R Programming: Worksheet 3

By the end of the practical you should feel confident writing and calling functions, and using `if()`, `for()` and `while()` constructions.

1. Review

(a) Write a function which takes a numeric vector \( x \), and returns a named list containing the mean, median and variance of the values in \( x \).

```r
> summarize = function(x) {
+   list(mean = mean(x), median = median(x), variance = var(x))
+ }
```

[Hint: If you’re not sure what the name of a function is, try using fuzzy search: e.g. `??variance`.

(b) Write a function with arguments \( x \) and \( n \), which evaluates \( \sum_{i=0}^{n} e^{-x} \frac{x^i}{i!} \) (you can use `factorial()` for this).

```r
> parsym = function(x, n) {
+   sq = seq(from = 0, to = n)
+   exp(-x) * sum(x^sq/factorial(sq))
+ }
```

*Note this is the same as `ppois()`:

```r
> ppois(15, lambda = 10)
> parsym(x = 10, 15)
```

(c) Write a function which goes through every entry in a list, checks whether it is a character vector (`is.character()`), and if so prints it (`print()` or `cat()`).

*There are various possibilities, here is one:

```r
> printChar = function(lst) {
+   for (i in lst) {
+     if (is.character(i))
+       print(i)
+   }
+ }
```

*This can be done more neatly using `sapply()`, as we’ll see in Lecture 6.*

(d) Write a function with an argument \( k \) which simulates a symmetric random walk (see Sheet 1, Question 4), but that stops when the walk reaches \( k \) (or \( -k \)).

```r
> rndwlk = function(k) {
+   curr = 0  # current position
+   out = 0   # vector of all positions
+   while (abs(curr) < k) {
+     curr = curr + sample(c(1, -1), 1)  # new position
+     out = c(out, curr)  # add to vector
+   }
+ }
```
2. Moving Averages

(a) Write a function to calculate the moving averages of length 3 of a vector \((x_1, \ldots, x_n)^T\). That is, it should return a vector \((z_1, \ldots, z_{n-2})^T\), where

\[
z_i = \frac{1}{3} (x_i + x_{i+1} + x_{i+2}), \quad i = 1, \ldots, n - 2.
\]

Call this function \texttt{ma3()}.

\[
\begin{align*}
> \texttt{ma3} & = \texttt{function(x) \{} \\
& \quad \texttt{n = length(x)} \\
& \quad \texttt{x1 = x[-(1:2)]} \\
& \quad \texttt{x2 = x[-c(1, n)]} \\
& \quad \texttt{x3 = x[-c(n - 1, n)]} \\
& \quad \texttt{(x1 + x2 + x3)/3} \\
& \quad \} \\
> x & = \texttt{rnorm(100)} \\
> \texttt{plot(ma3(x), type = "l")}
\end{align*}
\]

(b) Write a function which takes two arguments, \(x\) and \(k\), and calculates the moving average of \(x\) of length \(k\). [Use a \texttt{for()} loop.]

\[
\begin{align*}
> \texttt{ma} & = \texttt{function(x, k) \{} \\
& \quad \texttt{n = length(x)} \\
& \quad \texttt{out = x[-(1:(k - 1))]/k} \\
& \quad \texttt{for (i in 2:k)} \{ \\
& \quad \quad \texttt{out = out + x[seq(from = k + 1 - i, to = n + 1 - i)]/k} \\
& \quad \} \\
& \quad \texttt{out} \\
& \quad \} \\
> \texttt{max(abs(ma(x, 3) - ma3(x)))}
\end{align*}
\]

(c) How does your function behave if \(k\) is larger than (or equal to) the length of \(x\)? You can tell it to return an error in this case by using the \texttt{stop()} function. Do so.

(d) How does your function behave if \(k = 1\)? What should it do? Fix it if necessary. \textit{It should just return \(x\), but it may cause the \texttt{for()} loop to misbehave if you used \(1:(k-1)\) in it.}

\[
\begin{align*}
> \texttt{ma} & = \texttt{function(x, k) \{} \\
& \quad \texttt{if (k == 1)} \\
& \quad \quad \texttt{return(x)} \\
& \quad \texttt{n = length(x)} \\
& \quad \texttt{out = x[-(1:(k - 1))]/k}
\end{align*}
\]
for (i in 2:k) {
    out = out + x[seq(from = k + 1 - i, to = n + 1 - i)]/k
}
out

> max(abs(ma(x, 3) - ma3(x)))

3. Poisson Processes

A Poisson process of rate $\lambda$ is a random vector of times $(T_1, T_2, T_3, \ldots)$ where the interarrival times $T_1, T_2 - T_1, T_3 - T_2, \ldots$ are independent exponential random variables with parameter $\lambda$. Note that this implies $T_{i+1} > T_i$.

