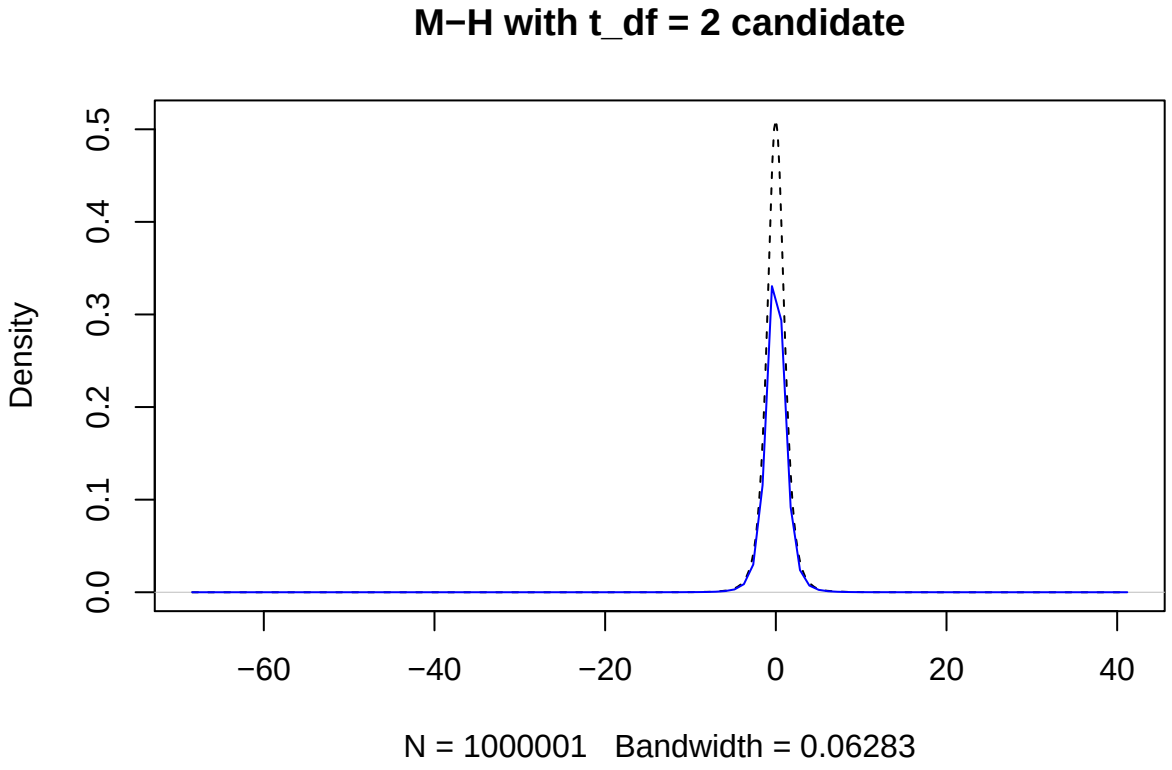
Student's t density with v degres of freedom is given by [insert pdf here]

Calculate the mean of a t distribution with $v = 4$ degrees of freedom using a M-H algorithm with candidate densities $N(0, 1)$ and $t_{v=2}$.

```
set.seed(987)
N = 10^6

#dt(x = x, df = 4)

X = numeric(N)
X[1] = rnorm(1) ## initialize the starting value

for(i in 1:N){
    Y = rnorm(1) ## independent of X_i
    rho = (dt(Y, df = 4) * dnorm(X[i])) /
            (dt(X[i], df = 4) * dnorm(Y))
    U = runif(1)
    if(U <= rho){
        X[i+1] = Y
    } else{
        X[i+1] = X[i]
    }

}




plot(density(X), type = "l",
        lty = 2, main = "M-H with N(0,1) candidate")
curve(dt(x, df = 4), add = TRUE, col = 4)
```




```
## now with a t distribution with df = 2

X = numeric(N)
X[1] = rt(1, df = 2) ## initialize the starting value

for(i in 1:N){
    Y = rt(1, df = 2) ## independent of X_i
    rho = (dt(Y, df = 4) * dt(X[i], df = 2)) /
            (dt(X[i], df = 4) * dt(Y, df = 2))
    U = runif(1)
    if(U <= rho){
        X[i+1] = Y
    } else{
        X[i+1] = X[i]
    }

}




plot(density(X), type = "l",
        lty = 2, main = "M-H with t_df = 2 candidate")
curve(dt(x, df = 4), add = TRUE, col = 4)
```



