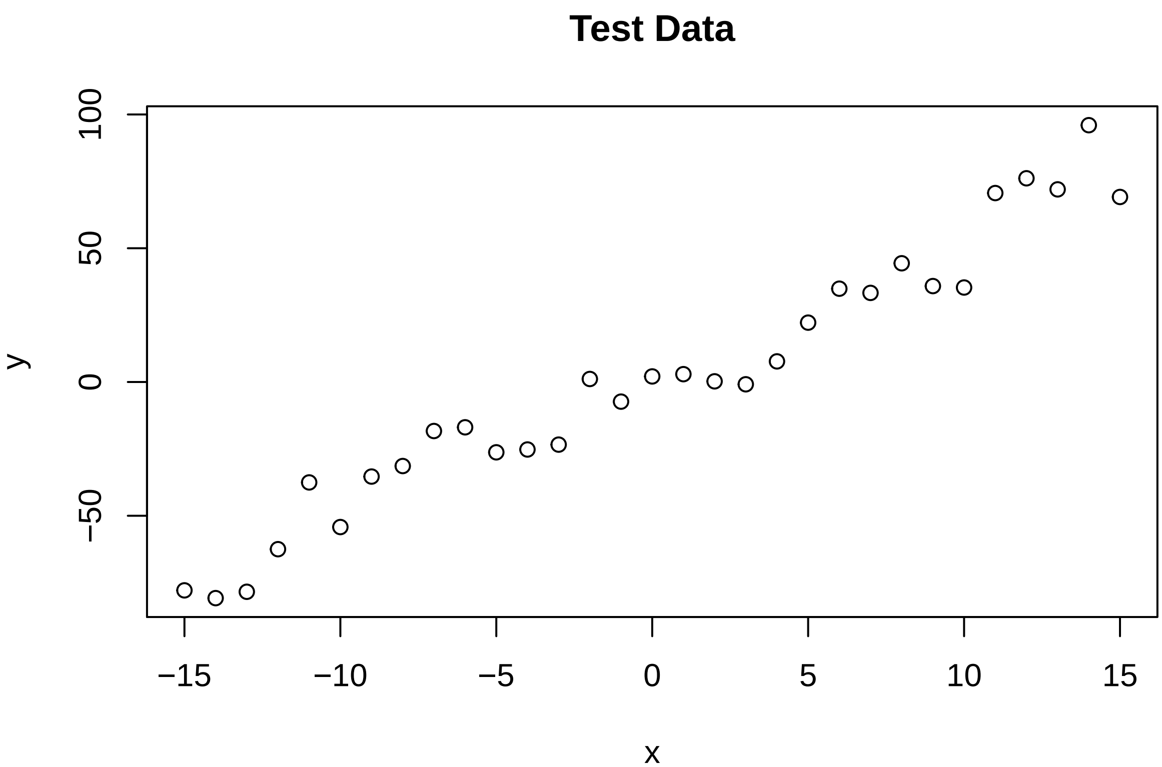
Accuracy Report

Weidi Pan

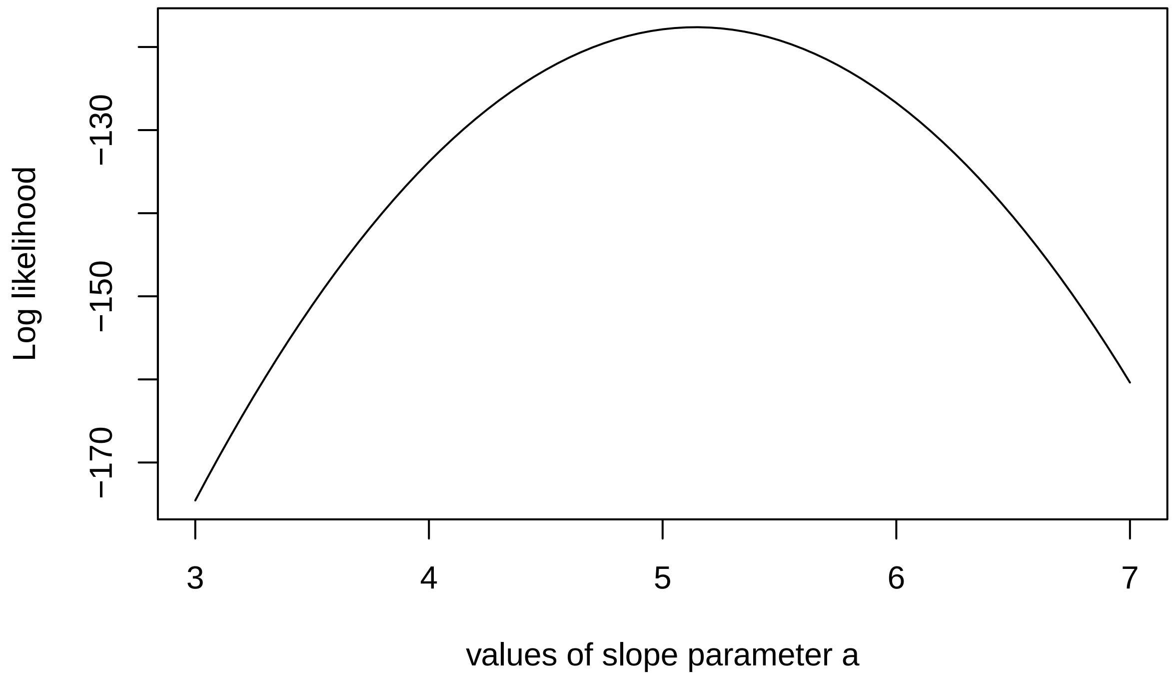
10/22/2018

```r
source("functions.R")

trueA <- 5   #true slope
trueB <- 0    #true intercept
trueSd <- 10    #true standard deviation of error
sampleSize <- 31  #sample size

# create independent x-values
x <- (-(sampleSize-1)/2):((sampleSize-1)/2)  # n=sampleSize
# create dependent values according to ax + b + N(0,sd)
y <-  trueA * x + trueB + rnorm(n=sampleSize,mean=0,sd=trueSd)

plot(x,y, main="Test Data")
```



