Lab 4: Simulation

Justin Pelak

Spring 2025

Packages needed: knitr, xtable, pander

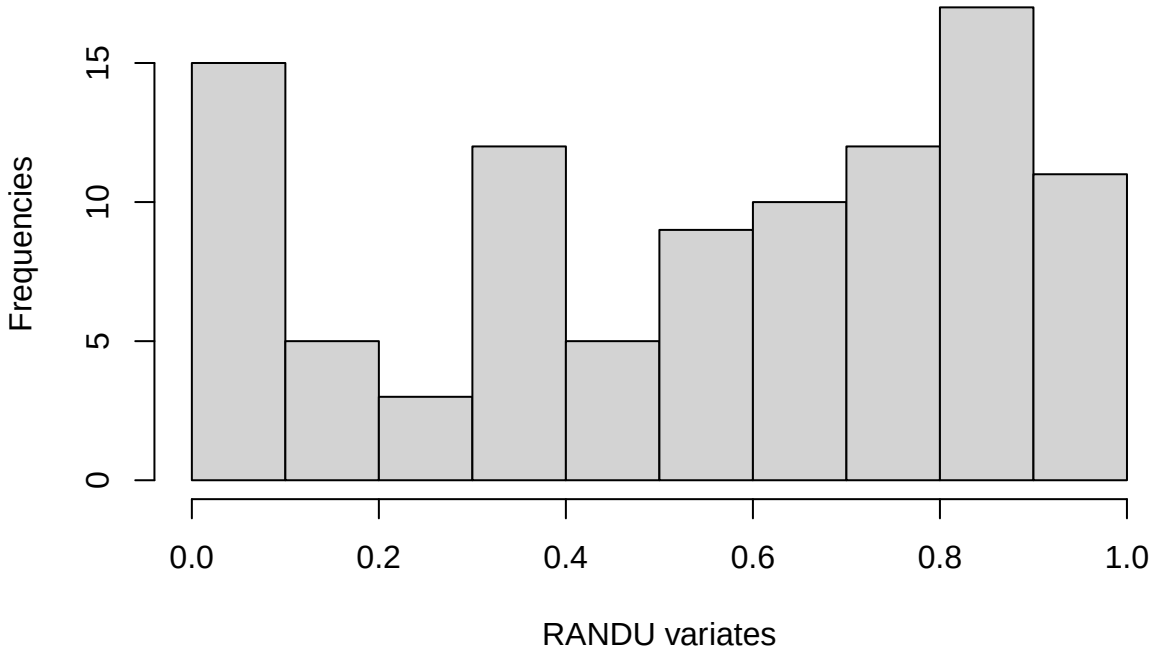
Task 1: Evaluation of a standard random number generator

