MCMC for the Frechet Distribution

Andy Shen, Devin Francom

LANL: CCS-6

Task 1

Write a function in R called `approxPi(n)` that uses n Monte Carlo samples to approximate pi (3.1415...). Look on the Wikipedia page for "Monte Carlo method" under "Overview" for hints.

Distance Function

We first write a function to calculate the euclidean distance for a set of points in n-dimensions.

```
distance <- function(a, b){
  sqrt( sum((a - b)^2) )
}
```

Algorithm

Our algorithm samples random uniform values from 0 to 1 and accepts all points where the distance is less than 1. These accepted points are in our quarter-circle.

We approximate the ratio of points in the circle to the total number of points is $\frac{\pi}{4}$. We multiply by 4 to obtain our approximation of π.

```
set.seed(12)
approxPi <- function(n) {
  x <- runif(n)
  y <- runif(n)
  pts <- data.frame(x, y)
  sq_dist <- (pts - c(0,0))^2 # you technically don't need the c(0,0)
  dists <- sqrt(rowSums(sq_dist))
  circle <- dists[dists < 1]
  ratio <- length(circle) / length(dists)
  4 * ratio
}
```

Testing it Out

```
pi
```

```
## [1] 3.141593
```

```r
approxPi(10^6)
```

```
## [1] 3.145056
```

Task 2

Write a function in R called `rfrechet(n,alpha,s,m)` that generates n random samples from the Frechet distribution with the given parameters (see the Wikipedia page for "Frechet distribution").

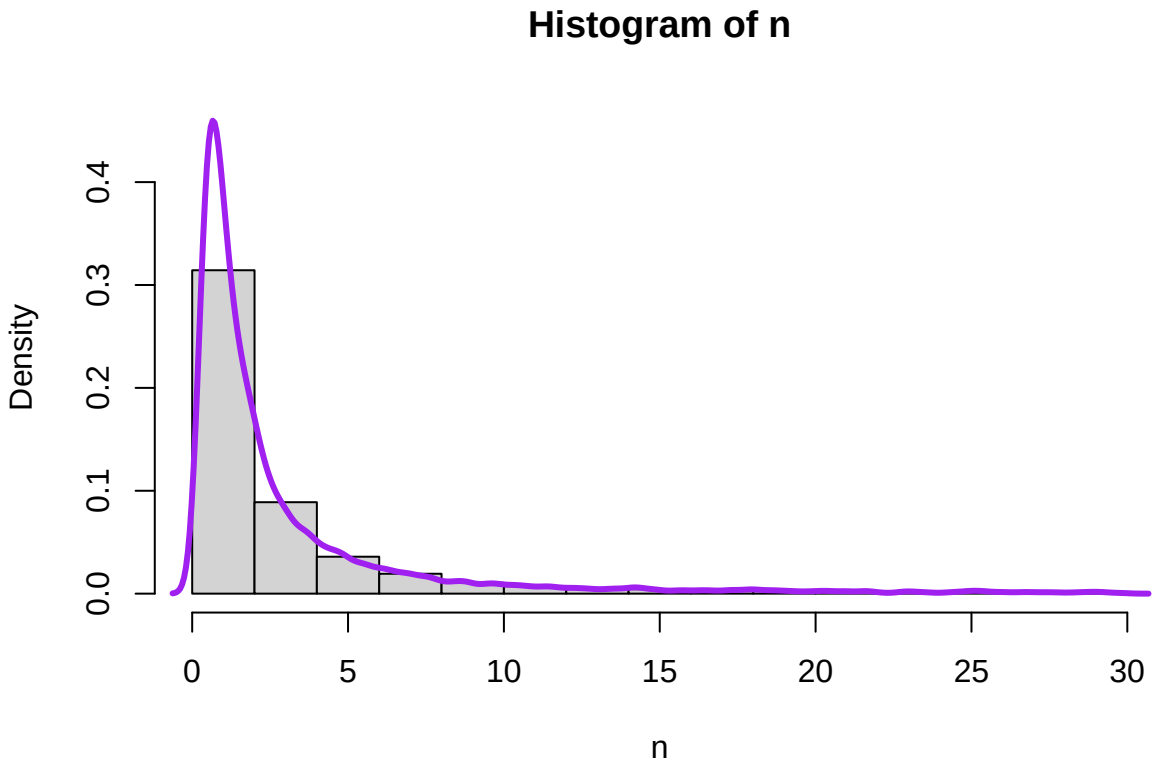
You could try a few methods, but ultimately we'll be interested in MCMC using the Metropolis-Hastings algorithm.

