Testing Unif(0,1) Random Number Generators

Project Group 218 – (Members (1) : Anhtuan Dang Ho)

17. (1–2 members) Test some Unif(0,1) random number generators. There are a lot of uniform random number generators around. How do we know if a particular generator is doing a good job? In this project, you should test various generators to see if they are actually producing numbers that are approximately independent, identically distributed uniforms. The tests you should use are formal hypothesis tests such as χ^2 goodness-of-fit tests, the von Neumann test for independence, runs tests, etc. You should also comment on the run-time efficiency of the generators; i.e., which generators yield the most random numbers per unit time? This could be a really good project for a group interested in everything from statistics to graphical analysis of data.

Abstract

Simulations of a system in which there are intrinsically random elements require a method of generating or accessing numbers that are random. For example, if you wanted to simulate a queuing model, you might need to obtain random interarrival and service times that are drawn from a specified probability distribution such as an exponential distribution. This is where the **uniform distribution** on the interval [0,1] comes into play: random variates from all other distributions, such as Erlang, normal, triangular, and realizations of various random processes can be obtained by **transforming** independent and identically distributed (IID) random Unif(0,1) numbers that are spawned from random number generators (RNGs). Thus, these random numbers created in computer simulation software are actually **deterministic**, and we can only believe that they *appear* as if they were truly IID Unif(0,1).

Background and Description of Problem

For this project, I will conduct some empirical tests to assess how closely the random numbers generated from various popular RNGs resemble truly IID Unif(0,1). I have chosen 3 criteria that will be assessed for a "good enough" RNG. A good RNG, at a **bare minimum**, should have the following traits:

1. Can generate values that are i.i.d. Unif(0,1)

2. Very fast

3. Reproducible

I have selected 3 RNGs: **RANDU, Desert Island, and L'ecuyer Combined Generator**, and will assess criteria (1) for each of them. After that, I will discuss criteria (2) and (3). But first let's import all the necessary libraries needed to conduct the relevant analysis.

Linear Congruential Generators¶

Most of the RNGs used today are **Linear Congruential Generators** (LCG) where a sequence Z_1, Z_2 , ... is defined by the recursive formula:

$$\mathbf{Z_i = (aZ_{i-1} + c)(mod\ m)}$$

where m (the modulus), a (the multiplier), c (the increment), and Z_0 (the seed) are nonnegative integers. Simply put, in order to obtain Z_i, divide $aZ_{i-1} + c$ by m and let Z_i be the remainder of this division.

So, $0 \leq Z_i \leq m - 1$, and to obtain the desired random numbers U_i (for i = 1, 2, . . .) on [0, 1], we let $U_i = Z_i/m$. Thus, we must **carefully choose a, c, m, and Z_i** to yield an LCG that generates random numbers that appear to be IID Unif(0,1).

It's also worth mentioning that there is a **looping** mechanism: whenever Z_i takes on a value it previously had, the same sequence of values will be generated and the cycle endlessly repeats, and the length of this cycle is called a **period**. An LCG has a **full period** if its period is equal to m. Having a full (or at least a long) period is a beneficial property of an LCG.

I have defined a function below that implements the LCG formula and takes in parameters: **m, a, and the number of random numbers to generate**. Notice how I didn't include **c**. This is because the LCGs that I am focusing on (RANDU and Desert Island) simply do not have a ***c*** parameter(these are special types of LCGs called **multiplicative LCGs**)

