Lecture 9 - Exercise Solutions

Exercise 1

```
set.seed(9)
x1 = rbinom(n = 20000, size = 5, prob = 0.3)
x2 = rgamma(n = 30000, shape = 2, rate = 1)
```

1.1

```
cat('The mean of X1 =',mean(x1))
```

```
## The mean of X1 = 1.5097
```

```
cat('\nThe mean of X2 =',mean(x2))
```

```
##
## The mean of X2 = 1.998389
```

The functions `rbinom` and `rgamma` draw n random numbers from the binomial and gamma distributions respectively, with the parameters n (number of independent Bernoulli trials) and p (probability of success) for the binomial distribution, and α (shape) and β (rate) for the binomial distribution. The theoretical mean of the binomial distribution is $n \times p = 5 \times 0.3 = 1.5$ and the theoretical mean of the gamma distribution is $\frac{\alpha}{\beta} = \frac{2}{1} = 2$. We find that the sample means are close to the theoretical means.

1.2

```
my.mean <- function(x){
  # retrieving number of observations in vector x
  n = length(x)

  # calculating the sum of all values in vector x
  my.sum = 0
  for(i in 1:n){
    my.sum = my.sum + x[i]
  }

  # calculating the mean of x
  mean.x = my.sum / n

  # returning the mean
  return(mean.x)
}
```

The results of the `my.mean` function corresponds with `R`'s default `mean` function.

```
# checking function
my.mean(x1)
```

```
## [1] 1.5097
```

```r
mean(x1)
```

```
## [1] 1.5097
```

```r
my.mean(x2)
```

```
## [1] 1.998389
```

```r
mean(x2)
```

```
## [1] 1.998389
```

1.3

Comparing the results of the benchmark, `R`'s base `mean()` function clearly outperforms `my.mean()`. Note that `x1` is evaluated faster than `x2`; this can be attributed to the fact that `x2` has more observations than `x1`.

```r
library(rbenchmark)

t.eval = benchmark(
  'my.mean(x1)' = {my.mean(x1)},
  'mean(x1)' = {mean(x1)},
  'my.mean(x2)' = {my.mean(x2)},
  'mean(x2)' = {mean(x2)},
  replications = 500
)

t.eval
```

```
##          test replications elapsed relative user.self sys.self user.child
## 2    mean(x1)          500   0.014    1.000     0.014    0.000          0
## 4    mean(x2)          500   0.033    2.357     0.031    0.001          0
## 1 my.mean(x1)          500   0.292   20.857     0.285    0.002          0
## 3 my.mean(x2)          500   0.429   30.643     0.404    0.006          0
##   sys.child
## 2         0
## 4         0
## 1         0
## 3         0
```

Exercise 2

```r
set.seed(9)
n = 2000
p = 500
m1 = matrix(rnorm(n*p,mean = 4.7, sd = 0.5), ncol = p)
```

2.1

```r
t.eval2 = benchmark(
  # the apply() function
  'apply' = {apply(m1, 2, mean)},

  # the colMeans() function
  'colMeans' = {colMeans(m1)},

  # pre-allocated for-loop
  'pre-allocated for loop' = {
    means = rep(NA, p)
    for(i in 1:ncol(m1)){
      means[i] = mean(m1[,i])
    }
  },

  # for-loop without pre-allocation
  'for loop without pre-allocation' = {
    means = c()
    for(i in 1:ncol(m1)){
      means = c(means, mean(m1[,i]))
    }
  },

  replications = 100
)

t.eval2
```

```
##                               test replications elapsed relative user.self
## 1                            apply          100   1.578   17.533     1.404
## 2                         colMeans          100   0.090    1.000     0.090
## 4 for loop without pre-allocation          100   1.298   14.422     1.149
## 3           pre-allocated for loop          100   1.186   13.178     1.067
##   sys.self user.child sys.child
## 1    0.118          0         0
## 2    0.000          0         0
## 4    0.108          0         0
## 3    0.090          0         0
```

2.2

Based on the benchmark, the `apply` function is the slowest for this task (look at the relative-column). Both `for`-loops appear to be about equally as fast. For this specific task `colMeans` is by far the fastest. This is to be expected, as it is specifically assigned for this task. Note that it is possible that you might get substantially different results depending on your hardware and operating system.

Exercise 3

For each flat inhabitant there is a probability $p = 0.38$ of owning a car. Since there are 127 inhabitants, we can say we have $n = 127$ independent observations. Thus we have a binomial distribution. The probability of having more than 44 inhabitants who own a car is about 75.28%.

,

$$P(X > 44) = 1 - P(X \leq 44) = 1 - \sum_{0}^{x} \binom{n}{x} p^x (1-p)^{n-x} = 1 - \sum_{i=0}^{44} \binom{127}{i} 0.38^i (1-0.38)^{127-i}$$

```
1-pbinom(q = 44, size = 127, prob = 0.38, lower.tail = T)
```

```
## [1] 0.7527784
```

Alternative, you could use the following two less efficient methods shown below.

```
# option 2
sum(dbinom(x = 45:127, size = 127, prob = 0.38))
```

```
## [1] 0.7527784
```

```
# option 3
prob = 0
for(i in 45:127){
  prob = prob + ((0.38^i)*(1-0.38)^(127-i))*choose(127,i)
}
prob
```

```
## [1] 0.7527784
```

Exercise 4

4.1

```
dpois(x = 7,lambda = 6)
```

```
## [1] 0.137677
```

4.2

Since the gamma distribution is continuous, $P(Y = 3) = 0$, since the probability of an exact number is always 0 in a continuous distribution.

4.3

The Poisson distribution is discrete, thus we could sum the probability density function as $P(2 < X < 5) = P(X = 3) + P(X = 4)$. Alternatively, we could take the difference between the CDFs $P(2 < X < 5) = P(X <= 4) - P(X <= 2) = F_X(4) - F_X(2)$.

```
# option 1
sum(dpois(3:4, lambda = 6))
```

```
## [1] 0.2230877
```

```
# option 2
ppois(q = 4, lambda = 6) - ppois(q = 2, lambda = 6)
```

```
## [1] 0.2230877
```

4.4

The gamma distribution is continuous, thus we need to take the difference between the cumulative distribution function $P(1 < Y < 3) = P(Y <= 3) - P(Y <= 1) = F_Y(3) - F_Y(1)$.

```
pgamma(q = 3, shape = 3, rate = 2) - pgamma(q = 1, shape = 3, rate = 2)
```

```
## [1] 0.6147076
```

4.5

In this exercise we want to evaluate the function $F_X(5)$, thus we need to fill in the x-value in the distribution function. This could be done by using `ppois()`.

```
ppois(q = 5, lambda = 6)
```

```
## [1] 0.4456796
```

4.6

```
ppois(q = 3, lambda = 6) + pgamma(q = 10, shape = 3, rate = 2)
```

```
## [1] 1.151203
```

Exercise 5

5.1

We use the `rnorm` and `rbeta` to generate 10000 observations from each distribution.

```
# we use set.seed to make sure the outcome is reproducible
set.seed(123)

# draw random observations
X <- rnorm(1e4, mean = 3, sd = 1.4)
Y <- rbeta(1e4, shape1 = 2, shape2 = 2)
```

5.2

```
# calculate z = x/y
Z <- X / Y

# look at the first 6 z's
head(Z)
```

```
## [1] 2.294287 5.910857 6.930086 9.037450 5.647067 6.799053
```

5.3

```
mean(Z)
```

```
## [1] 9.133582
```

```
var(Z)
```

```
## [1] 226.961
```