HW 4 Random Number Generators and Latex

STAT 5400

Due: Sep 27, 2024 9:30 AM

Submit your solutions as an .Rmd file and accompanying .pdf file. Include all the **relevant** R code and output.

Always comment on your result whenever it is necessary.

Problems

1. Uniform random number generators in R

- Explore the source code of the R function `runif.R` by yourself. (You don't need to submit anything about this sub-question.)

https://github.com/wch/r-source

The architecture of the R function `runif` has several layers: (1) the R API: `runif.R` (2) the C code: `r-source/src/nmath/runif.c` which calls `rand_unif()` (3) the algorithm layer: `r-source/src/main/RNG.c` which implements several algorithms and Mersenne-Twister is the default algorithm.

- The Mersenne-Twister is a bit complicated. Let us work on a simpler algorithm, Linear Congruential Generator (LCG) Method. The LCG method is used to generate a sequence of pseudo-random numbers, and it is one of the oldest and simplest methods for generating pseudo-random numbers.

A tutorial of LCG is seen as in Section 9.3.2 of

https://homepage.stat.uiowa.edu/~luke/classes/STAT7400-2023/_book/simulation.html#uniform-random-numbers

The LCG is defined by the linear congruential relation:

$$X_{n+1} = (aX_n + c) \mod m$$

Where:

- X is the sequence of pseudo-random numbers
- a is the multiplier
- c is the increment
- m is the modulus
- X_0 (the seed) is the starting value of the sequence

Each number in the sequence is generated using the previous number. The results, which are modulo m, are normalized by dividing by m to fall within the interval $[0, 1)$.

Below is the pseudo Code for LCG

```
Algorithm LinearCongruentialGenerator
Input:
  n: the number of random numbers to generate
  a: the multiplier
  c: the increment
  m: the modulus
  seed: the starting value (seed) of the sequence
Output:
  A list of n pseudo-random numbers between 0 and 1.

Procedure:
  1. Initialize state to seed
  2. Initialize an empty list, random_numbers
  3. For i from 1 to n do
      a. state <- (a * state + c) mod m
      b. Append state/m to random_numbers
  4. End For
  5. Return random_numbers
```

- Based on the pseudo-code provided above, write an R function, named `LCG`, to generate `n` pseudo-random numbers between 0 and 1 using the Linear Congruential Generator method. The function should take five parameters: `n`, `a`, `c`, `m`, and `seed`.

Example Here's a set of parameters you can use to test your function:

a=1103515245 (Multiplier) c=12345 (Increment) m=2^31(Modulus)

```r
LCG <- function(n, a, c, m, seed) {
  state <- seed
  random_numbers <- vector(length = n)
  for(i in 1:n){
    state <-  (a * state + c) %% m
    random_numbers[i] <- state

  }
  return (random_numbers)
}
```

```r
random_numbers <- LCG(n = 10, a = 1103515245, c = 12345, m = 2^31, seed = 5400)
print(random_numbers)
```

```
##  [1] 1862695793 1774529792 1789548800  665195776  974987520  611663104
##  [7] 1820403968 1132242176 1415082240 1108757760
```