Code set-up

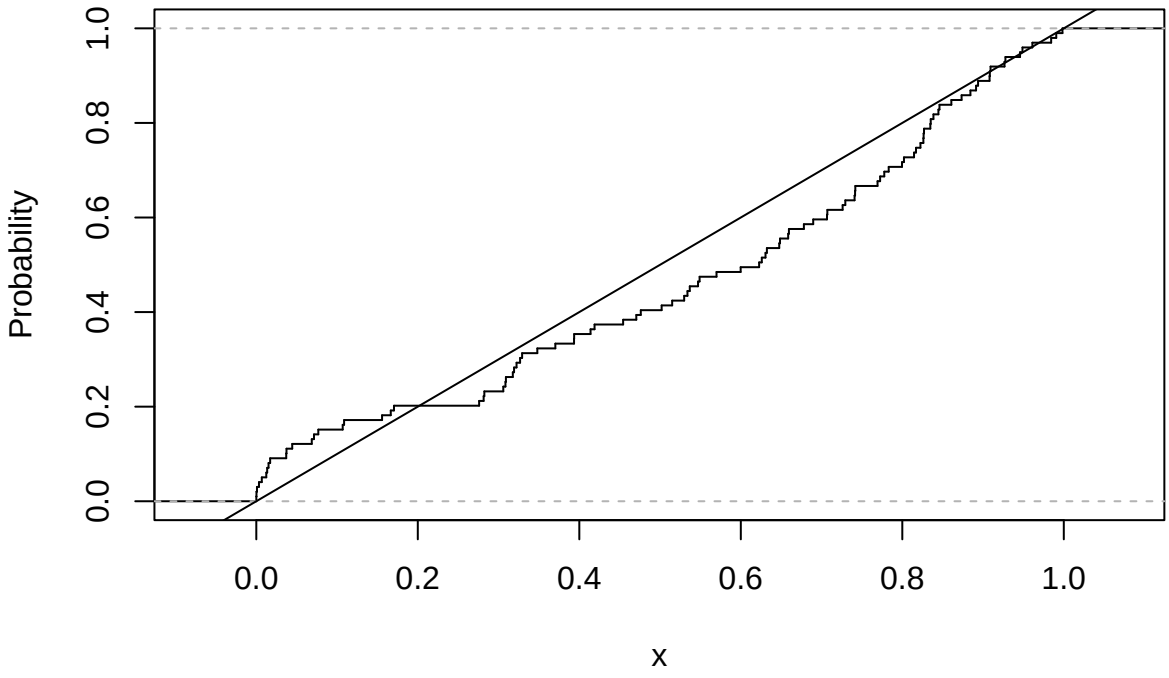
The code chunk below presents R code for the RANDU random number generator presented in the video lectures. The code chunk generates 100 pseudo-random numbers from using the RANDU generator and presents a histogram, empirical cdf, and Kolmogorov-Smirnov test to evaluate the performance of the algorithm. Try running it.

```
# generate random variates according to the RANDU generator
# X_{n+1} = 65539 X_n mod 2^31
n<-100 # number of variates
x<-1 # seed
for(i in 1:n){
   x<-c(x, (65539*x[i])%%(2^31))
}
x<-x/2^31
x<-x[2:n]

#par(mfrow=c(2,1))
# histogram of the n RANDU variates
hist(x, main="", xlab="RANDU variates", ylab="Frequencies")
```




```
# empirical distribution function of the n RANDU variates and
# uniform(0, 1) probability plot
plot.ecdf(x, verticals= TRUE, do.p = FALSE, main="", ylab="Probability")
abline(0,1)
```




```
# Kolmogorov-Smirnov test of RANDU variates against U(0, 1) distribution
ks.test(x,"punif",0,1)
```

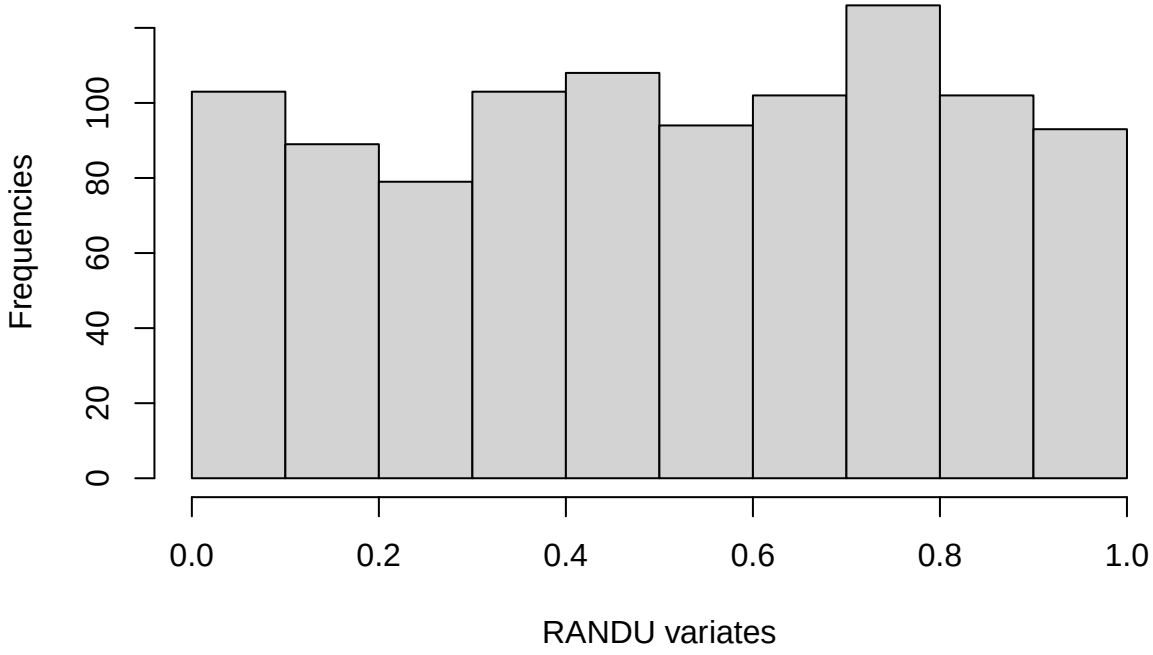
```
##
##  Exact one-sample Kolmogorov-Smirnov test
##
## data:  x
## D = 0.12773, p-value = 0.07229
## alternative hypothesis: two-sided
```

