Lab/HW 3: Lists, Data Frames, Functions

Your lab/homework must be submitted in with two files: (1) R Markdown format file; (2) a pdf or html file. Other formats will not be accepted. Your responses must be supported by both textual explanations and the code you generate to produce your result.

Part I - Lists, apply functions

Our goal in this question is to demonstrate the law of large numbers: Given $X_1, X_2, ...$ i.i.d. random variables with expected value μ, the law of large number states that

$$\frac{1}{n}\sum_{i=1}^{n} X_i \to \mu$$

when $n \to \infty$. That is, the average of the observations converges to the expected value.

[7 pts] 1. Create a sequence of 500 integers from 1 to 500. This is an atomic vector. However, lists are also vectors, so we can think of this sequence as a list. We have seen that the function `lapply` takes lists as inputs. Use the `lapply` function on the above sequence to generate a list of length 500, in which the i^{th} element contains i draws from a independent standard normal random variables.

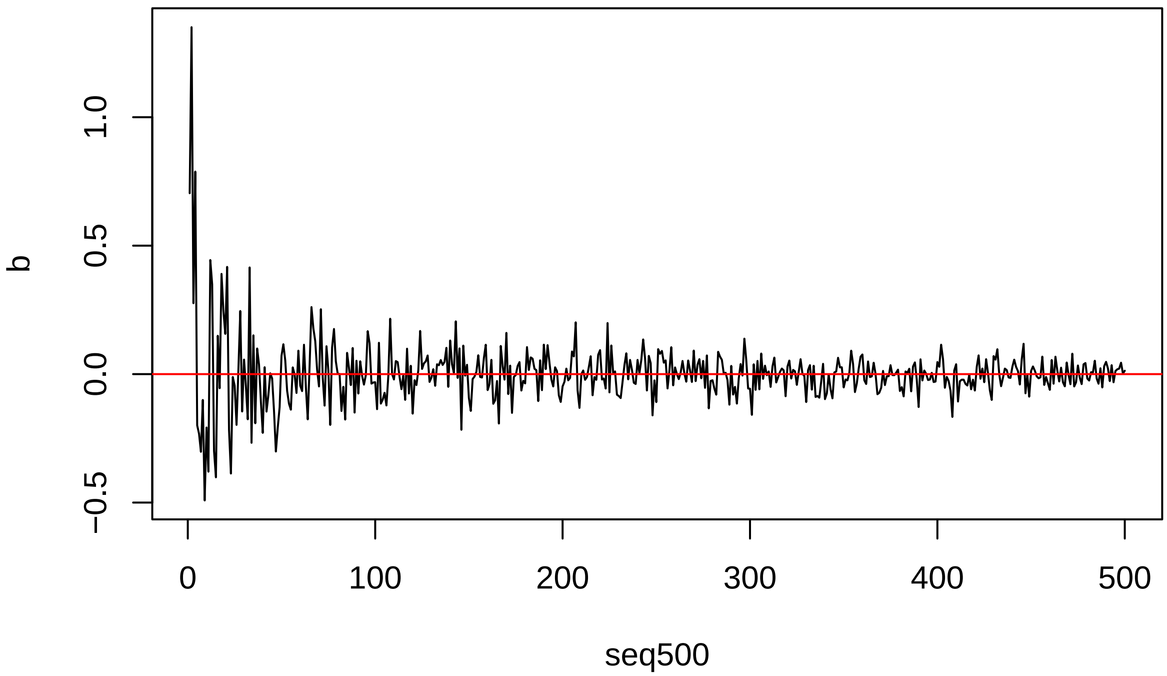
[7 pts] 2. Using a single command from the `apply` family generate a vector of length 500 in which the i^{th} element is the mean of the draws from the i^{th} element of the list you generated above. So, for instance, the 101th element in the list from section 3a has 101 random draws from a normal random variable. The 101th element in the vector you generate here will have the mean of these 101 draws.

In order to get full credit you must, in 1 and 2, explain the type of inputs, outputs, and what the functions you use do in your own words

[6 pts] 3. Plot the vector of means vs. the number of draws. Find in `?plot` how to make this a line plot. Use the following command `abline(h = 0, col = "red")` to add to the plot a horizontal line at the expected value of the standard normal random variables.

