MAST20005/MAST90058: Week 2 Lab Solutions

1.
   ```
   sum(log(1:100))

   ## [1] 363.7394
   ```

2.
   ```
   x <- rnorm(100000, 1, sqrt(2))
   mean(x^2)

   ## [1] 3.000038
   ```

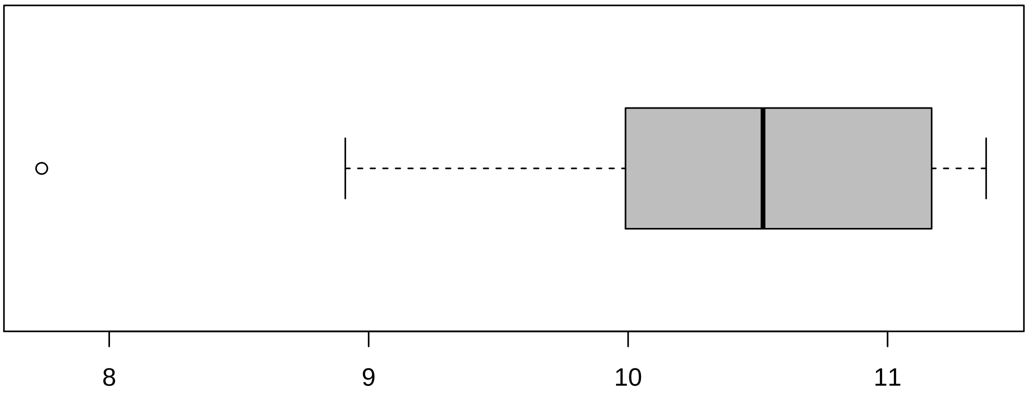
3. Run `help(qnorm)` to access the documentation. `qnorm(0.1)` calculates $\pi_{0.1}$ for a standard normal distribution. `qnorm(0.1, lower.tail = FALSE)` calculates $\pi_{1-0.1} = \pi_{0.9}$, and therefore gives the same answer as `qnorm(0.9)`.

4.
   ```
   exp1pdf <- function(x) {
       if (x < 0)
           d <- 0
       else
           d <- exp(-x)
       return(d)
   }
   exp1pdf(-1)  # should equal zero

   ## [1] 0
   ```

   Note that this version of the function is not vectorised but the in-built version is. Can you write a vectorised version? Hint: try using the `ifelse` function.

5.
   ```
   x <- c(10.39, 10.43, 9.99, 11.17, 8.91,
          11.20, 11.38, 7.74, 10.61, 11.11)
   boxplot(x, col = 8, horizontal = TRUE)  # using the R defaults
   ```



6. See the solutions to the tutorial problems.

7. We need to simulate from the given distribution. First, calculate the cdf,

$$F(x) = \int_{-1}^{x} \frac{3}{2} y^2 \, dy = \left[\frac{1}{2} y^3 \right]_{-1}^{x} = \frac{x^3 + 1}{2}, \quad \text{where } -1 < x < 1.$$

Then invert to get the inverse cdf,

$$F^{-1}(p) = (2p - 1)^{\frac{1}{3}},$$

which we will use to simulate X.

```
# Function to handle powers for negative numbers properly.
exponent <- function(x, p)
    sign(x) * abs(x)^p

# Function to generate random X's.
rx <- function(n)
    exponent(2 * runif(n) - 1, 1/3)

# Function to generate random Y's.
ry <- function(n) {
    y <- 1:n
    for (i in 1:n)
        y[i] <- sum(rx(15))
    return(y)
}
# A more efficient way to do the same thing is:
ry <- function(n)
    replicate(n, sum(rx(15)))

# Simulate Y's.
ys <- ry(10000)

# Estimate the probability.
mean((-0.3 < ys) & (ys < 1.5))

## [1] 0.2277
```

Notes:

- The `exponent` function is needed to properly handle powers for negative numbers. See this discussion for more info.
- The comparison operator (`<`) and the logical operator (`&`) are both vectorised, allowing a very compact expression for counting up how many of the simulated Y's are inside the interval of interest.