This one will be tricky, since there is a lot to learn about M-H and MCMC algorithms, but it should give you good intuition for the type of problems you can solve with these algorithms. (By the way, there are easy ways to sample from the Frechet, we're just doing this to learn MCMC).

Sampling from the Frechet Distribution

We first draw 10000 random values from the Frechet distribution with the default parameters. For clarity, we keep all values less than 30 and we see that the distribution of points is heavily right-skewed.

```
library(evd)
n <- rfrechet(10000) # Same default parameter values
n <- n[n < 30]
hist(n, prob = TRUE, ylim = c(0, max(density(n)$y)))
lines(density(n), lwd = 3, col = "purple")
```



MCMC Algorithm

Our MCMC algorithm uses a log transformation to account for potentially large differences in values from our target distribution and keep everything on the same scale.

We use the **Truncated Normal Distribution** as our proposal distribution. The Truncated Normal distribution restricts the random variable to a specific range. We use this as the proposal distribution to eliminate numerically unstable values that result from a Normal distribution. This is because our x value in the Frechet distribution cannot be negative.

Through trial-and-error, we set our truncated normal distribution to have a mean of the current candidate value, and a fixed standard deviation of 6.32.

```
library(truncnorm)
r_frechet <- function(start = 0, n = 10000, alpha = 1, s = 1, m = 0) {

  p <- function(x){ # target
    log((alpha / s) * ( ((x - m) / s)^(-1 - alpha) )) + (-( (x - m)/s ) ^ -alpha)
    # frechet PDF
  }

  propose <- function(x, mu, m = m) {
    log(dtruncnorm(x, a = 0, mean = mu, sd = 6.32))
  }

  values <- rep(NA, n)

  a <- 0
  values[1] <- start
  for(i in 2:n) {
    current <- values[i-1]
    y <- rtruncnorm(1, a = 0, mean = current, sd = 6.32)

    pmove <- min(
      0, (p(y) - propose(y, current)) - ((p(current) - propose(current, y)))
    )

    #if(i > 5000) browser()

    u <- log(runif(1))
    if(u < pmove) {
      values[i] <- y
      a <- a + 1
    }
    else {
      values[i] <- values[i-1]
    }
  }
  return(list(values, a))
}
```