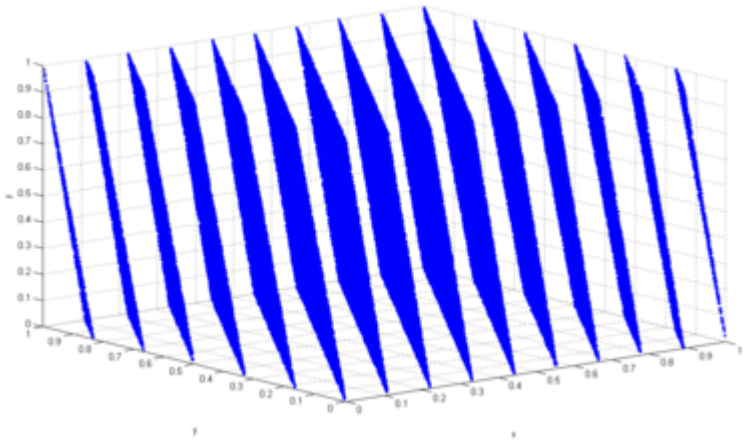
Main Findings

RANDU

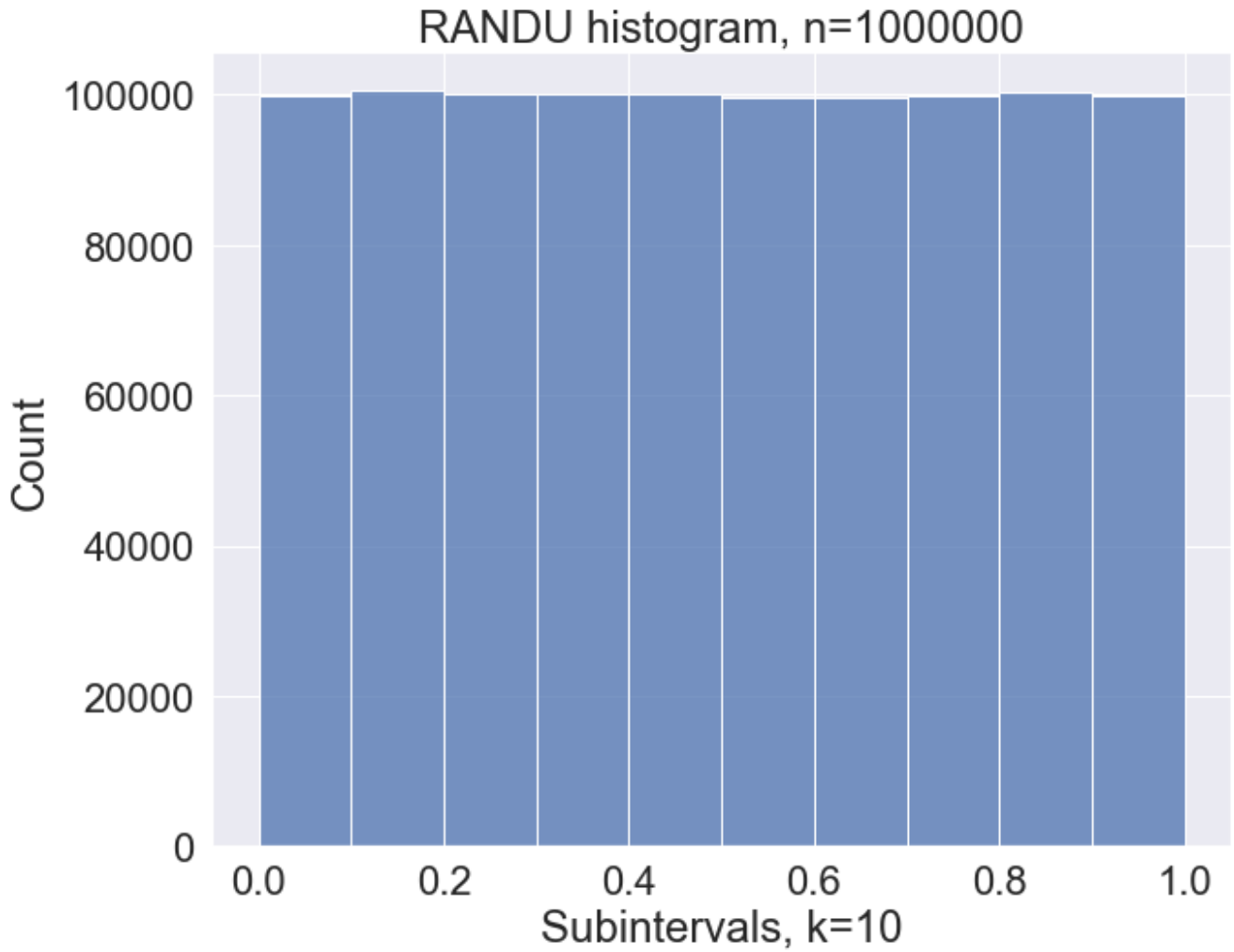
The first LCG I will test is called **RANDU**, which was a widely-used LCG in the 1960s and 1970s. Its formula is:

$$Z_i = (65539Z_{i-1})(\text{mod } 2^{31})$$

Today, RANDU is a poorly conceived LCG and is no longer used because it fails the **spectral test**, which is another statistical test that can be used to assess the quality of an RNG. If you were to make a 3-D plot of 100,000 numbers with RANDU, in which each point represents 3 consecutive values, all of the points fall onto 15 distinct 2-D planes. Below is the plot:



However, I wish to assess its quality via other statistical tests to see how "bad" it really is. Maybe it's not as bad as people think. Let's generate 1 million pseudorandom numbers (PRNs) using RANDU and make a histogram and see if we can visualize if they are uniformly distributed between 0 and 1.



