R Programming: Worksheet 5

By the end of today you should understand the `apply()` family of functions, as well as creating and manipulating multi-way arrays.

1. **Warm Up**

   Use `apply()`, `tapply()`, `sapply()`, `mapply()` and `replicate()` to solve these problems.

   (a) Generate a list of 50 random datasets, each consisting of 10 independent t_5 distributed random variables. [The relevant function is `rt()`.]

   ```
   > replicate(50, rt(10, df = 5), simplify = FALSE)
   ```

   (b) Generate a list of length 20, where the ith entry in the list is the sequence of numbers $1, \ldots, i$.

   ```
   > lapply(1:20, seq)
   ```

   (c) Generate the following random matrix `X`:

   ```
   > set.seed(2014)
   > X <- matrix(rexp(200), 20, 10)
   ```

   Find the smallest entry in each column of `X`.

   ```
   > apply(X, 2, min)
   ```

   (d) Look at the data frame `CO2` (this is preloaded into `R`). How would you determine which columns are numeric? *[Hint: there is a function called* ***`is.numeric()`****]* *Note that the approach using* ***`apply()`*** *doesn't work here, because it tries to convert the data frame into a matrix where everything has the same type.*

   ```
   > sapply(CO2, is.numeric)

   ##     Plant      Type Treatment      conc    uptake
   ##     FALSE     FALSE     FALSE      TRUE      TRUE

   > apply(CO2, 2, is.numeric)  # doesn't work!

   ##     Plant      Type Treatment      conc    uptake
   ##     FALSE     FALSE     FALSE     FALSE     FALSE
   ```

2. **Death Penalty Data**

   Here is a famous data set which is used to illustrate *Simpson's paradox*. It records the races of the victim and defendants in various murder cases in Florida between 1976 and 1987, and whether or not the death penalty was imposed upon the killer. The data are presented as counts.

Victim	White	
Defendant	White	Black
Yes	53	11
No	414	37

Victim	Black	
Defendant	White	Black
Yes	0	4
No	16	139

(a) Create a 3-way array in `R` to hold these data, with the three dimensions representing the penalty, the defendant's race and the victim's race respectively.

```
> count = c(53, 414, 11, 37, 0, 16, 4, 139)
> tab = array(count, c(2, 2, 2))
> tab

## , , 1
##
##      [,1] [,2]
## [1,]   53   11
## [2,]  414   37
##
## , , 2
##
##      [,1] [,2]
## [1,]    0    4
## [2,]   16  139
```

You can include suitable dimension names on a table with the `dimnames()` function.

```
> house <- dget("housing.dat")
> dimnames(house)
```

This is just a list of names for each dimension.

(b) Create a similar list of dimension names for your death penalty data, called (say) `listOfNames`. Set it to the table by running

```
> listOfNames <- list(DeathPen = c("Yes", "No"), Defendant = c("White",
+     "Black"), Victim = c("White", "Black"))
> dimnames(tab) <- listOfNames
```

(c) Obtain the marginal distribution (i.e. summing out over the other variables) of the penalty imposed against the defendant's race. *Either using* ***`apply()`****:*

```
> mar12 = apply(tab, 1:2, sum)
```

or ***`margin.table()`****:*

```
> mar12 = margin.table(tab, 1:2)  # special function for this
> mar12

##          Defendant
## DeathPen White Black
##      Yes    53    15
##      No    430   176
```

(d) Use this to estimate the probability of imposing the death penalty conditional upon race of the defendant. What do you conclude?

```
> mar12/rep(colSums(mar12), each = 2)

##          Defendant
## DeathPen  White   Black
##      Yes 0.1097 0.07853
##      No  0.8903 0.92147
```

White defendants are more likely to be executed.

(e) Now do the same but only for white victims. What do you conclude? What about only for black victims?

```
> tab[, , 1]/rep(colSums(tab[, , 1]), each = 2)

##          Defendant
## DeathPen  White  Black
##      Yes 0.1135 0.2292
##      No  0.8865 0.7708

> tab[, , 2]/rep(colSums(tab[, , 2]), each = 2)

##          Defendant
## DeathPen White   Black
##      Yes     0 0.02797
##      No      1 0.97203
```

We find that if the victim is white, a black defendant is more likely to be executed; similary if the victim is black, a black defendant is more likely to be executed.