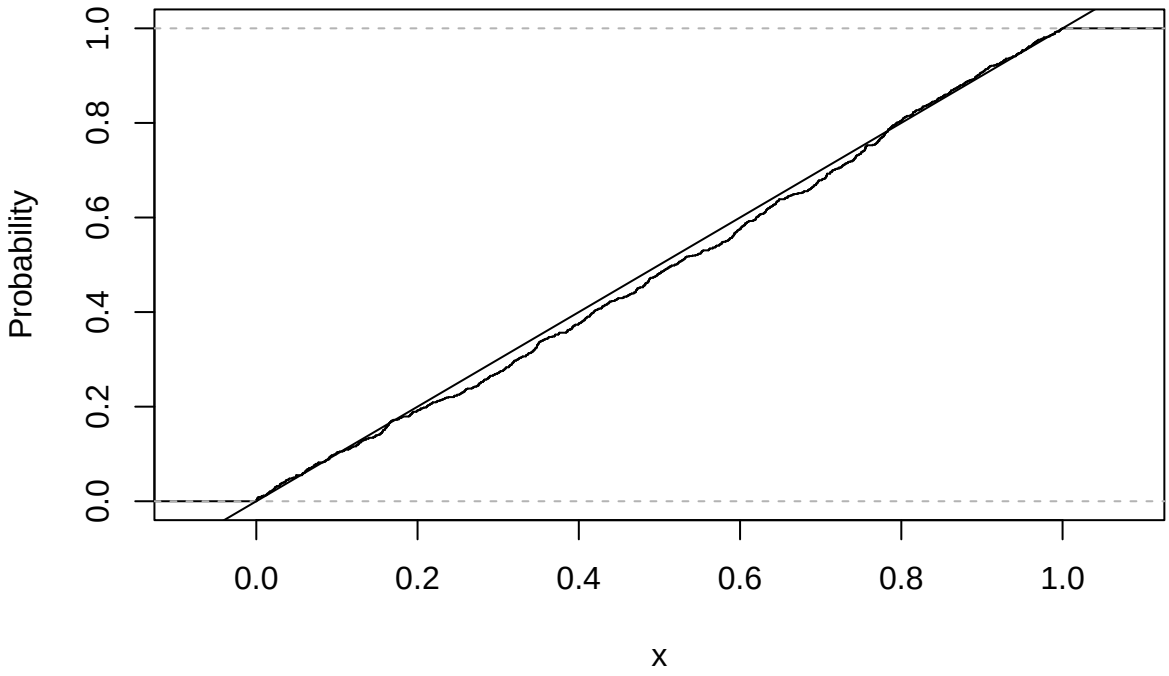
The problem

In this task, we will evaluate an original implementation of the `rand` command in *Matlab* for generating uniform random numbers. The random number generator has formulation

$$X_{n+1} = 16807 X_n mod(2^{31} - 1).$$

Ammend the code chunk above to generate 1000 random uniform variates from this random number generator.