Solution

```
seq500 = 1:500 # 3a
# ":"(1, 500)  # alternatively
a = lapply(seq500, rnorm) # 3a
b = sapply(a, mean) # 3b
plot(seq500, b, type = "l")  # 3c
abline(h = 0, col = "red")
```



3a: The operator `:` (operators are functions) takes the double inputs 1 and 500 and generates an integer vector of length 500, whose entries are 1,...500. The function `lapply` takes two inputs: `seq500`, a vector that is treated as a list, and `rnorm` a function to use on the list elements. And so it generates a list whose i^{th} element is the result of the function `rnorm(i)` which takes as input an integer and returns as output a double vector of length i where each element is an independent draw from a standard normal.

3b: `sapply` takes the list from 4a as input, and applies for each of it elements (a double vector) the function `mean()`. the result is a list of length 500, where each element is a double scalar. `sapply` then simplifies this list into a double vector of length 500.

Part II – Data Frames

R includes a number of pre-specified data objects as part of its default installation. We will load and manipulate one of these, a data frame of 93 cars with model year 1993. Begin by ensuring that you can load this data with the commands

```
library(MASS)
data(Cars93)
```

Begin by examining the data frame with the command `View(Cars93)` to understand the underlying object. You will need to use functions and other commands to extract elements for this assignment.

[5 pts] 1. Obtain a `summary()` of the full data structure. Can you tell from this how many rows are in the data? If so, say how; if not, use another method to obtain the number of rows.

[5 pts] 2. What is the mean price of a car with a rear-wheel drive train?

[5 pts] 3. What is the minimum horsepower of all cars with capacity for 7 passengers? With a capacity of at least 6 passengers?

[5 pts] 4. Assuming that these cars are exactly as fuel efficient as this table indicates, find the cars that have the maximum, minimum and median distance one can travel for highway driving. You will need at least two columns to work this out; why those two?

```r
library(MASS)
data(Cars93)
```

View(Cars93) does not work in Rmarkdown, however, the command can be run in the console and when it is another window tab opens and displays the data.