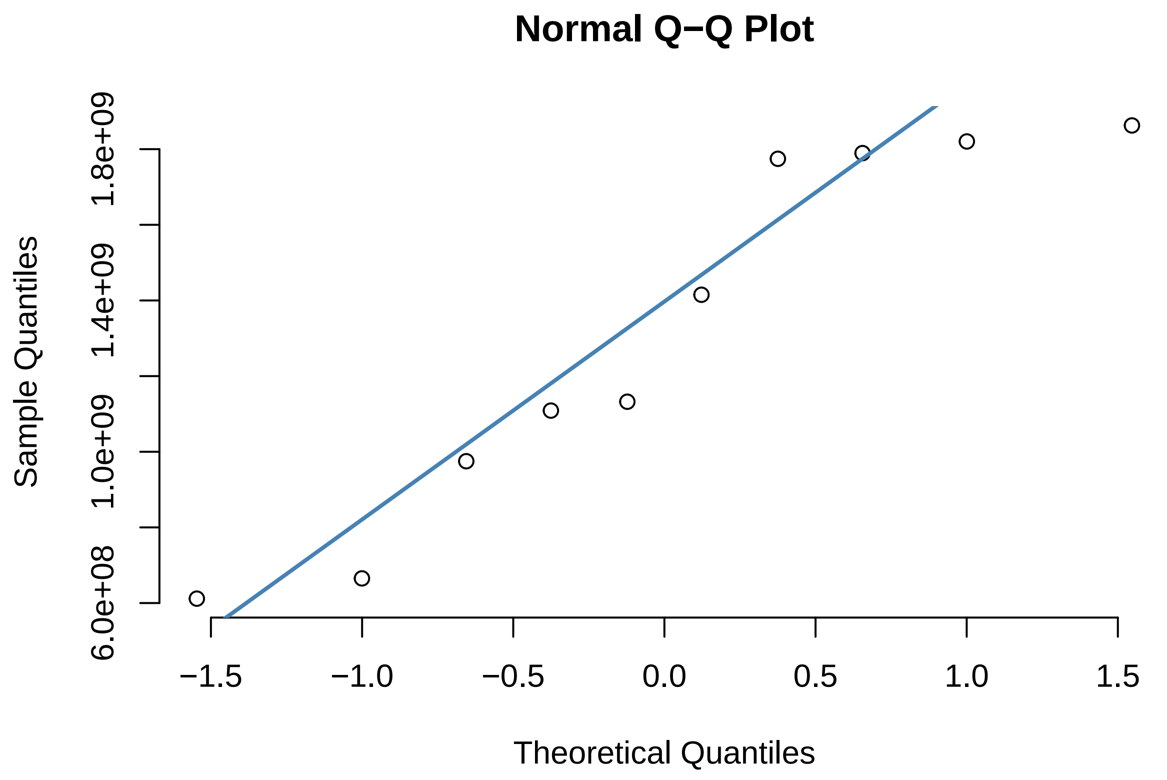
which basically functions the same to

```r
set.seed(5400)
runif(10)
```

```
##  [1] 0.96715156 0.85109119 0.90646319 0.40919220 0.84486948 0.25170803
##  [7] 0.24688742 0.03665934 0.94193288 0.54458507
```

2. Q-Q plot We will talk about the Q-Q plot in Monday's lecture. Several R packages provide implementations for Q-Q plots with a uniform distribution. You may explore this before the class.

```
qqnorm(random_numbers, pch = 1, frame = FALSE)
qqline(random_numbers, col = "steelblue", lwd = 2)
```



LCG <- function(n, a, c, m, seed) { state <- seed random_numbers <- vector(length = n) for(i in 1:n){ state <- (a * state + c) %% m random_numbers[i] <- state

} return (random_numbers) } random_numbers <- LCG(n = 10, a = 1103515245, c = 12345, m = 2^31, seed = 5400)

qqplot(qunif(ppoints(10)), random_numbers, main = "Q-Q Plot: LCG vs Uniform(0, 1)", xlab = "Theoretical Quantiles", ylab = "Sample Quantiles")

```
#### 3. LaTeX \& ‘Tikz‘ library I
+ Read the document latex.pdf in Tech Guide on the ICON site. Try to generate latex.pdf using the tex c
+ Use LaTeX to produce a pdf with the following table.

[](tab.png){width=250px}‘‘‘

+ Generate a very simple plot in R and save it as a pdf or eps. Include it in your tex code and thus in

+ Instead of saving the plot as a pdf or eps, try the following R code before the plot is produced by R
```

{r tikz, include=TRUE, eval=TRUE}

library(tikzDevice) tikz('myplot.tex', width=5, height=5) plot(sin, -pi, 2*pi)

~~~
+ Add `\usepackage{tikz}` in your tex code (before `\begin{document}`) and include the tikz-produced plo

No need to submit the pdf but attach all of your tex code in your R Markdown file.
```{include=TRUE, eval=FALSE}
\documentclass[14pt, letterpaper]{homework}
\title{My first LaTeX document}
\usepackage{graphicx}
\author{Moiyyad Sufi}
\date{September 2024}
\begin{document}
    \begin{center}
        \begin{tabular}{|c c|}
            \hline
             Name & PDF  \\
             \hline
             Gamma & $p({y}|\alpha,\beta)=\frac{\beta^\alpha}{\Gamma(\alpha)} y^{(\alpha-1)}exp(-\beta y
            \hline


        \end{tabular}

        \begin{figure}[h]
            \centering
            \includegraphics[width=0.5\textwidth]{sin_plot.pdf} % Use the file path of your plot
            \caption{A simple plot created in R}
        \end{figure}
    \end{center}
\end{document}

copy your tex code here.
~~~

""

4. LaTeX & Tikz library II

- Use the tikz package in latex to reproduce the following plot.

Below are some code you may begin with, although you do not have to use them.