Frequencies
0
20
40
60
80
100
0.0
0.2
0.4
0.6
0.8
1.0
RANDU variates




```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  x
## D = 0.034188, p-value = 0.1934
## alternative hypothesis: two-sided
```

Report the following: (these are all outputs of the code chunk above already)

- Present a histogram of the 1000 variates you generated.
- Present the empirical distribution function of your data and a probability plot under the assumption of independent and identically distributed data points (on the same set of axes as we did in the online lecture).
- Use your generations from (a) to perform a Kolmogorov-Smirnov test to validate the routine.

Questions:

- Do the pseudo-random numbers appear uniformly distributed? Why or why not?

The numbers do appear to be roughly uniform, because in the histogram, we see the frequencies appear to be nearly equal, and in the empirical distribution function, the probabilities fall nearly on a straight line.

- What are your conclusions from the Kolmogorov-Smirnov test at the $\alpha = 0.05$ level?

Because we have a low D value and a p-value of .19 (>0.05), we fail to reject that the sample does not come from a uniform distribution.

Task 2: Simulation of discrete distributions

A bag contains one red, two blue, three green, and four yellow marbles. A sample of three marbles is taken without replacement. Let B denote the number of blue marbles and Y denote the number of yellow marbles in the sample. The probabilities of each outcome is as follows:

```
# we will create a table using xtable and pander
library(knitr)
library(xtable)
library(pander)
table.elts = rbind(
      c(0, "4/120", "24/120", "24/120", "4/120", "56/120"),
      c(1, "12/120", "32/120", "12/120", "0/120", "56/120"),
      c(2, "4/120", "4/120", "0/120", "0/120", "8/120"),
      c("Yellow marg", "20/120", "60/120", "36/120", "4/120", "1"))
#row.names(table.elts) = cbind("# coin tosses", "Number of heads", "Proportion of heads")
colnames(table.elts) = cbind("B/Y", "0", "1", "2", "3", "Blue marg")
lab3.table = xtable(table.elts, caption = "Discrete distribution of marble game.", label="distr_table",
pander(lab3.table, hline.after = c(3))
```

Table 1: Discrete distribution of marble game.

B/Y	0	1	2	3	Blue marg
0	4/120	24/120	24/120	4/120	56/120
1	12/120	32/120	12/120	0/120	56/120
2	4/120	4/120	0/120	0/120	8/120
Yellow marg	20/120	60/120	36/120	4/120	1

Code set-up

We can use the `sample` function of R to randomly generate from any sequence, including characters. For example, we can code the bag of marbles as

```
pop = c("r","b","b","g","g","g","y","y","y","y")
```

We may then sample from this bag of letters (marbles). The following code chunk draws three marbles (so one run of the experiment). We record the number of blue marbles and yellow marbles from this draw.

```
# True values:
# P(0 blue) = 56/120; P(1 blue) = 56/120; P(2 blue) = 8/120
# P(0 Y) = 20/120; P(1 Y) = 60/120; P(2 Y) = 36/120; P(3 Y) = 4/120

# Bag of marbles (population)
pop = c("r","b","b","g","g","g","y","y","y","y")
samp = sample(pop,3) # draw 3 marbles
blues = sum(samp=="b") # number of blue marbles drawn
yellows = sum(samp=="y") # number of yellow marbles drawn
c(blues, yellows) # just checking how many blue and yellow marbles are drawn
```

```
## [1] 1 1
```

```
print(sprintf("%d blues %d yellows", blues, yellows))
```

```
## [1] "1 blues 1 yellows"
```

For this task, you will repeat this experiment 10,000 times. You still need to store the number of blue and yellow marbles from each experiment. The easiest way is to just turn the `blue` and `yellow` variables into vectors in which to store the values each step of a for-loop over the 10,000 experiments.

