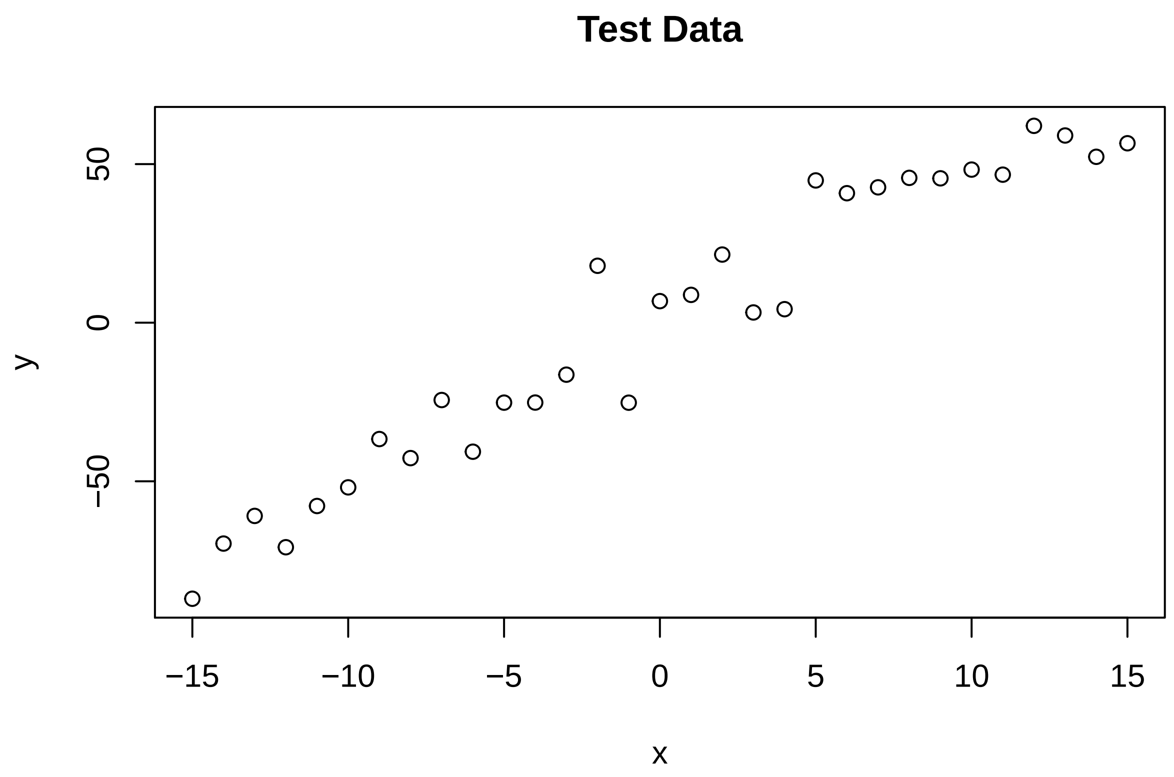
Part 1

Jaymo Kim

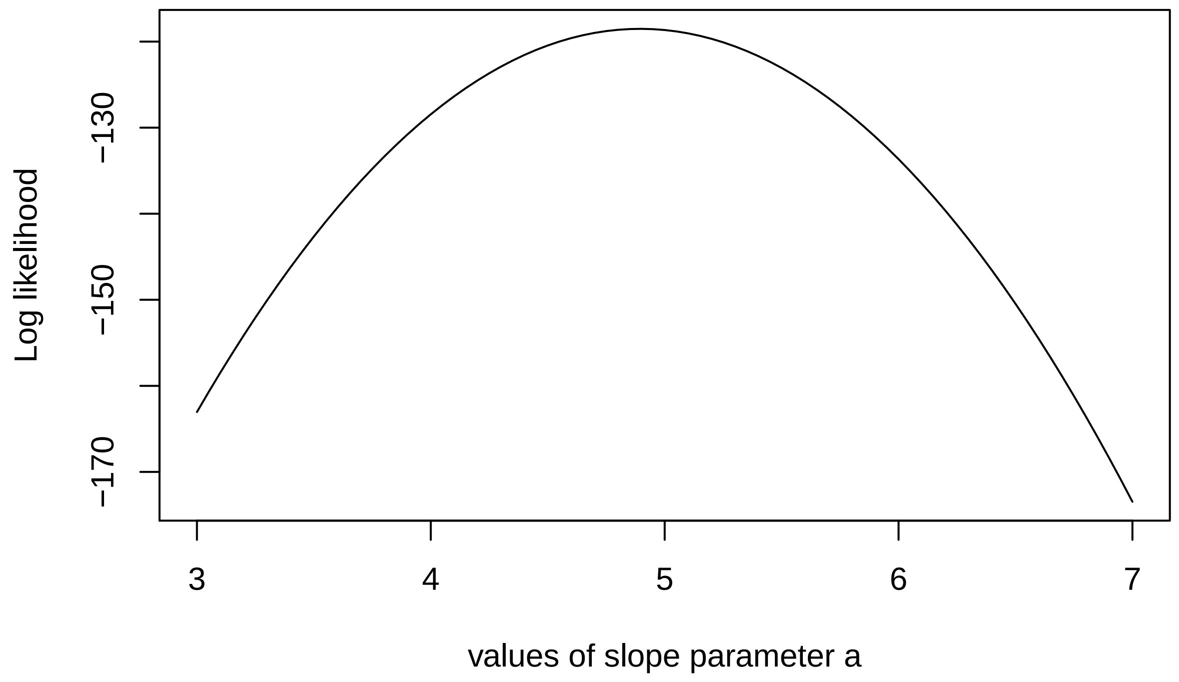
```r
source("likelihood.R")
source("slopevalues.R")
source("prior.R")
source("posterior.R")
source("proposalfunction.R")
source("run_metropolis_MCMC.R")
source("summaryfunc.R")
```

```r
# Creating Test Data
trueA <- 5    # slope
trueB <- 0    # intercept
trueSd <- 10    # true sd
sampleSize <- 31    # sample size
```