Above is a histogram that counts how many of the 1 million psuedorandom numbers are in each subinterval. For the sake of simplicity, I have selected **10 equal-probable subintervals, k=10**. Thus, with 1 million PRNs and 10 subintervals, we would expect that the probability that an observation would fall into subinterval i is 1/10 with each bin ranging from [0, 1/10), [1/10, 2/10),..., [9/10, 1].

Below is a dataframe that compares the actual number of observations vs the expected number of observations that fall into each subinterval.

	Actual Number of Observations	Expected Number of Observations
[0.0, 0.1)	99815	100000
[0.1, 0.2)	100561	100000
[0.2, 0.3)	100032	100000
[0.3, 0.4)	100009	100000
[0.4, 0.5)	100074	100000
[0.5, 0.6)	99693	100000
[0.6, 0.7)	99723	100000
[0.7, 0.8)	99882	100000
[0.8, 0.9)	100420	100000
[0.9, 1.0)	99790	100000

At first glance, these observations appear to be uniformly distributed between 0 and 1. But how do we know for sure? Fortunately, we can conduct **two types of empirical tests** to statistically examine how closely they resemble true IID Unif (0,1) numbers, which is assessing the first criteria aforementioned (1).

Empirical Tests (Testing Criteria (1))

Chisquared Goodness of Fit

The first type of empirical test is a chisquared goodness-of-fit test, which will be used to determine if the PRNs generated appear to be uniformly distributed between 0 and 1. Here's the formal test setup:

H_0: $R_1, R_2, ...R_n \sim$ Unif(0,1)

H_0: $R_1, R_2, ...R_n$ *are not* Unif(0,1)

We consider the null hypothesis as the status quo, so we'll only reject the null if there's sufficient evidence against it. (Innocent until proven guilty.) That is, the null hypothesis assumes that the PRNs resemble true Unif(0,1)s. I will set the ***level of significance***, α **= P(RejectH0|H0 true) = P(Type I error) = 0.05**

1. Divide [0,1] into k subintervals. If you choose equi-probable cells [0, 1/k), [1/k, 2/k),...,[k-1/k, 1], then a particular observation R_i will fall in a particular cell with probability 1/k

2. Tally how many of the n observations fall into the k cells. If O_i = the # of R_j's in subinterval i, then (since the R_j's are i.i.d.), we can easily see that

   $O_i \sim Bin(n, 1/k), \quad i = 1,2,...,k.$

3. Thus, the expected number of R_j's to fall in subinterval i will be $E_i = E[O_i] = n/k, \ i = 1,2,...,k.$

Runs Test

The second type of empirical test is a runs test, which is used to determine if a dataset comes from a random process or if it follows a pattern by considering runs of data. This directly tests the ***independence and identically distributed assumption*** from criteria (1). A run is defined as a sequence of increasing or decreasing values, and the number of increasing or decreasing values is the length of a run. There are various ways of defining a positive and negative runs but for our purposes, any value that is **greater than the mean (> 0.50)** is a *positive* value and any value that is **less than the mean (<0.50)** is a *negative* value. Here's the formal test setup:

> **H_0: The data was produced in a random manner**
>
> **H_a: The data was *not* produced in a random manner**

Again, we consider the null hypothesis as the status quo, so we'll only reject the null if there's sufficient evidence against it. That is, the null hypothesis assumes that the PRNs were generated in a random manner. I will set the *level of significance*, α **= P(RejectH0|H0 true) = P(Type I error) = 0.05.**

Testing RANDU

Now it's time to apply these two tests on RANDU. Also, I wish to see how the number of subintervals for the chisquared goodness of fit test affects the test statistic. For example, will we draw different conclusions for **10 subintervals vs 100 subintervals vs 1000 subintervals?**

> **Note #1:** I will basically conduct a chisquared goodness of fit test and runs **test 3 times each, one for each number of subintervals of 10, 100, 1000.** Because the number of subintervals has nothing to do with the runs test statistic, I predict that there won't be any noticeable differences among the various number of subintervals.
>
> **Note #2:** For each number of subintervals (10, 100, 1000), I will conduct a chisquared goodness of fit test **100 times** and a runs test **100 times**, and use the **average** of their test statistics along with the corresponding p-values. Even if RANDU appears to be "good", the p-value will be below the level of significance, 0.05, in about 5% of my test. Thus, I believe it's necessary to do as many trials as possible before making any conclusions.