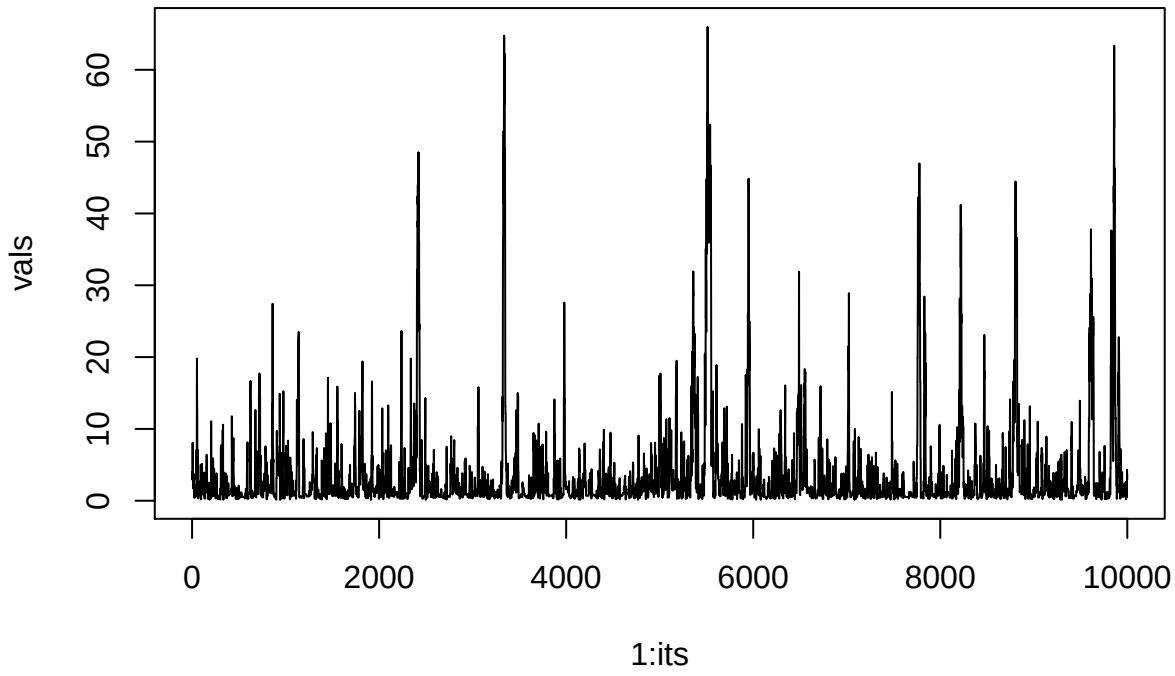
Testing it Out

We test our algorithm with multiple parameter combinations. We see that our algorith works best when $m = 1$.

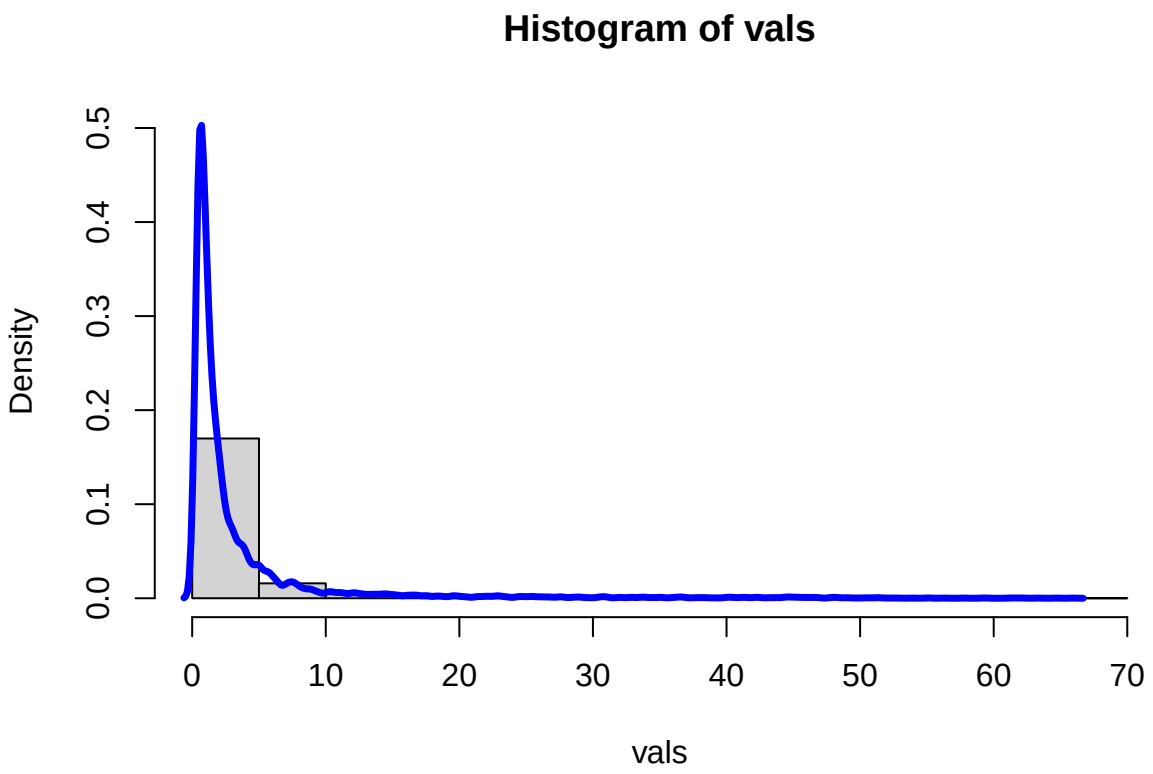
```
its <- 10000
results <- r_frechet(start = 3, n = its) # default parameter values
vals <- results[[1]]
a <- results[[2]]
cat("Acceptance Rate:", a/its, "\n")
```

```
## Acceptance Rate: 0.3527
```

```
plot(1:its, vals, type = "l")
```