Bayesian Analysis

O-ring Challenger Data

The following is a well covered logistic regression example using the O-ring data set related to the 1986 space shuttle Challenger exploision. The output is modeled as the probability of failure (Y = 1) given the data.

$$P(Y = 1|X = x) = p = \frac{exp(\alpha + x\beta)}{1 + exp(\alpha + x\beta)}$$

Or, equivalently with the logit transformation on P

$$logit(p) = \frac{p}{1-p} = \alpha + x\beta$$

```
failure <- c(1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
             0, 0, 0, 1, 0, 0, 0, 0, 0)
temperature <- c(53, 57, 58, 63, 66, 67, 67, 67, 68,
                 69, 70, 70, 70, 70, 72, 73, 75, 75,
                 76, 76, 78, 79, 81)

df = data.frame(failure, temperature)

head(df)
```

```
##   failure temperature
## 1       1          53
## 2       1          57
## 3       1          58
```

```
## 4       1           63
## 5       0           66
## 6       0           67
```

The frequentist logistic regression

```
fit = glm(formula = failure ~ temperature, data = df,
          family = binomial(link = "logit"))

summary(fit)
```

```
##
## Call:
## glm(formula = failure ~ temperature, family = binomial(link = "logit"),
##     data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.0611  -0.7613  -0.3783   0.4524   2.2175
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  15.0429     7.3786   2.039   0.0415 *
## temperature  -0.2322     0.1082  -2.145   0.0320 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 28.267  on 22  degrees of freedom
## Residual deviance: 20.315  on 21  degrees of freedom
## AIC: 24.315
##
## Number of Fisher Scoring iterations: 5
```

Observed are

$$Y_i \sim Bernoulli(p(x_i)), \text{ where } p(x_i) = \frac{exp(\alpha + x_i\beta)}{1 + exp(\alpha + x_i\beta)}$$

where $p(x_i)$ is the probability of O-ring failure at temperature x_i. The likelihood is

$$L(\alpha, \beta|\mathbf{y}) \propto \prod_{i=1}^{n} (\frac{exp(\alpha + x_i\beta)}{1 + exp(\alpha + x_i\beta)})^{y_i} \times (\frac{1}{1 + exp(\alpha + x_i\beta)})^{1-y_i}$$

and as a prior Robert & Casella choose

$$\pi_\alpha(\alpha|b) \times \pi_\beta(\beta) = \frac{1}{b} e^{\alpha} e^{-e^{\alpha/b}}$$

which puts an exponential prior on $log(\alpha)$ and a flat prior on β (uniform), and insures a proper posterior distribution.