Number of Cells	Chisquare Critical Value	Chisquare Test Statistic	Chisquare p-value	Reject Chisquare GOF test?	Runs Test Critical Value	Runs Test Statistic	Runs p-value	Reject Runs Test for Independence?
10	16.918978	8.532178	0.529848	Fail to reject	1.959964	-0.057306	0.476909	Fail to reject
100	123.225221	98.230352	0.509929	Fail to reject	1.959964	-0.029921	0.541078	Fail to reject
1000	1073.642651	992.145760	0.529371	Fail to reject	1.959964	0.073675	0.505958	Fail to reject

RANDU Test Results

Chisquared Goodness of Fit

The relevant chisquared critical values for k = [10, 100, 1000] are (16.9190, 123.2252, 1073.6427) and the corresponding chisquared test statistics and p-values are (8.5322, 98.2304, 992.1458) and (0.5298, 0.5099, 0.5294), respectively. Since all of chisquared test statistics are less than the critical values and the p-values are greater than 0.05, we **fail to reject the null hypothesis** and have sufficient evidence that the PRNs generated from RANDU are uniformly distributed between 0 and 1.

Runs Test for Independence

The relevant runs test critical value for k = [10, 100, 1000] is 1.96 and the corresponding runs test statistics and p-values are (-0.0573, -0.0299, 0.0737) and (0.4769, 0.5410, 0.5060), respectively. Since all of the absolute value of the runs test statistics are less than the critical values and the p-values are greater than 0.05, we **fail to reject the null hypothesis** and have sufficient evidence that the PRNs generated from RANDU were produced in a random manner and are independent.

Conclusion

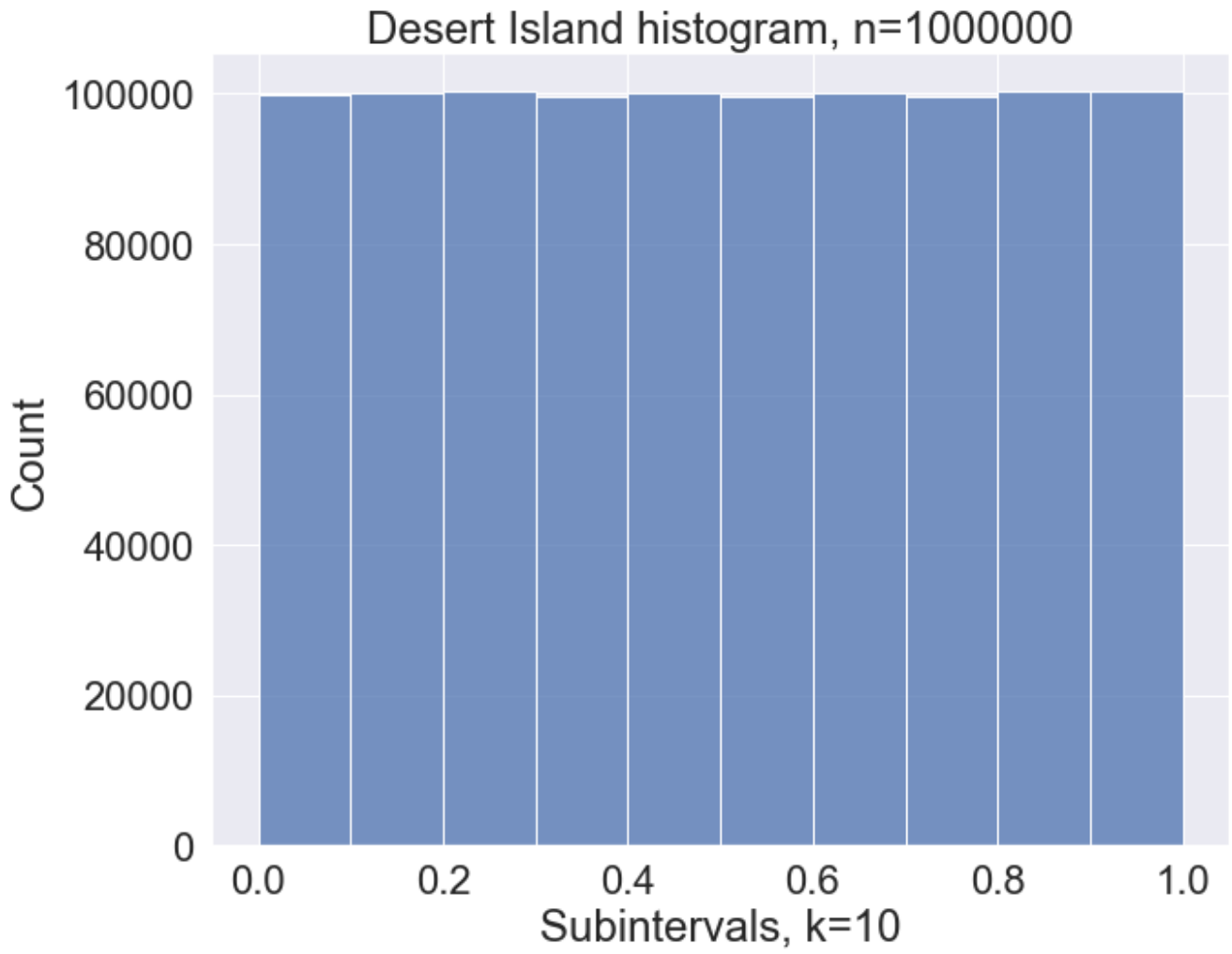
RANDU passed both empirical tests so we can safely say that it satisfies criteria (1) in that RANDU can generate values that are IID Unif(0,1). Notice how the average p-value for each number of subintervals is around 0.50 for both the chisquared goodness of fit test **and** the runs test, which effectively nullifies any Type 1 errors. Also, it seems as though the number of subintervals don't really matter. Could this be a coincidence? Maybe.