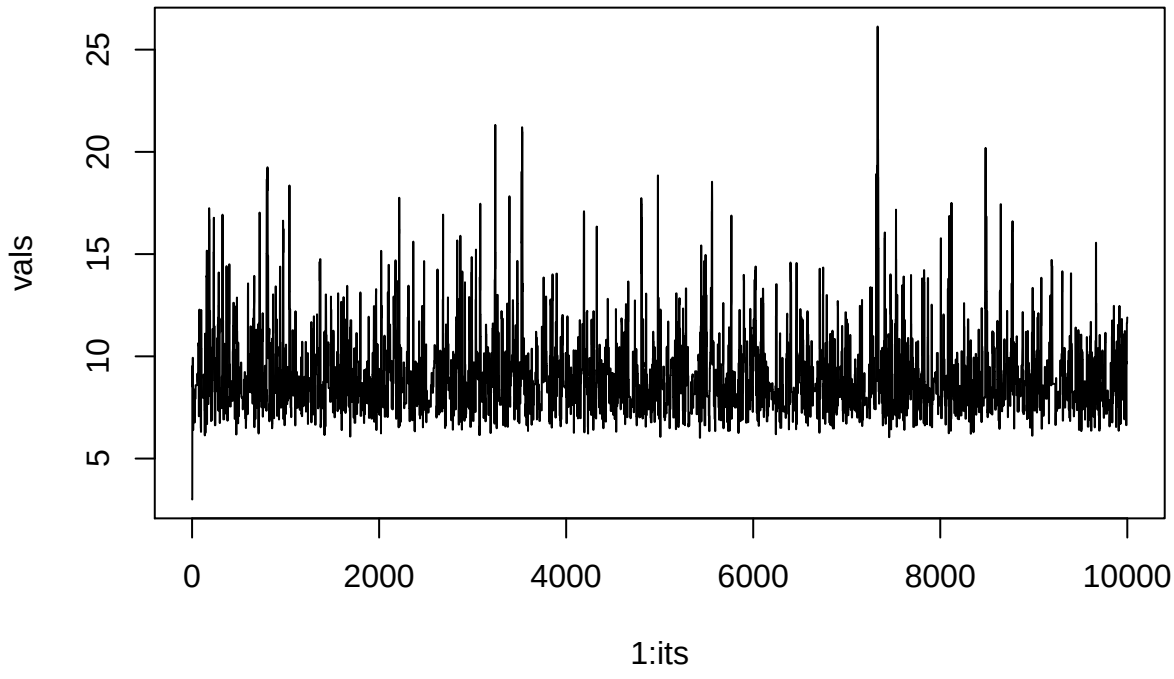

```
hist(vals, prob = TRUE, ylim = c(0, max(density(vals)$y)))
lines(density(vals), col = "blue", lwd = 3.5)
```