A note, the `table` command in R is a handy way of summarizing output. In particular, if you have a storage vector `blues` of the number of blue marbles for each of 10,000 experiments, `table(blues)` will tabulate the number of experiments with 0 blue marbles drawn, 1 blue marble drawn, 2 blue marbles drawn, and 3 blue marbles drawn.

The problem

Perform a simulation experiment of drawing three marbles from the bag 10,000 times. I have initialized storage vectors for you.

```
# True values:
# P(0 blue) = 56/120; P(1 blue) = 56/120; P(2 blue) = 8/120
# P(0 Y) = 20/120; P(1 Y) = 60/120; P(2 Y) = 36/120; P(3 Y) = 4/120
simnum = 10000 # number of experiments
# Initialize storage vectors for the number of blue and yellow marbles drawn
# for each of the 10,000 runs of the experiment.
blues=numeric(simnum); yellows=numeric(simnum)

for (i in 1:simnum) {
  samp = sample(pop, 3)
  blues[i] = sum(samp=="b")
  yellows[i] = sum(samp=="y")
}
```

Reporting and questions

It is up to you how you want to present the values requested. Rather than getting into R Markdown tables (`xtable` and `pander`) for this lab report, I recommend using R data frames (`data.frame` command). The command `setNames` allows you to rename the columns of a data frame for purposes of clear labeling.

By empirical pmf (probability mass function), we mean the proportion of experiments in which we draw 0 marbles of the given color, 1 marble of the given color, 2 marbles of the given color, and 3 marbles of the given color. So if B is a random variable denoting the number of blue marbles drawn, the empirical pmf is an estimate of $P(B = 0)$, $P(B = 1)$, $P(B = 2)$, and $P(B = 3)$.

- Present the empirical pmf of the number of blue marbles drawn.

```
epmf_blue = prop.table(table(factor(blues, levels=0:3)))
df_blue = as.data.frame(epmf_blue)
colnames(df_blue) = c("Blue Count", "Probability")
df_blue$`Blue Count` = sprintf("P(B=%d)",c(0:3))
print(df_blue)
```

```
##   Blue Count Probability
## 1     P(B=0)      0.4704
## 2     P(B=1)      0.4604
## 3     P(B=2)      0.0692
## 4     P(B=3)      0.0000
```

- Present the empirical pmf of the number of yellow marbles drawn.

```
epmf_yellow = prop.table(table(yellows))
df_yellow = as.data.frame(epmf_yellow)
colnames(df_yellow) = c("Yellow Count", "Probability")
df_yellow$`Yellow Count` = sprintf("P(Y=%d)",c(0:3))
print(df_yellow)
```

```
##   Yellow Count Probability
```

```
## 1       P(Y=0)      0.1606
## 2       P(Y=1)      0.5058
## 3       P(Y=2)      0.3010
## 4       P(Y=3)      0.0326
```

- Present the mean and variance of the number of blue marbles drawn.

```
print(sprintf("Blue: mean = %f, variance = %f", mean(blues), var(blues)))
```

```
## [1] "Blue: mean = 0.598800, variance = 0.378676"
```

- Present the mean and variance of the number of yellow marbles drawn.

```
print(sprintf("Yellow: mean = %f, variance = %f", mean(yellows), var(yellows)))
```

```
## [1] "Yellow: mean = 1.205600, variance = 0.549784"
```

- Compare the empirical pmf with the true values.

```
df = data.frame(num_marbles=c(0,1,2,3))
df$blue_diff = sprintf("%.3g", c(56/120, 56/120, 8/120, 0) - df_blue$Probability)
df$yellow_diff = sprintf("%.3g", c(20/120, 60/120, 36/120, 4/120) - df_yellow$Probability)
print(df)
```

```
##   num_marbles blue_diff yellow_diff
## 1           0  -0.00373     0.00607
## 2           1   0.00627     -0.0058
## 3           2  -0.00253      -0.001
## 4           3         0    0.000733
```

The differences in proportions from our simulation marbles and the true proportions, as shown in the table above, are very small, with the largest difference between below 0.01.