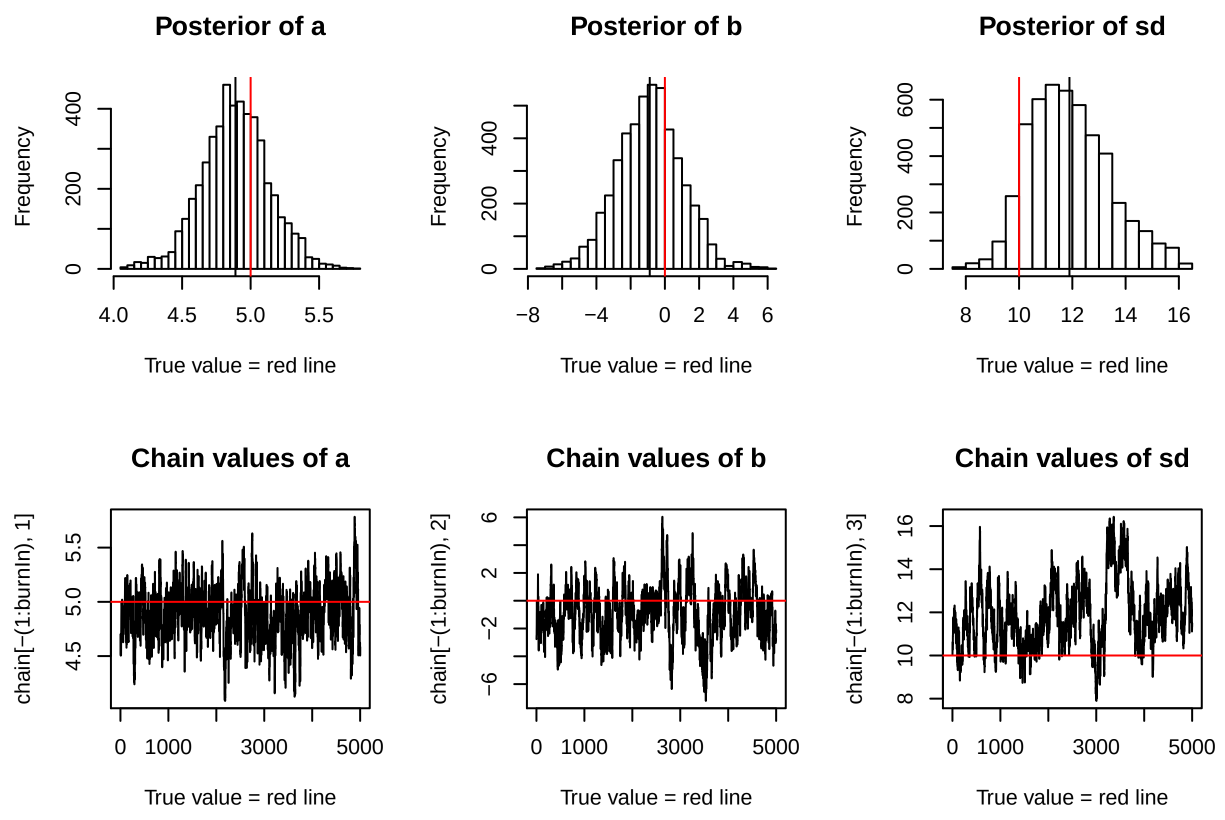
```r
# create independent x-values
# x-values with a size of a sample size
x <- (-(sampleSize-1)/2):((sampleSize-1)/2)

# create dependent values according to ax + b + N(0,sd)
# y-values according to the model
y <-  trueA * x + trueB + rnorm(n=sampleSize,mean=0,sd=trueSd)

# A scatterplot of the created x-values and dependent values according to the model.
plot(x,y, main="Test Data")
```



```
# Example: plot the likelihood profile of the slope a
# a sum of likelihoods for different values of slope
slopelikelihoods <- lapply(seq(3, 7, by=.05), slopevalues )

# plot of the likelihood profile of the slope a
plot (seq(3, 7, by=.05), slopelikelihoods , type="l",
      xlab = "values of slope parameter a", ylab = "Log likelihood")
```



```r
# setting the start value
startvalue = c(4,0,10)
# getting the chain by starting with the startvalue and iterating for 10000 times
chain = run_metropolis_MCMC(startvalue, 10000)

burnIn = 5000    # number of burning iterations
acceptance = 1-mean(duplicated(chain[-(1:burnIn),]))    # Accepance rate

summaryfunc(chain = chain, burnIn = burnIn, trueA = trueA, trueB = trueB, trueSd = trueSd)
```



```r
# for comparison:
summary(lm(y~x))    # comparing the methodf of MCMC with the method of linear regression
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -19.437  -8.350  -0.223   7.546  28.622
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.8891     2.0303  -0.438    0.665
## x             4.8949     0.2270  21.565   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.3 on 29 degrees of freedom
## Multiple R-squared:  0.9413, Adjusted R-squared:  0.9393
## F-statistic:   465 on 1 and 29 DF,  p-value: < 2.2e-16
```

```r
set.seed(1023)
# compare_outcomes function
compare_outcomes <- function(n) {
    alpha = runif(10, min=0, max=10)
    beta = rnorm(10, sd = 5)
    sigma = runif(10, min=0, max=30)
    for(i in 1:10) {
        ch <- run_metropolis_MCMC(startvalue = c(alpha[i], beta[i], sigma[i]),
                                  iterations = n)
        print(c(mean(ch[, 1]), sd(ch[, 1])))
    }
}