Solutions

```r
summary(Cars93)
```

```
##    Manufacturer     Model         Type      Min.Price         Price      
##  Chevrolet: 8    100    : 1   Compact:16   Min.   : 6.70   Min.   : 7.40  
##  Ford     : 8    190E   : 1   Large  :11   1st Qu.:10.80   1st Qu.:12.20  
##  Dodge    : 6    240    : 1   Midsize:22   Median :14.70   Median :17.70  
##  Mazda    : 5    300E   : 1   Small  :21   Mean   :17.13   Mean   :19.51  
##  Pontiac  : 5    323    : 1   Sporty :14   3rd Qu.:20.30   3rd Qu.:23.30  
##  Buick    : 4    535i   : 1   Van    : 9   Max.   :45.40   Max.   :61.90  
##  (Other)  :57    (Other):87                                               
##    Max.Price       MPG.city      MPG.highway                  AirBags  
##  Min.   : 7.9   Min.   :15.00   Min.   :20.00   Driver & Passenger:16  
##  1st Qu.:14.7   1st Qu.:18.00   1st Qu.:26.00   Driver only       :43  
##  Median :19.6   Median :21.00   Median :28.00   None              :34  
##  Mean   :21.9   Mean   :22.37   Mean   :29.09                          
##  3rd Qu.:25.3   3rd Qu.:25.00   3rd Qu.:31.00                          
##  Max.   :80.0   Max.   :46.00   Max.   :50.00                          
##                                                                        
##  DriveTrain  Cylinders    EngineSize      Horsepower         RPM      
##  4WD  :10   3      : 3   Min.   :1.000   Min.   : 55.0   Min.   :3800  
##  Front:67   4      :49   1st Qu.:1.800   1st Qu.:103.0   1st Qu.:4800  
##  Rear :16   5      : 2   Median :2.400   Median :140.0   Median :5200  
##             6      :31   Mean   :2.668   Mean   :143.8   Mean   :5281  
##             8      : 7   3rd Qu.:3.300   3rd Qu.:170.0   3rd Qu.:5750  
##             rotary: 1   Max.   :5.700   Max.   :300.0   Max.   :6500  
##                                                                       
##   Rev.per.mile  Man.trans.avail Fuel.tank.capacity   Passengers   
##  Min.   :1320   No :32          Min.   : 9.20      Min.   :2.000  
##  1st Qu.:1985   Yes:61          1st Qu.:14.50      1st Qu.:4.000  
##  Median :2340                   Median :16.40      Median :5.000  
##  Mean   :2332                   Mean   :16.66      Mean   :5.086  
##  3rd Qu.:2565                   3rd Qu.:18.80      3rd Qu.:6.000  
##  Max.   :3755                   Max.   :27.00      Max.   :8.000  
##                                                                   
##      Length        Wheelbase         Width        Turn.circle   
##  Min.   :141.0   Min.   : 90.0   Min.   :60.00   Min.   :32.00  
##  1st Qu.:174.0   1st Qu.: 98.0   1st Qu.:67.00   1st Qu.:37.00  
##  Median :183.0   Median :103.0   Median :69.00   Median :39.00  
##  Mean   :183.2   Mean   :103.9   Mean   :69.38   Mean   :38.96  
##  3rd Qu.:192.0   3rd Qu.:110.0   3rd Qu.:72.00   3rd Qu.:41.00  
##  Max.   :219.0   Max.   :119.0   Max.   :78.00   Max.   :45.00  
##                                                                 
##  Rear.seat.room   Luggage.room       Weight         Origin              Make   
##  Min.   :19.00   Min.   : 6.00   Min.   :1695   USA    :48   Acura Integra: 1  
##  1st Qu.:26.00   1st Qu.:12.00   1st Qu.:2620   non-USA:45   Acura Legend : 1  
##  Median :27.50   Median :14.00   Median :3040                Audi 100     : 1  
##  Mean   :27.83   Mean   :13.89   Mean   :3073                Audi 90      : 1  
```

```
##  3rd Qu.:30.00   3rd Qu.:15.00   3rd Qu.:3525                    BMW 535i      : 1
##  Max.   :36.00   Max.   :22.00   Max.   :4105                    Buick Century: 1
##  NA's   :2       NA's   :11                                      (Other)      :87
```

One could figure out how many rows there are by summing up values for certain columns such as Manufacturer or origin, or by using the `dim` function:

```
numb.row = dim(Cars93)[1]
cat("There are", numb.row, "rows")
```

```
## There are 93 rows
```

```
RWD.mean.cost = mean(Cars93$Price[Cars93$DriveTrain =='Rear'])
cat('The average cost of cars with rear wheel drive is', RWD.mean.cost)
```

```
## The average cost of cars with rear wheel drive is 28.95
```

```
hp.capacity.7 = min(Cars93$Horsepower[Cars93$Passengers >= 7])

cat("The minimum horsepower of all cars with capacity for 7 passengers is",hp.capacity.7)
```

```
## The minimum horsepower of all cars with capacity for 7 passengers is 109
```

```
hp.capacity.6 = min(Cars93$Horsepower[Cars93$Passengers >= 6])

cat("The minimum horsepower of all cars with capacity for 6 passengers is",hp.capacity.6)
```

```
## The minimum horsepower of all cars with capacity for 6 passengers is 100
```

Question 4

```
Cars93$highway.dist = Cars93$MPG.highway*Cars93$Fuel.tank.capacity

max.highway.dist = max(Cars93$highway.dist)
min.highway.dist = min(Cars93$highway.dist)
med.highway.dist = median(Cars93$highway.dist)


max.highway.cars = as.character(Cars93$Make[Cars93$highway.dist == max.highway.dist])
min.highway.cars = as.character(Cars93$Make[Cars93$highway.dist == min.highway.dist])
med.highway.cars = as.character(Cars93$Make[Cars93$highway.dist == med.highway.dist])

cat('The car(s) with the best milage on the highway is/are', max.highway.cars)
```

```
## The car(s) with the best milage on the highway is/are BMW 535i
```

```
cat('The car(s) with the worst milage on the highway is/are', min.highway.cars)
```

```
## The car(s) with the worst milage on the highway is/are Mercury Capri
```

```
cat('The car(s) with the median milage on the highway is/are', med.highway.cars)
```

```
## The car(s) with the median milage on the highway is/are Mazda MPV
```

The two columns used are the MPG.highway Fuel.tank.capacity because MPG.highway*Fuel.tank.capacity gives us the total amount of miles that can be driven on the highway.