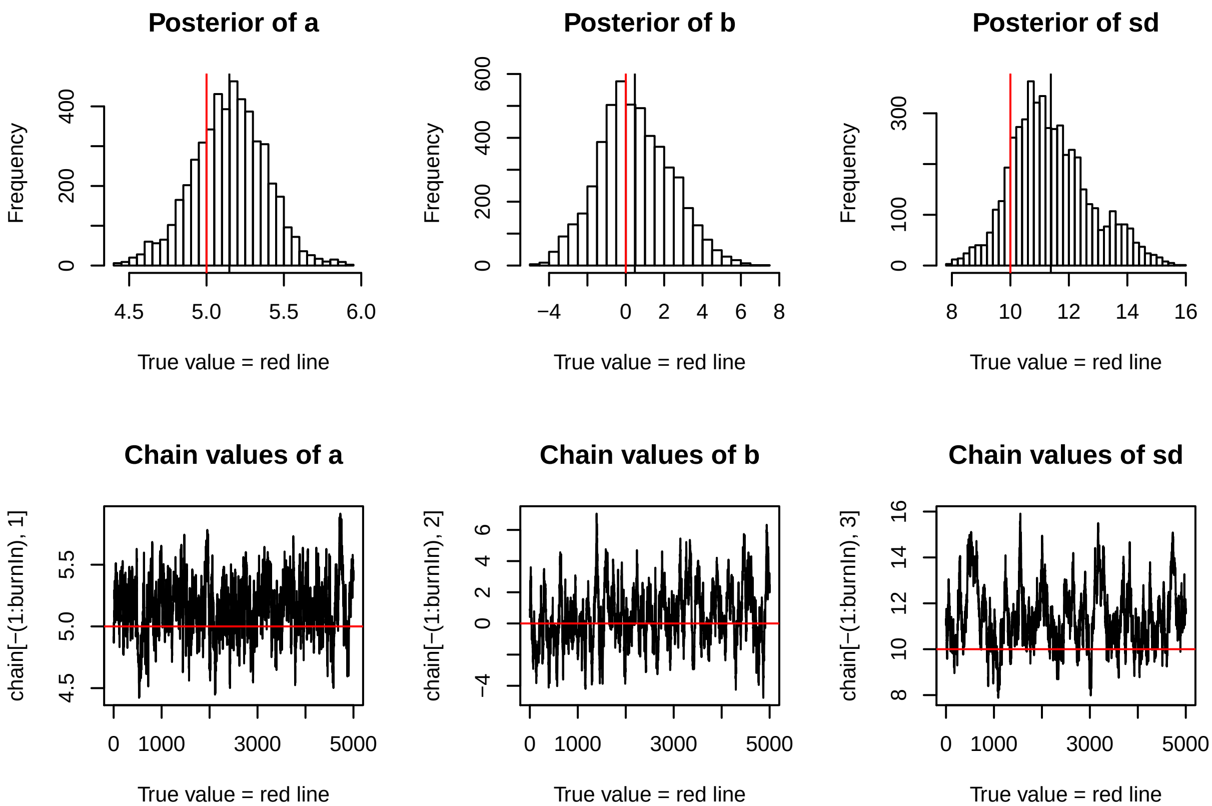

```r
#balanced x values around zero to "de-correlate" slope and intercept
#a scatterplot of test data
```

```r
# Example: plot the likelihood profile of the slope a

slopelikelihoods <- lapply(seq(3, 7, by=.05), slopevalues) #applies the function on a vector of slopes
plot (seq(3, 7, by=.05), slopelikelihoods , type="l", xlab = "values of slope parameter a", ylab = "Log
```




```
#plot log likelihoods against different values of slope parameter a
```

```
######## Metropolis algorithm ################

startvalue = c(4,0,10)  #starting value of the procedure
chain = run_metropolis_MCMC(startvalue, 10000) #inputs startvalue and number of iterations, outputs mat

burnIn = 5000
#The first steps of the algorithm may be biased by the initial value,
#and are therefore usually discarded for the further analysis (burn-in time)

acceptance = 1-mean(duplicated(chain[-(1:burnIn),]))  # acceptance rate

### Summary: #######################

graph_summary(chain,burnIn,trueA,trueB,trueSd)
```




```r
# for comparison:
summary(lm(y~x))
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -17.169  -8.864  -1.667   9.449  23.104
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.8846     1.9803   0.447    0.658
## x             5.1430     0.2214  23.229   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.03 on 29 degrees of freedom
## Multiple R-squared:  0.949,  Adjusted R-squared:  0.9472
## F-statistic: 539.6 on 1 and 29 DF,  p-value: < 2.2e-16
```

```r
#compare this procedure MCMC with linear regression
```

```r
source("functions.R")
#compare_outcomes, that takes as input an iteration number
#for instance 1000, 10000, or 100000. Your function should loop 10 times; each
#time, it should initialize the MCMC chain with randomly selected start values
#for a, b, and sd and then run to completion after the required number of
```

```r