(f) * Can you explain how this is possible? *The 'paradox' occurs because in cases for which the victim is white, the death penalty is more likely, and cases with white victims are more likely to involve white defendants.*

[You can also use the function `conditionTable()` in the excellent(!) package `rje` to do the bits in part (d) and (e).]

3. **Central Limit Theorem**

(a) Write a function with arguments `n` and `k`, which generates `k` data sets each consisting of `n` independent uniform random variables on $(0, 1)$, and returns their mean. Do this in **three** different ways, using (i) a loop; (ii) `replicate()`; (iii) the function `rowMeans()`. [The function `runif()` may be useful.]

```
> uniMeans1 = function(n, k) {
+     out = numeric(k)
+     for (i in 1:k) out[i] = mean(runif(n))
+     return(out)
+ }
> uniMeans2 = function(n, k) {
+     replicate(k, mean(runif(n)))
+ }
```

```
> uniMeans3 = function(n, k) {
+     rowMeans(matrix(runif(n * k), k, n))
+ }
```

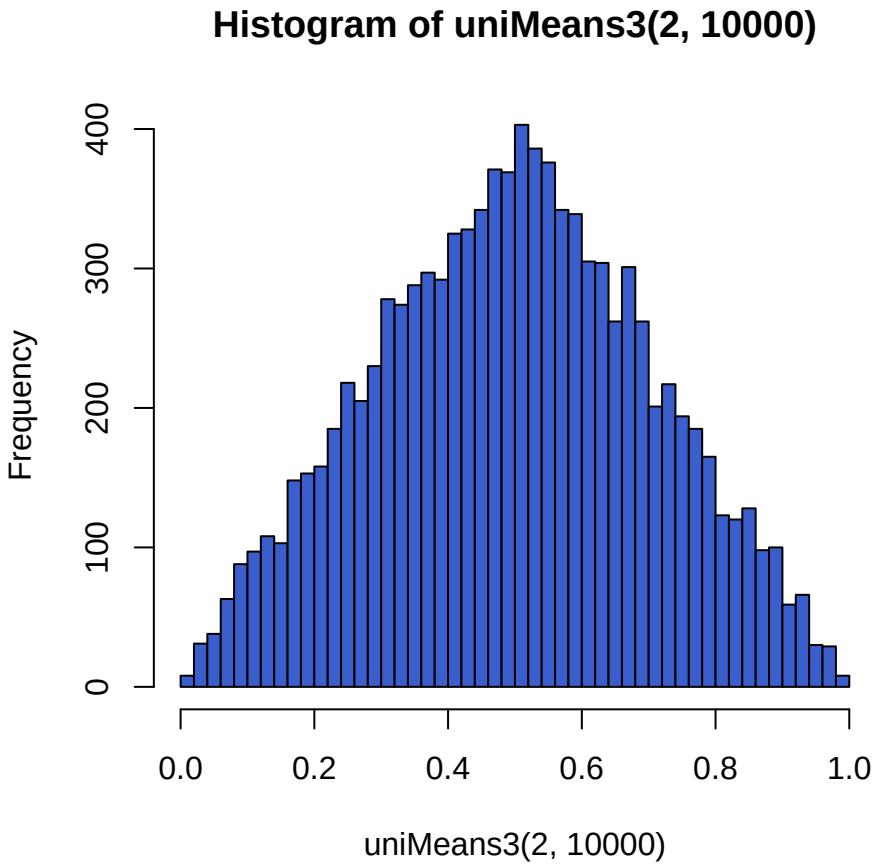
(b) Using `system.time()`, time your functions for a few different values of `n` and `k`. Which is the fastest? Which function is the easiest for you to read and understand? *Using code below you should find (looking at the 'elapsed' time) that the loop is the slowest, `replicate()` is slightly faster, and `rowSums()` is much faster.*

```
> n = 1
> k = 1000
> system.time(uniMeans1(n, k))

##    user  system elapsed
##    0.01    0.00    0.01

> system.time(uniMeans2(n, k))

##    user  system elapsed
##   0.008   0.000   0.008

> system.time(uniMeans3(n, k))

##    user  system elapsed
##       0       0       0
```

On the other hand, the `replicate()` version is somewhat simpler and easier to read.