Part III – Functions

We have seen in HW 2 that our method for sampling from the truncated normal random variable is problematic because we cannot fully control the number of observations. In this part, we will develop two functions that return a sample of truncated normal draws of a given length.

[20 pts] 1.

[15 pts] a. Write a function that uses a while/repeat loop to generate draws from a truncated normal random variable one by one till we get the exact number needed. I.e., use the `rnorm(1, mu, sd)` function to generate samples one by one. Each sample that is in the desired range for the truncated normal will be saved, all others discarded till we have enough samples. Stop generating samples when you have enough. Return that sample.

[5 pts] b. As a sanity check, generate a sample of 10000 observations with `a = 0.5`, `b = 3`, `mu = -0.1`, `sigma = 1.1`. Verify that the result is of the correct length. Draw a histogram of this sample and add to it a line showing the theoretical density for this distribution.

[25 pts] 2.

[20 pts] a. Given `mu`, `sigma`, a lower bound `a`, an upper bound `b` for the range of truncation and `N` draws from a normal random variable, `n` = the expected number of observations for a truncated normal sample that are filtered from an appropriate normal sample of length `N` is

$$n = N \cdot (\Phi(b) - \Phi(a))$$

Now, n is known - the number of truncated normal draws we wish to generate. Based on n, choose N. In your function, generate a sample of truncated normal draws filtered from `rnorm(N, ...)`. If the length of the sample you generated is too long, cut it to the needed length. If it is too short, call the function from 1 to generate samples according to how many are missing to complete n samples. Add the newly generated samples to the lacking sample and return that vector.

[5 pts] b. As a sanity check, generate a sample of 10000 observations with `a = 0.5`, `b = 3`, `mu = -0.1`, `sigma = 1.1`. Verify that the result is of the correct length. Draw a histogram of this sample and add to it a line showing the theoretical density for this distribution. Do you expect the vector you generated here to be "the same" as the one you generated in 1? (the answer can be point-wise, distributional or any other sense you can think of).

[5 pt + 10 pt Extra Credit] 3.

[5 pts] a. Time the two versions and the `truncnorm::rtruncnorm()` function (that generates a sample of truncated normal draws) with `microbencmark`. Which is the fastest? Which the slowest? By how much? (compare median execution time to answer all questions) Try different settings (lengths of vectors, parameters for truncation)