#iterations. After each loop, the function should compute the mean and std
#of the values in the chain for a, and it should print these out. Test your
#function for 1,000, 10,000, and 100,000 iterations.


compare_outcomes <- function(iterations) {
  for (i in 1:10) {
    a = runif(1, min=0, max=10)
    b = rnorm(1, 0, 5)
    sd = runif(1, min=0, max=30)
    startvalue=c(a,b,sd)
    chain = run_metropolis_MCMC(startvalue,iterations)
    print(c(mean(chain[,1]),sd(chain[,1])))
  }
}

compare_outcomes(1000)
```

```
## [1] 4.9001116 0.7271469
## [1] 5.0374254 0.4500609
## [1] 5.1968695 0.3936781
## [1] 5.3269349 0.8176538
## [1] 5.4014838 0.8390039
## [1] 5.0619177 0.2668544
## [1] 4.987807 0.291718
## [1] 4.9603720 0.7515964
## [1] 5.1491683 0.2466659
## [1] 4.7369434 0.9857535
```

```r
compare_outcomes(10000)
```

```
## [1] 5.170628 0.346492
## [1] 5.1881224 0.4359597
## [1] 5.1543642 0.2726847
## [1] 5.1409152 0.2273817
## [1] 5.1706545 0.3764299
## [1] 5.1534378 0.3296608
## [1] 5.1337848 0.2318604
## [1] 5.1383639 0.2209119
## [1] 5.1631443 0.2889778
## [1] 5.1493873 0.2520079
```

```r
compare_outcomes(100000)
```

```
## [1] 5.1404316 0.2385761
## [1] 5.146323 0.232953
## [1] 5.1408837 0.2438869
## [1] 5.1495599 0.2316727
## [1] 5.1415549 0.2364506
## [1] 5.1358966 0.2411718
## [1] 5.1525921 0.2592225
## [1] 5.1384591 0.2433755
## [1] 5.1457031 0.2410367
## [1] 5.1445782 0.2461923
```

From the outputs above, we can see that as iterations grow larger, the values of mean of A are more stable, and standard deviations become smaller.

n = 1000, mean ranges from 4.60 to 5.16, sd ranges from 0.19 to 1.03 n = 10000, mean ranges from 4.82 to 4.88, sd ranges from 0.19 to 0.44 n = 100000, mean ranges from 4.85 to 4.87, sd ranges from 0.20 to 0.25

Thus, the accuracy of this algorithm in finding A increases as iterations grow; accordingly, the time to process also increases