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.

   [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 evaulates $\sum_{i=0}^{n} \frac{e^{-x}x^i}{i!}$ (you can use `factorial()` for this).

   (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()`).

   (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 $-k$).

2. **Moving Averages**

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

   $$z_i = \frac{1}{3}(x_i + x_{i+1} + x_{i+2}), \qquad i = 1, \dots, n-2.$$

   Call this function `ma3()`.

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

   (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 `stop()` function. Do so.

   (d) How does your function behave if $k = 1$? What should it do? Fix it if necessary.

3. **Poisson Processes**

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

   (a) Write a function with arguments λ and M which generates the entries of a Poisson process up until the time reaches M. [Hint: `rexp()` generates exponential random variables.]

   (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.

   What sort of distribution do you think the lengths have?

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)
```

(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 $\log(1 + x)$.

(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 `eps`. Make sure the maximum number of terms is still `n+1`. [Hint: a `break` statement might be useful: look at `?break`.]

5. ***Ellipsis**

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

$$\begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix}$$

Some functions have an ellipsis argument which looks like three dots `...`

```
> 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

```
> 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

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

(c) Make sure your function works sensibly even if the entries are vectors (treat these as column vectors) or scalars.