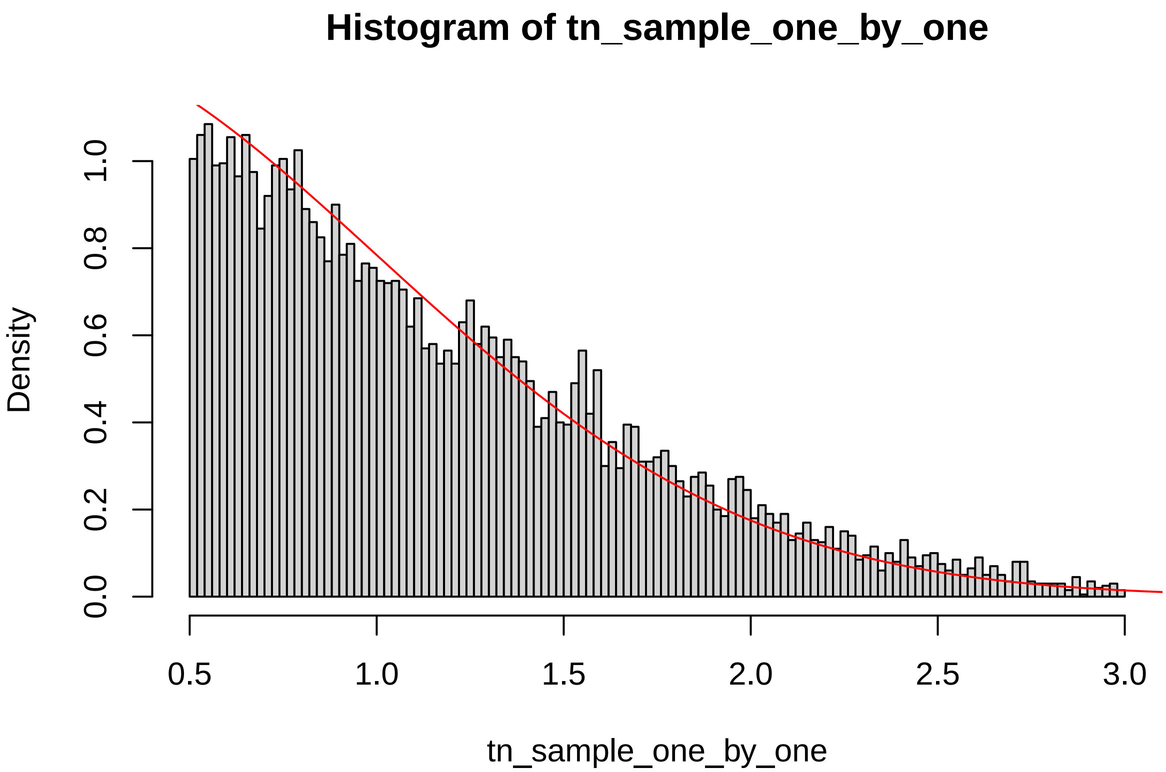
[10 pt Extra Credit] b. write code that examines how the scaling differs between the two approaches. That is, loop over the comparison from 3a for different values of `n` and store the results in an appropriate data type. Chart on a single plot the execution time (in milliseconds) of each of the three methods as a function of `n`. Another useful method for comparison of complexity is drawing the above plot on a log-log scale. Explain what you see and what lessons you learn from each of the plots (what do the shapes of lines or slopes, what do the absolute magnitudes mean in the linear-linear scale? what do they mean in the log-log scale?)

```
sample_truncated_normal_one_by_one <-
  function(n, mu = 0, sigma = 1, a = -Inf, b = Inf) {
    i <- 1;
    trunc_s <- numeric(n)
    while (i <= n) {
       s <- rnorm(1, mean = mu, sd = sigma)
       if (s >= a & s <= b) {
         trunc_s[i] <- s
```

```r
        i <- i + 1
      }
    }
    return(trunc_s)
  }

tn_sample_one_by_one <-
  sample_truncated_normal_one_by_one(10000, mu = -0.1, sigma = 1.1, a = 0.5, b = 3)
length(tn_sample_one_by_one)
```

```
## [1] 10000
```

```r
hist(tn_sample_one_by_one, breaks = 100, freq = FALSE)
lines(
  seq(0.5, 4, 0.01),
  truncnorm::dtruncnorm(seq(0.5, 4, 0.01), a = 0.5),
  col = "red"
  )
```