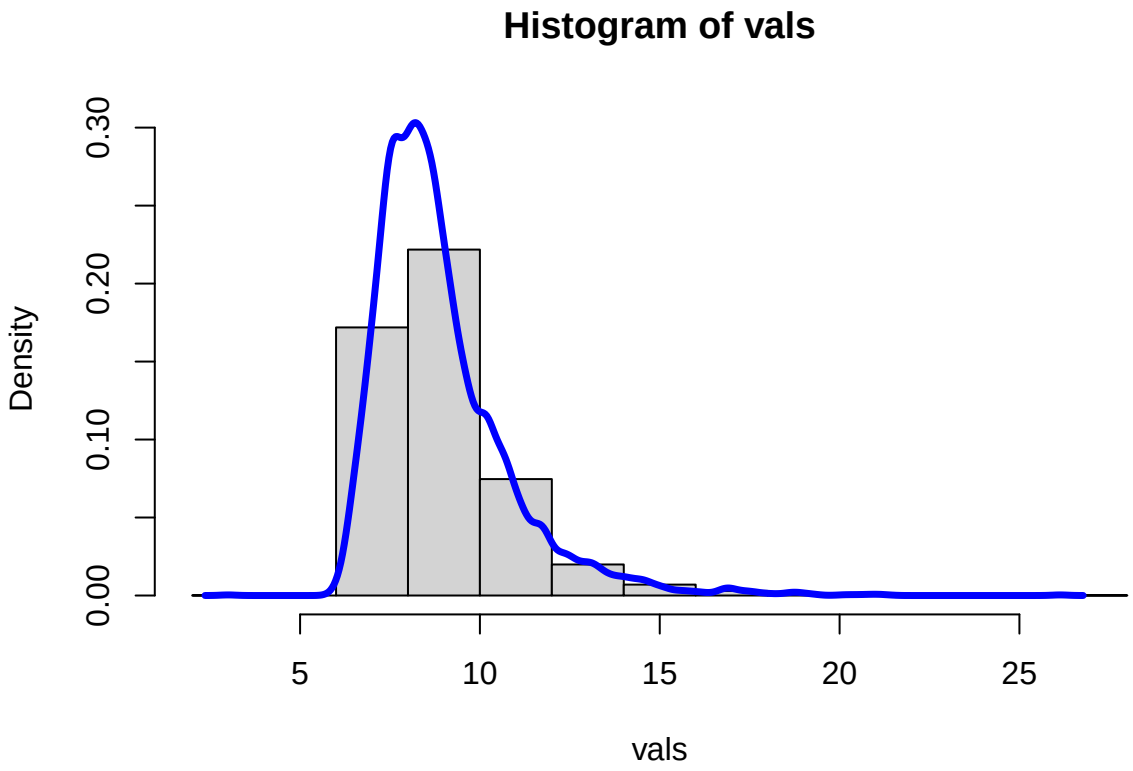

```
its <- 10000
results <- r_frechet(start = 3, alpha = 7, s = 8, m = 0, n = its)
vals <- results[[1]]
a <- results[[2]]
cat("Acceptance Rate:", a/its, "\n")
```

```
## Acceptance Rate: 0.2807
```

```
plot(1:its, vals, type = "l")
```



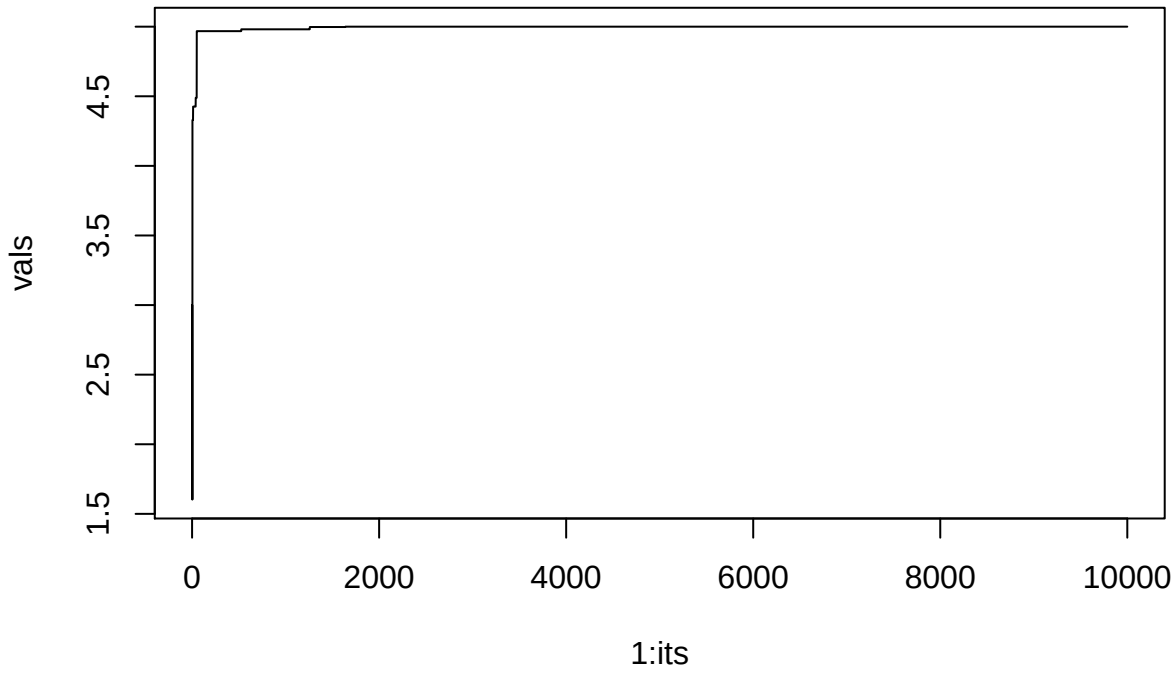
```r
hist(vals, prob = TRUE, ylim = c(0, max(density(vals)$y)))
lines(density(vals), col = "blue", lwd = 3.5)
```