(a) Write a function with arguments $\lambda$ and $M$ which generates the entries of a Poisson process up until the time reaches $M$. [Hint: `rexp()` generates exponential random variables.] We need to stop when the first $T_i > M$, and then only return the values of $T_i$ which are less than $M$.

```r
> poisProc = function(lambda, M) {
+    out = rexp(1, lambda)
+    len = 1
+    while (out[len] < M) {
+        # keep adding values until one exceeds M
+        out = c(out, out[len] + rexp(1, lambda))
+        len = len + 1
+    }
+    # return everything except the value > M
+    return(out[-len])
+}
```

(b) Generate 10,000 of these with $\lambda = 5$ and $M = 1$, recording the lengths of the vectors returned in each case. Plot these lengths as a histogram (`hist()`), and calculate their mean and variance.

```r
> lens = numeric(10000)
> for (i in 1:10000) lens[i] = length(poisProc(5, 1))

The mean and variance will both be about 5. What sort of distribution do you think the lengths have? A Poisson distribution with parameter $\lambda$, hence the name!

4. *Functions of Functions

(a) Write a function which calculates the value of arbitrary Taylor series given the symbolic form of each term, a position, and a specific number of terms. For example, if I want the Taylor expansion for $\exp(x) = \sum_{i=0}^{n} x^i/i!$, I would provide $x$, $n$, and the function
> tayExp = function(x, i) x^i/factorial(i)

There are better ways to do this using \texttt{sapply()}, but for now...

> taylor = function(f, x, n) {
+   out = 0
+   for (i in seq(from = 0, to = n)) out = out + f(x, i)
+   out
+ }
> taylor(tayExp, 0.5, 20) - exp(0.5)

(b) Try this with the series $\sum_{i=1}^{n}(-1)^{i-1}x^i/i$ (note where the index on the sum starts), and compare the answer for $x = 0.5$, $n = 20$ to $\log(1 + x)$. You just have to make sure you deal with the $i = 0$ case separately.

> tayLog = function(x, i) ifelse(i == 0, 0, -(-x)^i/i)

(c) Make the function so that instead of specifying a specific number of terms, it will stop when the difference between successive terms is smaller than some tolerance \texttt{eps}. Make sure the maximum number of terms is still \(n+1\). [Hint: a \texttt{break} statement might be useful: look at \texttt{?break}.]

> taylor2 = function(f, x, n, eps = 1e-16) {
+   tmp = eps + 1
+   out = 0
+   for (i in seq(from = 0, to = n)) {
+     tmp = f(x, i)
+     out = out + tmp
+     if (abs(tmp) < eps)
+       break
+   }
+   out
+ }

5. *Ellipsis*

(a) Construct a function which takes two matrices \(A\) and \(B\), and returns the block diagonal matrix

\[
\begin{pmatrix}
A & 0 \\
0 & B
\end{pmatrix}
\]

> blkDiag = function(A, B) {
+   dA = dim(A)
+   dB = dim(B)
+   out[1:dA[1], 1:dB[2]] = A
+   out
+ }
Some functions have an ellipsis argument which looks like three dots ...

```r
> max
## function (..., na.rm = FALSE) .Primitive("max")
```

This means they can have an arbitrary number of arguments. You can turn your ellipsis into a list by putting the line

```r
> myargs <- list(...)
```

in your function. `myargs` is then a list of all the arguments supplied.

(b) Construct a function which takes an arbitrary number of matrices $A_1, A_2, \ldots, A_k$ as separate arguments (not as a list) and returns the block diagonal matrix

$$
\begin{pmatrix}
A_1 & 0 & \cdots & 0 \\
0 & A_2 & & \\
& \vdots & \ddots & 0 \\
0 & \cdots & 0 & A_k
\end{pmatrix}
$$

(c) Make sure your function works sensibly even if the entries are vectors (treat these as column vectors) or scalars. This involves just being careful using `dim()`. 