```
% Define Shapes
\tikzstyle{block} = [rectangle, draw, fill=blue!20,
text width=10em, text centered, rounded corners, minimum height=5em]
\tikzstyle{line} = [draw, very thick, color=black!80, -latex']
\tikzstyle{cloud} = [draw, ellipse,fill=red!20, node distance=2.5cm,
minimum height=5em, text width=7em]

\begin{tikzpicture}[scale=0.5, auto]
```

```
% Place nodes
\node [block] (pop) {population};
\node [block, right of=pop, node distance=7cm] (rs) {data (sample) \\ \textit{$\mathbf{X} = (X_1, X_2, 
\node [cloud, below of=pop, node distance=4cm] (param) {\begin{center}parameter \\$\theta$ \end{center}

\node [cloud, below of=rs, node distance=4cm] (stat) {\begin{center}statistic \\$\mathbf{T(X)}$ \end{ce

% Draw edges
\path [line] (pop) -- (rs) node[pos=0.5, above, align=left] {sampling};
\path [line] (pop) -- (param) node[pos=0.5, right, align=right] {characterization};
\path [line] (stat) -- (param) node[pos=0.5, below, align=left] {statistical inference};
\path [line] (rs) -- (stat) node[pos=0.5, right, align=left] {function $\mathbf{T}$};

\end{tikzpicture}
```

5. (Coding practice) Regrouping problem Suppose 30 numbers are grouped into six groups with group sizes 5, 5, 5, 5, 5, 5:

```
(oldgroup <- split(1:30, rep(1:6, rep(5, 6))))
```

```
## $`1`
## [1] 1 2 3 4 5
##
## $`2`
## [1]  6  7  8  9 10
##
## $`3`
## [1] 11 12 13 14 15
##
## $`4`
## [1] 16 17 18 19 20
##
## $`5`
## [1] 21 22 23 24 25
##
## $`6`
## [1] 26 27 28 29 30
```

Write an R function 'regroup" to randomly form these numbers in new groups. It is required that (1) the group sizes, i.e., (5, 5, 5, 5, 5, 5), are unchanged and (2) any old group partners are not in the same group again.

```
old_grp <- split(1:30, rep(1:6, rep(5, 6)))

 regroup <- function(grp) {
  set.seed(5400)  # Setting seed for reproducibility

  # Flatten the old groups into a single vector
  all_elements <- unlist(grp)

  # Function to check if any group has common elements with old groups
```

```r
  no_common_elements <- function(new_grp, old_grp) {
    for (i in seq_along(new_grp)) {
      if (any(new_grp[[i]] %in% old_grp[[i]])) {
        return(FALSE)
      }
    }
    return(TRUE)
  }

  # Generate random groups until no common elements with old groups
  repeat {
    shuffled <- sample(all_elements)  # Shuffle the elements randomly

    # Split the shuffled elements back into groups of size 5
    new_grp <- split(shuffled, rep(1:6, each = 5))

    # Check if any old partners are in the same group
    if (no_common_elements(new_grp, grp)) {
      return(new_grp)  # Return the new group if condition is met
    }
  }
}

 regroup(old_grp)
```

```
## $`1`
## 22 33 54 32 61
##  7 13 24 12 26
##
## $`2`
## 52 63 31 43 15
## 22 28 11 18  5
##
## $`3`
## 21 65 51 24 55
##  6 30 21  9 25
##
## $`4`
## 62 53 25 64 14
## 27 23 10 29  4
##
## $`5`
## 34 11 42 13 12
## 14  1 17  3  2
##
## $`6`
## 41 35 23 44 45
## 16 15  8 19 20
```

```r
'Sample Output: regroup(old_grp)
[[1]]
[1]  2 10 12 17 21
```

```
[[2]]
[1] 28  3  6 11 16

[[3]]
[1] 23 26  1  8 14

[[4]]
[1] 18 25 27  5  9

[[5]]
[1] 15 19 24 30  4

[[6]]
[1]  7 13 20 22 29'
```

```
## [1] "Sample Output: regroup(old_grp)\n[[1]]\n[1]  2 10 12 17 21\n\n[[2]]\n[1] 28  3  6 11 16\n\n[[3]]
```