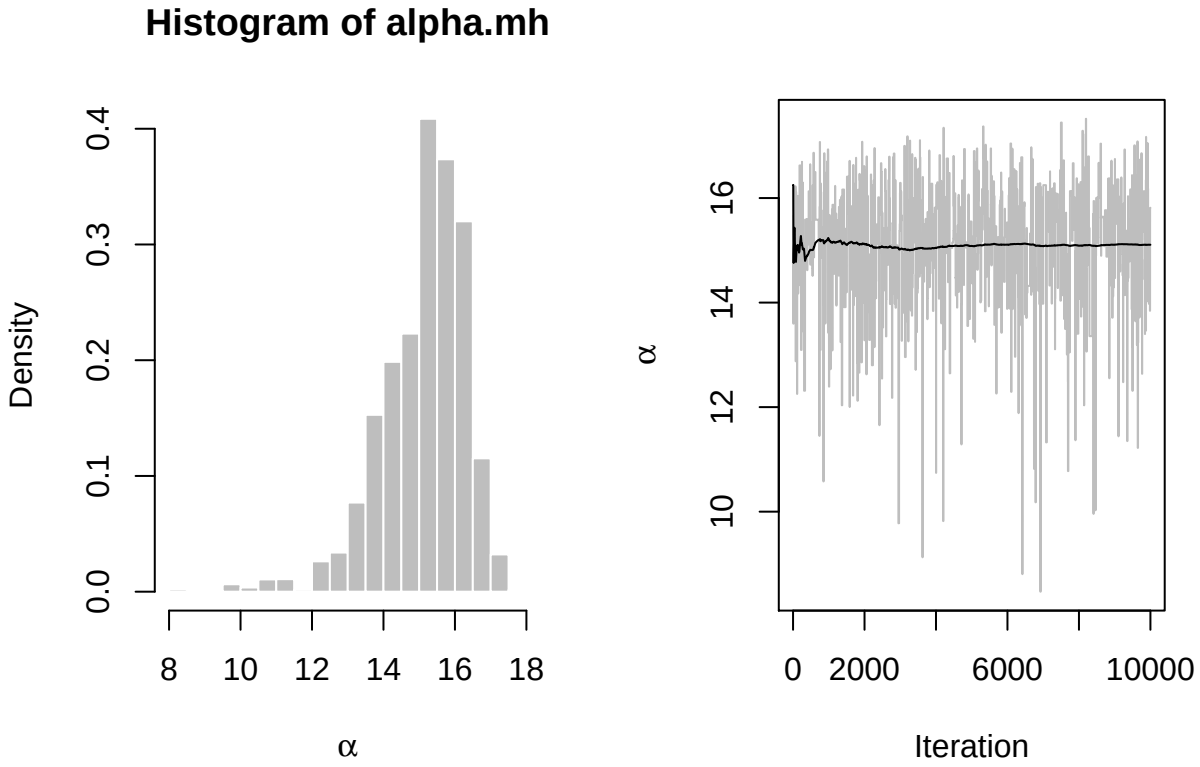
```
# Preliminary output from ML estimation
a.mle <- as.numeric(fit$coefficients[1])
b.mle <- as.numeric(fit$coefficients[2])
var.a.mle <- summary(fit)$cov.scaled[1, 1]
var.b.mle <- summary(fit)$cov.scaled[2, 2]
b.mme <- exp(a.mle + 0.577216)
```

```
## setting up functions

# Posterior distribution
dpost <- function(theta, y = failure, x = temperature){
    ## density of Y is binomial/bernoulli
    a <- theta[1]
    b <- theta[2]
    p <- 1 - 1 / (1 + exp(a + b * x)) ## logistic CDF
    lik <- exp(sum(dbinom(y, size=1, prob=p, log=TRUE)))
    dprior <- exp(a) * exp(-exp(a) / b.mme) * 1/ b.mme ## density of prior
    return(lik * dprior)
}


# Proposal distribution (independent proposal, so "theta0" is not used)
dprop <- function(theta){
    ## ignore theta0
    a <- theta[1]
    b <- theta[2]
    pr1 <- exp(a) * exp(-exp(a) / b.mme) / b.mme
    pr2 <- dnorm(b, b.mle, sqrt(var.b.mle))
    return(pr1 * pr2)
}




rprop <- function(theta0){
    ## independent proposals for a and b
    #a <- log(rexp(1, 1 / b.mme))
    a <- log(rexp(1, 1 / b.mme)) ## log for computational purposes
    b <- rnorm(1, b.mle, sqrt(var.b.mle))
    return(c(a, b))
}
```

```
## Metropolis-Hastings set up to run


# Run Metropolis-Hastings
N <- 10^4
B <- 10^3
#mh.out <- MH(c(a.mle, b.mle), dpost, dprop, rprop, N, B)


## x0: initializing values for MH algorithm
## f: the pdf of posterior; target distribution
## dprop: the pdf of the proposal distribution
## rprop: the random number gen for proposal
## N: Number of samples needed
## B: Number of burn in samples
## function generates N+B samples then drops B
x0 <- c(a.mle, b.mle)
x <- matrix(NA, nrow = N + B, ncol = length(x0))
fx <- rep(NA, N + B)
```

```
x[1,] <- x0 ## add the initializing values
fx[1] <- dpost(x0) ## initiate the M-H algorithm
ct <- 0 ## counter for acceptance

for(i in 2:(N + B)){
    u <- rprop(x[i-1,])
    fu <- dpost(u)
    ## use log values for computational purposes
    r <- log(fu) + log(dprop(x[i-1,])) - log(fx[i-1]) - log(dprop(u))
    R <- min(exp(r), 1)
    if(runif(1) <= R){
        ct <- ct + 1 ## update counter
        x[i,] <- u
        fx[i] <- fu
    } else {
    x[i,] <- x[i-1,]
    fx[i] <- fx[i-1]
    }
}


mh.out <- x[-(1:B), ]
alpha.mh <- mh.out[,1]
beta.mh <- mh.out[,2]
```

```
par(mfrow = c(1,2))
hist(alpha.mh, freq=FALSE, col="gray", border="white", xlab=expression(alpha))
plot(alpha.mh, type="l", col="gray", xlab="Iteration", ylab=expression(alpha))
lines(1:N, cumsum(alpha.mh) / (1:N))
```



```
hist(beta.mh, freq=FALSE, col="gray", border="white", xlab=expression(beta))
plot(beta.mh, type="l", col="gray", xlab="Iteration", ylab=expression(beta))
lines(1:N, cumsum(beta.mh)/(1:N))
```