(c) Run your preferred function for $k = 10,000$ and each of $n = 1, 2, 3, 5, 10$; plot the results as a histogram in each case. (Use `hist()`, and play around with the `breaks` and `col` options.) *For example:*

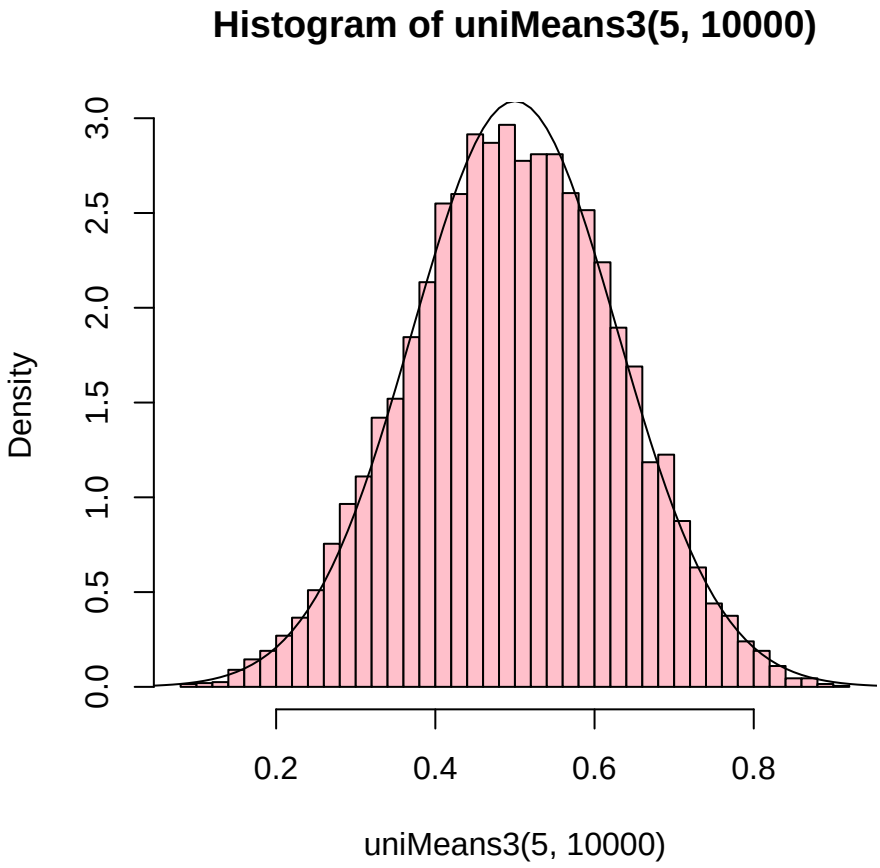
```
> hist(uniMeans3(2, 10000), col = "royalblue3", breaks = 50)
```



(d) For the case $n = 5$, plot a normal density over your histogram with the same mean and variance as the random variables generated by your function: here I mean the *exact* mean and variance. The function `dnorm()` may be useful. [Hint: recall that the variance of a Uniform$(0, 1)$ random variable is $\frac{1}{12}$].

The mean is $\frac{1}{2}$ and the standard deviation $(12n)^{-1/2}$. We have to set `freq=FALSE` to get the histogram to represent a density.

```
> den = function(x, n) dnorm(x, mean = 0.5, sd = sqrt(1/(12 *
+     n)))
> hist(uniMeans3(5, 10000), freq = FALSE, breaks = 40, col = "pink")
> plot(function(x) den(x, 5), 0, 1, add = TRUE)
```



4. **Confidence Intervals**

   (a) Write a function which, given a real vector `x`, calculates a 95% confidence interval for the mean, using the usual t-distribution approximation. [See `?qt`]

```
> confInt = function(x) {
+     mu = mean(x)
+     s = sd(x)
+     n = length(x)
+     out = mu + c(-1, 1) * qt(0.975, df = n - 1) * s/sqrt(n)
+     out
+ }
```