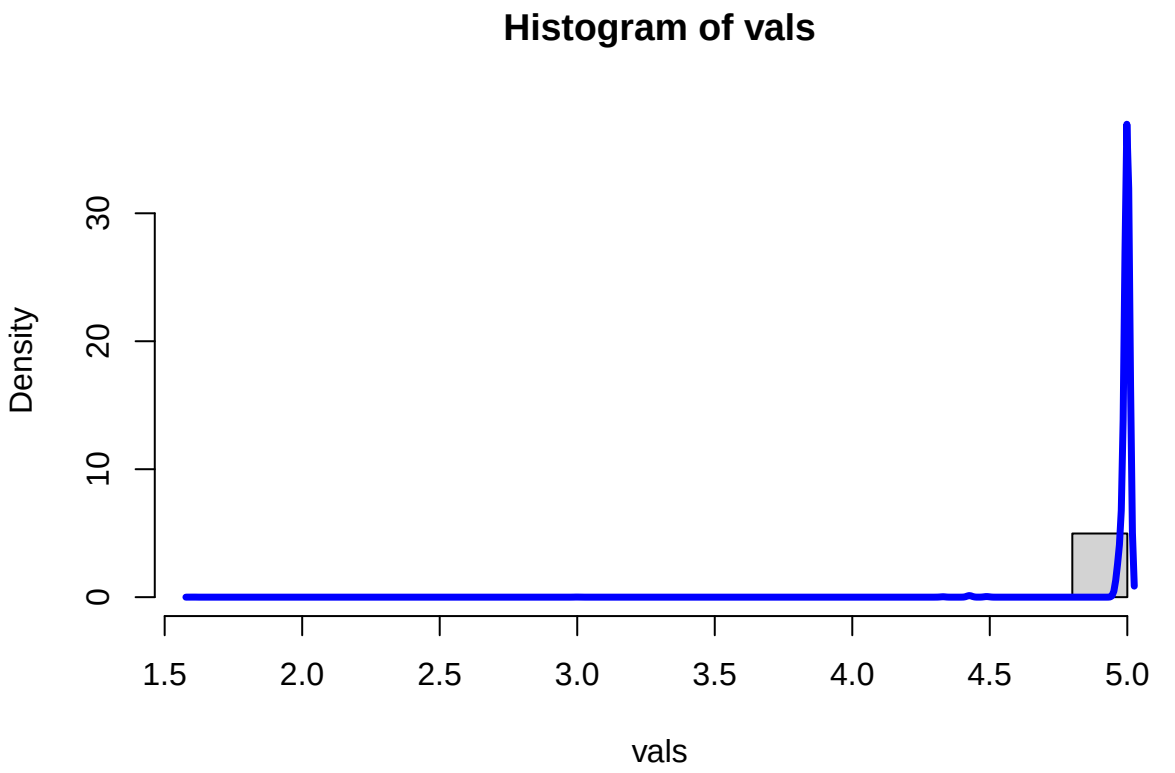

```r
its <- 10000
results <- r_frechet(start = 3, alpha = 1, s = 1, m = 5, n = its)
vals <- results[[1]]
a <- results[[2]]
cat("Acceptance Rate:", a/its, "\n")
```

```
## Acceptance Rate: 8e-04
```

```r
plot(1:its, vals, type = "l")
```




```
hist(vals, prob = TRUE, ylim = c(0, max(density(vals)$y)))
lines(density(vals), col = "blue", lwd = 3.5)
```




```
its <- 10000
results <- r_frechet(start = 3, alpha = 6, s = 5, m = 1, n = its)
```

```
## Warning in log((alpha/s) * (((x - m)/s)^(-1 - alpha))): NaNs produced
```

```
## Error in if (u < pmove) {: missing value where TRUE/FALSE needed
```