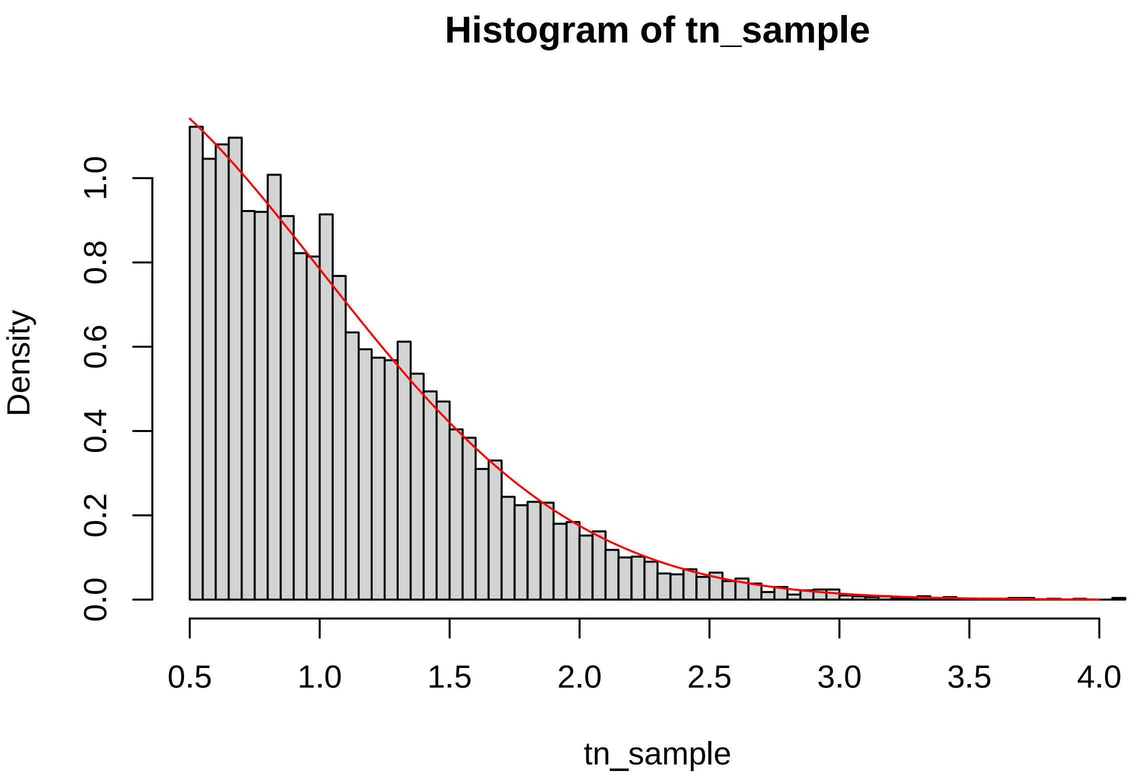



```r
sample_truncated_normal_batch <- function(n, mu = 0, sigma = 1, a = -Inf, b = Inf) {
    N <- floor(n / (pnorm(b) - pnorm(a)))
    s <- rnorm(N, mean = mu, sd = sigma)
    trunc_s_temp <- s[s > a & s < b]

    n_temp <- length(trunc_s_temp)
    if (n_temp >= n) {
      trunc_s <- trunc_s_temp[1:n]
    } else {
      extra_samples <-
        sample_truncated_normal_one_by_one(
```

```r
          n = n_temp,
          mu = mu,
          sigma = sigma,
          a = a,
          b = b
          )
      trunc_s <- c(trunc_s_temp, extra_samples)
    }
  return(trunc_s)
}

tn_sample <- sample_truncated_normal_batch(10000, a = 0.5)
length(tn_sample)
```

```
## [1] 10000
```

```r
hist(tn_sample, breaks = 100, freq = FALSE)
lines(seq(0.5, 4, 0.01), truncnorm::dtruncnorm(seq(0.5, 4, 0.01), a = 0.5), col = "red")
```




```r
n <- 100
a <- 0.5
compare_times <-
  microbenchmark::microbenchmark(
    {
      t1 <- sample_truncated_normal_one_by_one(n, a = a)
      stopifnot
    }, {
      t2 <- sample_truncated_normal_batch(n, a = a)
    }, {
      t3 <- truncnorm::rtruncnorm(n, a = a)
```

```r
    }
    )
summary_compare_times <- summary(compare_times)
summary_compare_times
```

```
##                                                                          expr
## 1 {     t1 <- sample_truncated_normal_one_by_one(n, a = a)     stopifnot }
## 2                     {     t2 <- sample_truncated_normal_batch(n, a = a) }
## 3                             {     t3 <- truncnorm::rtruncnorm(n, a = a) }
##       min       lq      mean   median      uq      max neval cld
## 1 454.506 527.7385 651.84521 578.3185 657.398 4495.505   100  a
## 2  25.463  27.5665 124.48902  36.7895  64.576 7485.907   100   b
## 3  17.510  20.0055  24.34887  21.2915  24.344   81.764   100   b
```