# mean and sd for a, iteration(n) = 1,000
compare_outcomes(n = 1000)
```

```
## [1] 4.7523238 0.4569491
## [1] 4.9271106 0.4110873
## [1] 5.258859 1.151076
## [1] 4.9525752 0.2549839
## [1] 4.7424358 0.4327343
## [1] 4.8054678 0.2441665
## [1] 5.1952298 0.8067238
## [1] 4.5486747 0.8543691
## [1] 4.8211054 0.4797164
## [1] 4.8325774 0.6599564
```

```r
# mean and sd for a, iteration(n) = 10,000
compare_outcomes(n = 10000)
```

```
## [1] 4.8651261 0.2753535
## [1] 4.9317013 0.3108263
## [1] 4.8895911 0.2377609
## [1] 4.8773666 0.3242463
## [1] 4.875426 0.295292
## [1] 4.9803413 0.5931446
## [1] 4.8971354 0.2863823
## [1] 4.8867130 0.2481042
## [1] 4.849878 0.430257
## [1] 4.8909778 0.2719273
```

```r
# mean and sd for a, iteration(n) = 100,000
compare_outcomes(n = 100000)
```

```
## [1] 4.8911849 0.2398706
## [1] 4.8937391 0.2364651
## [1] 4.8889424 0.2569233
## [1] 4.8948053 0.2408847
## [1] 4.8917427 0.2480971
## [1] 4.8913192 0.2337502
## [1] 4.8889404 0.2390057
## [1] 4.8953345 0.2473005
## [1] 4.8869015 0.2538829
## [1] 4.8988112 0.2477351
```

Based on the outcome of compare_outcome function, we can observe that there are subtle differences in accuracy of this algorithm in finding a for different number of iterations.

For the outcome of this algorithm with iteration = 1,000, the mean and the standard deviation of a fluctuate: the mean of a ranges from 4.43 to 5.17 and the standard deviation of a ranges from 0.20 to 1.18.

For iteration = 10,000, the mean and the standard deviation of a fluctuate less. The mean of a ranges from 4.73 to 4.86 and the standard deviation of a ranges from 0.21 to 0.59.

Lastly, for iteration = 100,000, the mean and the standard deviation of a is most consistent. The mean of a only ranges from 4.76 to 4.77 and the standard deviation of a only ranges from 0.20 to 0.22.

In summary, the algorithm gets more accurate for the mean and the standard deviation of a are more consistent as the number of iteration increases. Therefore, we can conclude that in order for the algorithm to be as accurate, and thus close to the linear regression, as possible, we would need a large number of iteration.