Computer Practical 1
Simulation Basics

1. *Pseudorandom Numbers & Transformation Methods*

   (a) RANDU was an extremely popular PRNG for many years (it was widely used on IBM and DEC mainframes, for example). It's a linear congruential generator with recursion:

$$Z_j = 65539 Z_{j-1} \mod 2^{31} \qquad X_j = Z_j / 2^{31}$$

   ⋆ i. Iterate the recursion twice to write Z_j as a function of Z_{j-1} *and* Z_{j-2}. Is there a clear issue here?

   ii. The following code is one possible implementation of the PRNG. Using this or your own code obtain a long sequence of random numbers from this generator.

```
randu.map <- function(z,a=2**16 + 3, c=0, M=2**31) {
    (a * z + c) %% M
}
randu <- function (n,z0, a=(2**16) + 3, c=0, M=2**31) {
    z <- vector('numeric', n)
    z[1] <- randu.map(z0,a,c,M)
    for(i in seq(from=2,to=n,by=1)) z[i] <- randu.map(z[i-1],a,c,M)
    return(z / M)
}

unif <- randu(n=100000,z0=123142)
```

   Plot a histogram of a reasonably long sequence obtained from this generator. Is there any clear departure from uniformity?

   iii. Plot a time series showing the evolution of the sequence over a thousand or so iterations. Is there any clear pattern?

   iv. Plot the even items in the sequence against the odd items; is there any clear pattern between successive items in the sequence?

   v. Investigate what you see if you treat the triples in the `randu` data set as coordinates in $\mathbb{R}^3$. If the `rgl` package is available then you might find its `plot3d` function useful. Do you see anything interesting?

   (b) Recall the Box-Muller method which transforms pairs of uniformly-distributed random variables to obtain a pair of independent standard normal random variates. If

$$U_1, U_2 \overset{\text{iid}}{\sim} \mathsf{U}[0,1]$$

   and

$$\begin{aligned} X_1 &= \sqrt{-2\log(U_1)} \cdot \cos(2\pi U_2) \\ X_2 &= \sqrt{-2\log(U_1)} \cdot \sin(2\pi U_2) \end{aligned}$$

   then $X_1, X_2 \overset{\text{iid}}{\sim} \mathsf{N}(0,1)$.

   i. Write a function which takes as arguments two vectors and returns the two vectors obtained by applying the Box-Muller transform elementwise.

   ii. Apply this function with the first vector corresponding to those numbers generated at odd indices and the second to those generated at even indices in your sequence; you should find 10,000 or so samples sufficient. Plot the two "independent" normal variables obtained by doing this, one against the other. What do you see?

iii. The R function `runif` provides access to a PRNG. The type of PRNG can be identified using the `rngkind` command; by default it will be a Mersenne-Twister (`http://en.wikipedia.org/wiki/Mersenne_twister`). Generate 10,000 $\mathsf{U}[0,1]$ random variables using this function, convert this to two vectors of 5,000 elements in the same way as you did with the RANDU values.

iv. Apply your Box Muller transformation to the same elements of this array as you did with the `randu` dataset. How do the results compare?

⋆ v. Try exploring other values of the coefficients with the linear congruential generator, e.g., try the sequence arising from the recursion

$$Z_j = 1229 Z_{j-1} + 1 \mod 2048 \qquad X_j = Z_j/2048$$

and see what happens when you push the resulting values through the Box Muller transformation.

2. *The Bootstrap:* This question can be answered in two ways. The more direct (and perhaps more informative, if you have the time to do so and the inclination to implement a bootstrap algorithm from scratch) is to use the `sample` function to obtain bootstrap replicates and to compute the required confidence intervals by direct means. More pragmatically, the `boot` library provides a function `boot` to obtain bootstrap samples and another, `boot.ci` which will provide various bootstrap confidence intervals.

   (a) The `Nile` dataset shows the rivers. Use a histogram or other visualisation to briefly explore this data.

   (b) What's the mean and median length of a river in this category?

   (c) Treating the data as a simple random sample, appeal to asymptotic normality to construct a confidence interval for the mean annual flow of the Nile.

   (d) Using the `boot::boot` function to obtain the sample and `boot::boot.ci` to obtain confidence intervals from that sample, or otherwise, obtain a *bootstrap percentile interval* for both the mean and median of the Nile's annual flow. For the median you may also wish to obtain the interval obtained by an optimistic appeal to asymptotic normality combined with a bootstrap estimate of the variance (`boot::boot.ci` will provide this).

   Note the following: `boot::boot` does the actual bootstrap resampling; it needs a function which takes two arguments to compute the statistic for each bootstrap sample, the first contains the *original* data and the second the index of the values included in a particular bootstrap resampling.

   (e) Are there any interesting qualitative differences between the various confidence intervals? How does this relate to the data?

   (f) Are your findings stable? If you repeat sthe bootstrap sampling do you recover similar behaviour?

   (g) Are there any reasons to doubt the accuracy of these confidence intervals?

3. *Convergence of Sample Approximations*

   (a) The `stats::ecdf` and `stats::plot.ecdf` functions compute and plot empirical distribution functions from a provided sample.

      i. Show plots of the empirical distribution function of samples of a variety of sizes ranging from 10 to 10,000 from a $\mathsf{U}[0,1]$ distribution. Add to your plots the distribution function of the $\mathsf{U}[0,1]$ distribution.

      ii. Repeat part *i.* with a standard normal distribution.

      iii. Repeat part *i.* with a Cauchy distribution.

   (b) For each of the three distributions considered in the previous part, determine $\sup_x |\hat{F}_n(x) - F(x)|$ for each n considered (consider only the sup over the sampled values of x) and plot these quantities against n. Do you notice anything interesting?

⋆ Indicates questions which can easily be done later if you're short of time.