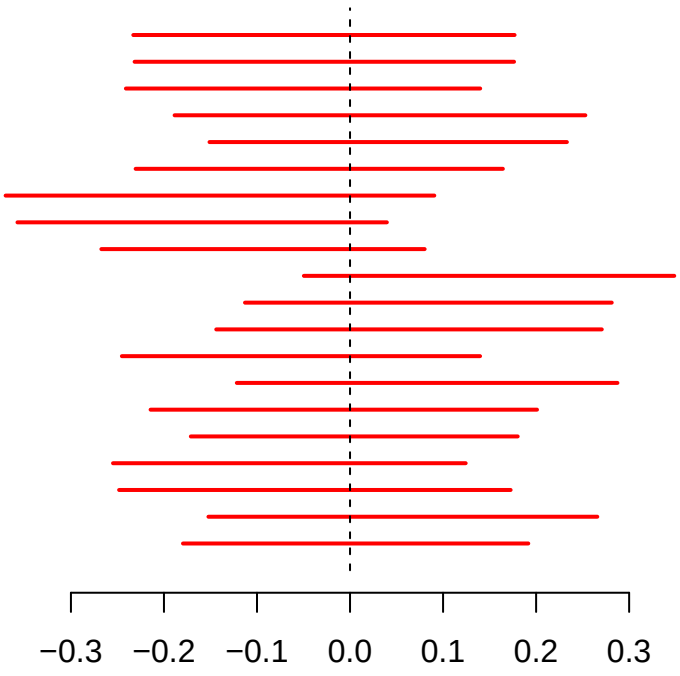
   (b) Write a second function with argument `n`, which generates 20 data sets each consisting of `n` standard normal random variables, and calculates a 95% confidence interval for each data set. The function should return a list containing two vectors of length 20 named `lower` and `upper`, consisting of the lower and upper ends of each interval.

```
> genCIs = function(n) {
+     upper = lower = numeric(20)
+     data = replicate(20, rnorm(n), simplify = FALSE)  # or use a loop
```

```
+     tmp = sapply(data, confInt)
+     return(list(upper = tmp[2, ], lower = tmp[1, ]))
+ }
```

(c) Write a third function which obtains 20 confidence intervals as above, and then plots them. The plot should consist of 20 horizontal lines, one drawn below the other.

```
> plotCIs = function(n) {
+     CIs = genCIs(n)
+     plot.new()
+     plot.window(ylim = c(0, 21), xlim = c(min(CIs$lower),
+         max(CIs$upper)))
+     axis(side = 1)
+     for (i in 1:20) points(c(CIs$lower[i], CIs$upper[i]),
+         c(i, i), type = "l", lwd = 2, col = 2)
+     points(c(0, 0), c(0, 21), type = "l", lty = 2)
+ }
> plotCIs(100)
```



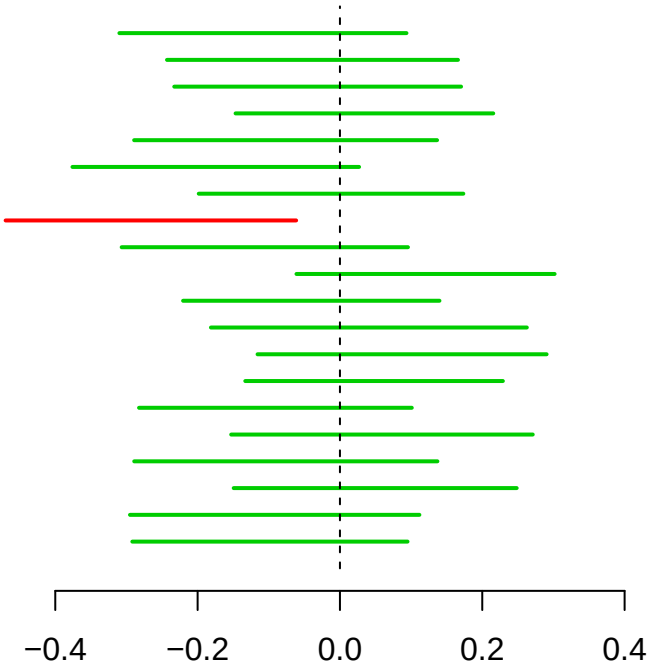
Plot these a few times and watch the lines dance around!

(d) Improve your plot function in the following ways:

(i) make the true mean always be in the centre of the plot;

(ii) plot intervals which cover the true mean in a different colour.

```
> plotCIs = function(n) {
+     CIs = genCIs(n)
+     plot.new()
+     plot.window(ylim = c(0, 21), xlim = c(-1, 1) * max(abs(unlist(CIs))))
+     axis(side = 1)
+     covers = (CIs$lower < 0) & (CIs$upper > 0)
+     for (i in 1:20) points(c(CIs$lower[i], CIs$upper[i]),
+         c(i, i), type = "l", lwd = 2, col = 2 + covers[i])
+     points(c(0, 0), c(0, 21), type = "l", lty = 2)
+ }
> set.seed(2140)
> plotCIs(100)
```



(e) Try changing the mean you use to generate the data, and see how this changes your plot.

(f) * How might you use this function to explain the frequentist interpretation of confidence intervals to a student?