Normal Distribution Sampling with Metropolis Algorithm

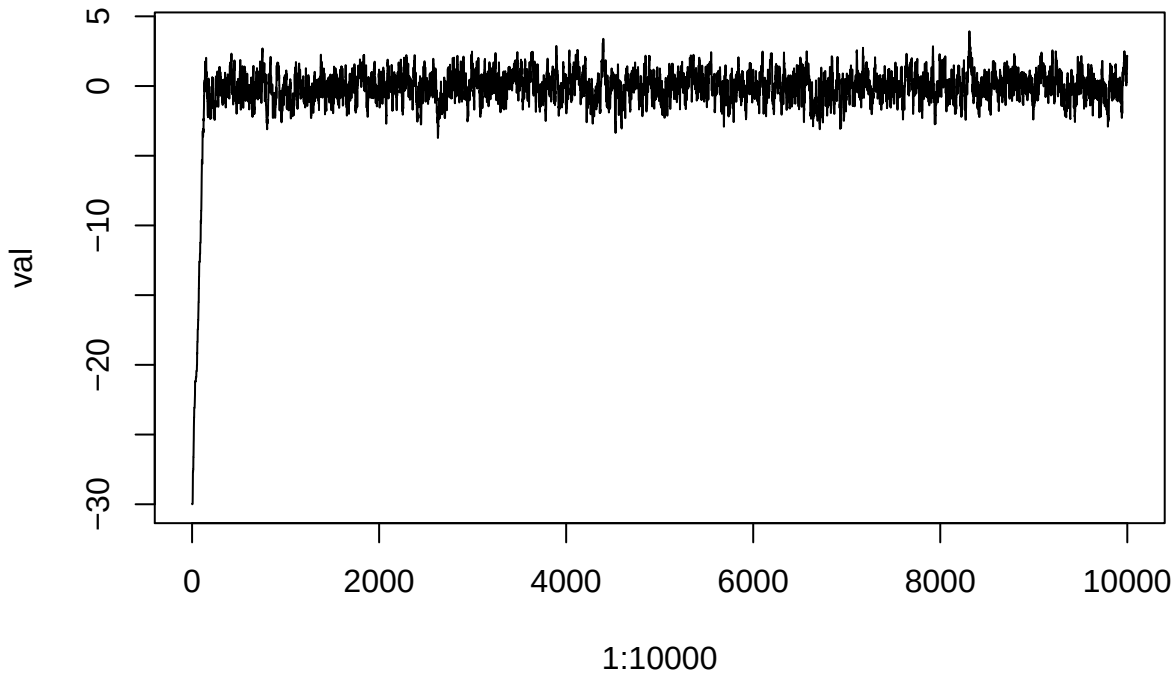
I also use the MCMC and the Metropolis-Hastings algorithm to sample from the normal distribution.

```
mh_norm <- function(start = -30, n = 10000) {
  # target
  p <- function(x){
    dnorm(x)
  }

  propose <- function(xt) {
    runif(1, xt - 1, xt + 1) # proposal distribution
  }

  values <- rep(NA, n)

  values[1] <- start
  a = 0
  for(i in 2:n) {
    y <- runif(1, values[i-1]-1, values[i-1]+1)
    pmove <- min(1, (p(y)/p(values[i-1]))) #* (propose(values[i-1]) / propose(y)))
    u <- runif(1)
    if(u < pmove) {
      values[i] <- y
      a <- a + 1
    }
    else {
      values[i] <- values[i-1]
    }

  }
  return(list(values, a))
}
```

```
its <- 10000
v <- mh_norm(n = its)
val <- v[[1]]
a <- v[[2]]
cat("Acceptance Rate:", a/its, "\n")
```

```
## Acceptance Rate: 0.7947
```

```
plot(1:10000, val, type = "l")
```




```
index<- 175:10000
hist(val[index], prob = TRUE, ylim = c(0, max(density(val[index])$y)))
lines(density(val[index]), col = "blue", lwd = 3.5)
```