R Programming: Worksheet 1

Try to focus on questions 1–3; there are a couple extra for those who finish quickly. Bits with an asterisk (*) are slightly harder, and are either non-examinable or will be covered later.

Functions used:

```
seq(), rep()
sample(), rnorm()
matrix(), t(), solve(), ncol()
apply()
sd(), var(), cumsum()
rbinom(), pbinom(), diag(), %*%
plot() # 1-dimensional data
```

1. **Sequences**

   Generate the following sequences and matrices

   (a) $1, 3, 5, 7, \ldots, 21$.

   (b) $50, 47, 44, \ldots, 14, 11$.

   (c) $1, 2, 4, 8, \ldots, 1024$.

   (d)
   $$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix}$$

```
> seq(1, 21, by = 2)
>
> seq(50, 11, by = -3)
>
> 2^(0:10)
>
> t(matrix(1:16, 4))
> # computationally better:
> matrix(1:16, 4, byrow = TRUE)
```

2. **Sampling**

   The command `sample()` performs random sampling; for example, to give a random permutation of the numbers 1 to 10, we could do one of:

```
> sample(10)
> sample(1:10)
```

   (a) A scientist needs to experiment upon 4 conditions, 5 times each. Generate a vector $(1, 1, 1, 1, 1, 2, \ldots, 4, 4)^T$ of length 20, representing these conditions.

```
> ex = rep(1:4, each = 5)
```

(b) The scientist wants to do the 20 experiments in a completely random order; use `sample()` to reorder the elements of the vector from (a).

```
> ex = sample(ex)
```

(c) The scientist calls the conditions A, B, C and D. How would you return a character vector with entries `"A"`, `"B"`, `"C"`, `"D"` containing your random permutation?

The slickest way is

```
> LETTERS[ex]
```

but

```
> lets = c("A", "B", "C", "D")
> lets[ex]
```

also works.

3. **Matrices and `apply()`**

Remember that a matrix can be created with the command `matrix()`, and that it fills in by column first:

```
> A = matrix(1:12, nrow = 3)
```

The command `apply()` allows you to neatly perform an operation on each row (or column) of a matrix. For example, if you want the row-by-row averages of the matrix A, you could use

```
> apply(A, 1, mean)
```

or for the column means, use

```
> apply(A, 2, mean)
```

(a) Create a 10×11 matrix of independent standard normal random variables; call it `A`.

```
> A = matrix(rnorm(110), 10, 11)
```

(b) How would you find the maximum entry in each row of A?

```
> apply(A, 1, max)
```

(c) Calculate the standard deviation of each column of `A` (the command you need is `sd()`).

```
> apply(A, 2, sd)
```

(d) Select the last column of `A`, and call it `b`. Then remove the last column from the original `A`. Do this using the function `ncol()`.

```
> b = A[, ncol(A)]
> A = A[, -ncol(A)]
```

(e) Solve the system of linear equations $Ax = b$. *Either of:*

```
> x = solve(A) %*% b
> x = solve(A, b)  # more elegant
```

(f) Find a vector containing the sums of each row of `A`.

```
> apply(A, 1, sum)
```

Can you think of (or find) any other ways of achieving this? *You could use*

```
> rowSums(A)
> A %*% rep(1, ncol(A))
```

(g) * Create a second matrix `B`, where the ith column of `B` is the sum of the first i columns of `A`.

```
> B = t(apply(A, 1, cumsum))
```

Note that `apply()` *will put output from the function you use along the first dimension of the matrix or array it returns, so you need to transpose to get this question to work. Alternatively and less elegantly,*

```
> B = A
> for (i in 2:ncol(A)) B[, i] = B[, i - 1] + A[, i]
```

4. **Random Walks**

A *random walk* on the integers is a sequence $X_0, X_1, X_2, \ldots$ with $X_0 = 0$, and

$$X_i = X_{i-1} + D_i,$$

where the D_i are independent with $P(D_i = +1) = P(D_i = -1) = \frac{1}{2}$.

(a) Have a look at the documentation for the function `sample()`. Use it to generate a vector $(D_1, \ldots, D_{25})^T$.

```
> D = sample(c(-1, 1), 25, replace = TRUE)
```

or

```
> D = 2 * rbinom(25, 1, 0.5) - 1
```

(b) Use the command `cumsum()` to generate $(X_0, X_1, \ldots, X_{25})^T$ from this.

```
> X = c(0, cumsum(D))
```

(c) Plot your random walk:

```
> plot(X, type = "l")
```

Try plotting the first 1,000 steps of a random walk.

(d) We can rewrite

$$X_n = \sum_{i=1}^{n} D_i = 2Z_n - n$$

where the distribution of Z_n is binomial (with what parameters?) To generate a random binomial distribution use `rbinom()`:

```
> rbinom(1, 25, 0.5)
```

What does each of the arguments `1`, `25`, and `0.5` do? Remember to use the help file if necessary. *1 is the number of independent random variables to generate, 25 is the number of binomial trials, and 0.5 the probability of success.*
Write some code to generate a realization of X_{25}.

```
> x25 = 2 * rbinom(1, 25, 0.5) - 25
```

(e) Generate a vector containing the value of X_{25} for 100,000 independent realizations of the symmetric random walk. How could we estimate the probability of X_{25} exceeding 10?

```
> x = 2 * rbinom(1e+05, 25, 0.5) - 25
> mean(x > 10)
```

(f) How could we calculate this exactly? Compare to your answer above. [Try looking at `?pbinom`.] $X_{25} \geq 10$ *whenever* $N_{25} \geq 17.5$. *So*

```
> 1 - pbinom(17, 25, 0.5)

## [1] 0.02164
```

5. **Diagonals**

(a) Create a diagonal matrix whose diagonal entries are $1, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{10}$. Call it `D`.

```
> D = diag(1/(1:10))
```

(b) Now define a 10×10 matrix whose entries are all -1, except on the diagonal, where the entries should be 4. Call it `U`

```
> U = 5 * diag(10) - 1
```

(c) What is the length of the first column vector in `U`?

```
> sqrt(sum(U[, 1]^2))

## [1] 5
```

Renormalize the entries of `U` so that each column is a unit vector. *All the vectors are the same length, so*

```
> U = U/sqrt(sum(U[, 1]^2))
```

Check directly that your approach is correct. *Two ways of doing this:*

```
> apply(U, 2, function(x) sqrt(sum(x^2)))
> 10 * apply(U, 2, var)
```

(d) Calculate the matrix UDU^T, and call it `X`.

```
> X = U %*% D %*% U
```

(e) Find the eigenvalues of `X` numerically (try typing `??eigenvalue`). Is this what you expected?

```
> eigen(X)
```

These are just the entries of `D`, which is clear from the definition of `X`.

(f) * Can you use vector recycling to calculate DU^T without using matrix multiplication? *Writing out the consequences of left-multiplying by a diagonal matrix, we can see that it's the same as scaling each row of U^T by the corresponding entry on the diagonal of D.*

```
> c(1/(1:10)) * t(U)
```