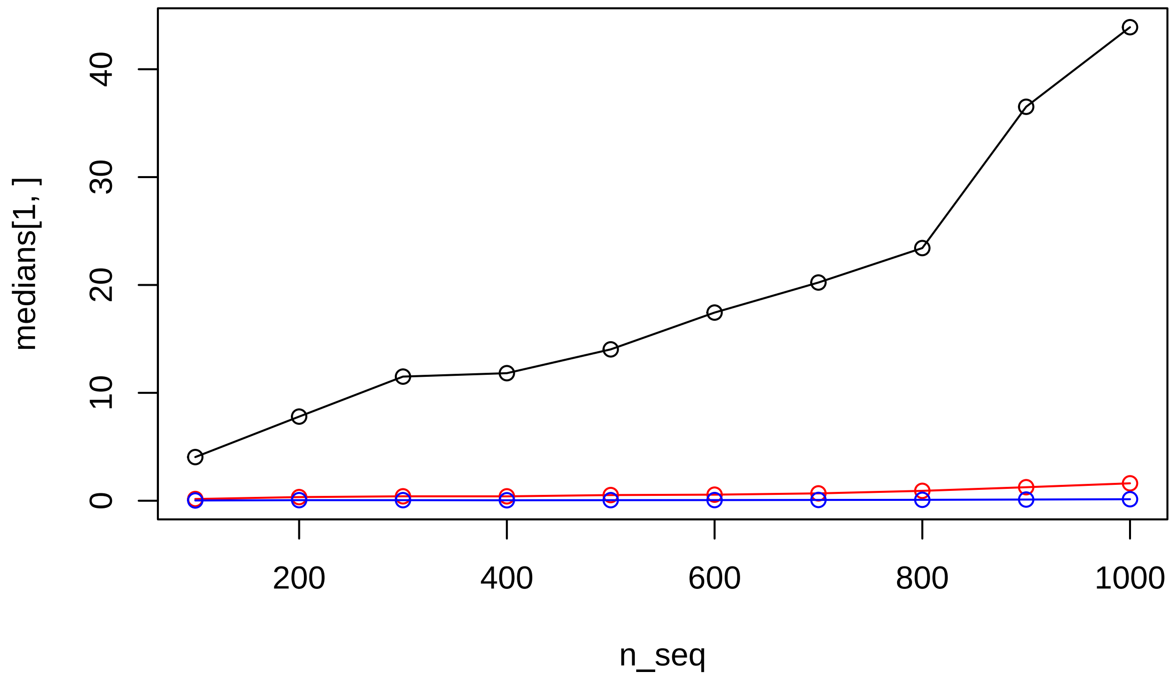
```r
summary_compare_times$median[1] / summary_compare_times$median[2]
```

```
## [1] 15.71966
```

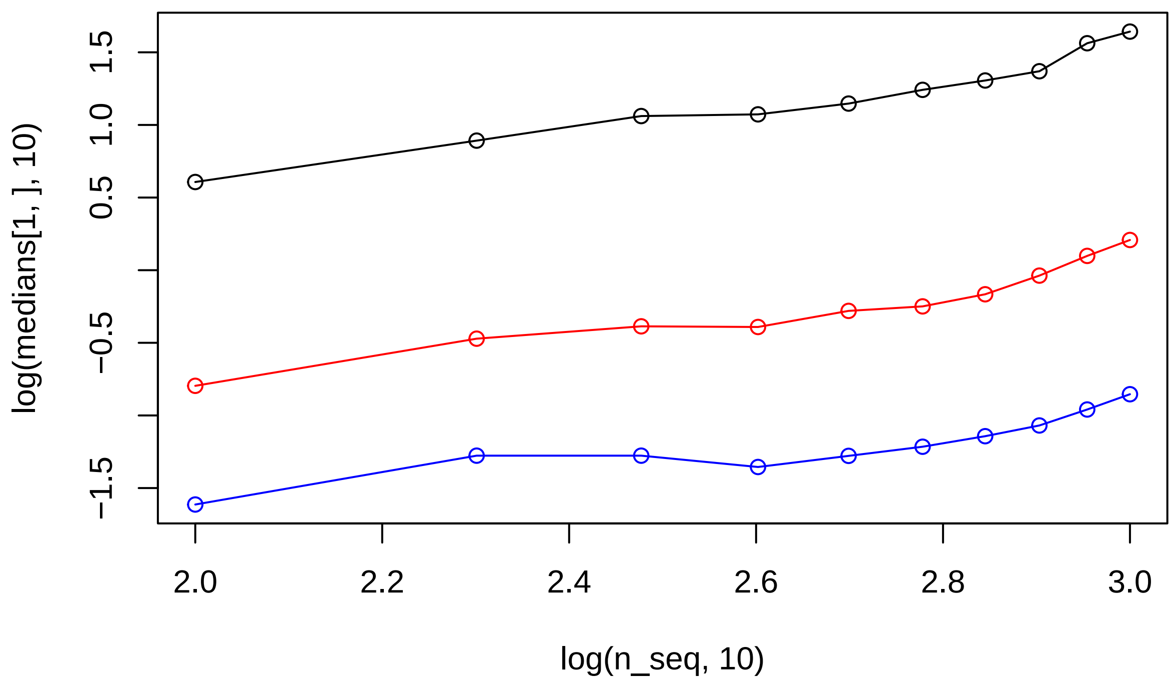
```r
summary_compare_times$median[2] / summary_compare_times$median[3]
```

```
## [1] 1.727896
```

```r
n_seq <- seq(100, 1000, 100)
a <- 1.5
b <- Inf
mu <- 0
sigma <- 1

medians <- matrix(0, nrow = 3, ncol = length(n_seq))
for (i in 1:length(n_seq)) {
  compare_times <-
    microbenchmark::microbenchmark(
      {
        t1 <- sample_truncated_normal_one_by_one(n_seq[i], a = a)
        stopifnot
      }, {
        t2 <- sample_truncated_normal_batch(n_seq[i], a = a)
      }, {
        t3 <- truncnorm::rtruncnorm(n_seq[i], a = a)
      },
      unit = "ms"
    )
  medians[ , i] <- summary(compare_times)$median
}
plot(n_seq, medians[1, ], ylim = range(medians))
lines(n_seq, medians[1, ])
points(n_seq, medians[2, ], col = "red")
lines(n_seq, medians[2, ], col = "red")
points(n_seq, medians[3, ], col = "blue")
lines(n_seq, medians[3, ], col = "blue")
```




```
plot(log(n_seq, 10), log(medians[1, ], 10), ylim = range(log(medians, 10)))
lines(log(n_seq, 10), log(medians[1, ], 10))
points(log(n_seq, 10), log(medians[2, ], 10), col = "red")
lines(log(n_seq, 10), log(medians[2, ], 10), col = "red")
points(log(n_seq, 10), log(medians[3, ], 10), col = "blue")
lines(log(n_seq, 10), log(medians[3, ], 10), col = "blue")
```




```
lm(log(medians[1, ], 10) ~ log(n_seq, 10))
```

```
##
## Call:
```

```
## lm(formula = log(medians[1, ], 10) ~ log(n_seq, 10))
##
## Coefficients:
##    (Intercept)  log(n_seq, 10)
##        -1.2997          0.9375
```

```r
lm(log(medians[2, ], 10) ~ log(n_seq, 10))
```

```
##
## Call:
## lm(formula = log(medians[2, ], 10) ~ log(n_seq, 10))
##
## Coefficients:
##    (Intercept)  log(n_seq, 10)
##        -2.5695          0.8744
```

```r
lm(log(medians[3, ], 10) ~ log(n_seq, 10))
```

```
##
## Call:
## lm(formula = log(medians[3, ], 10) ~ log(n_seq, 10))
##
## Coefficients:
##    (Intercept)  log(n_seq, 10)
##        -2.8034          0.6022
```

The linear-linear scale demonstrates best the difference in absolute time. The log-log scale provides us with an estimate of the complexity: straight lines in log-log scale correspond to polynomial complexity (that is, the time for execution is a polynomial function of the sample size). Higher slope means higher order polynomial. Although the slopes are similar, the estimated slope of the "one-by-one" method is a bit steeper, suggesting that it really scales poorly compared to the two vectorized methods.