Desert Island

The second LCG I will test is called **Desert Island**. Its formula is:

$$Z_i = (16807Z_{i-1})(\text{mod } 2^{31}-1)$$

It's full period and its cycle length is > 2 billion. Let's generate 1 million of these.



	Actual Number of Observations	Expected Number of Observations
[0.0, 0.1)	99866	100000
[0.1, 0.2)	100197	100000
[0.2, 0.3)	100217	100000
[0.3, 0.4)	99577	100000
[0.4, 0.5)	100058	100000
[0.5, 0.6)	99684	100000
[0.6, 0.7)	100134	100000
[0.7, 0.8)	99708	100000
[0.8, 0.9)	100259	100000
[0.9, 1.0)	100299	100000

Similar to RANDU, these observations ***appear*** to be uniformly distributed between 0 and 1, but we don't know for sure. Thus, let's conduct the same empirical tests (with the exactly the same test setup) to assess the first criteria (1).

Number of Cells	Chisquare Critical Value	Chisquare Test Statistic	Chisquare p-value	Reject Chisquare GOF test?	Runs Test Critical Value	Runs Test Statistic	Runs p-value	Reject Runs Test for Independence?
10	16.918978	8.726542	0.518515	Fail to reject	1.959964	-0.019235	0.464996	Fail to reject
100	123.225221	97.558150	0.545480	Fail to reject	1.959964	-0.034886	0.528516	Fail to reject
1000	1073.642651	998.234020	0.493610	Fail to reject	1.959964	0.049668	0.399953	Fail to reject

Desert Island Test Results

Chisquared Goodness of Fit

The relevant chisquared critical values for k = [10, 100, 1000] are (16.9190, 123.2252, 1073.6427) and the corresponding chisquared test statistics and p-values are (8.7265, 97.5582, 998.2340) and (0.5185, 0.5455, 0.4936), respectively. Since all of chisquared test statistics are less than the critical values and the p-values are greater than 0.05, we **fail to reject the null hypothesis** and have sufficient evidence that the PRNs generated from Desert Island are uniformly distributed between 0 and 1.

Runs Test for Independence

The relevant runs test critical value for k = [10, 100, 1000] is 1.96 and the corresponding runs test statistics and p-values are (-0.0192, -0.0348, 0.0497) and (0.4650, 0.5285, 0.400), respectively. Since all of the absolute value of the runs test statistics are less than the critical values and the p-values are greater than 0.05, we **fail to reject the null hypothesis** and have sufficient evidence that the PRNs generated from Desert Island were produced in a random manner and are independent.

Conclusion

Desert Island passed both empirical tests so we can safely say that it satisfies criteria (1) in that Desert Island can generate values that are IID Unif(0,1). Similar to RANDU, notice how the average p-value for each number of subintervals is around 0.50 for both the chisquared goodness of fit test **and** the runs test, which effectively nullifies any Type 1 errors. Could this be a coincidence? Maybe.

L'ecuyer Combined Generator

One interesting feature of LCG's is that you can combine them to create a more robust LCG, as mathematician Pierre L'ecuyer did in 1999. In simplified form, the idea is to let Z_{1i} and Z_{2i} denote the integer sequences generated by two different LCGs with different moduli, then let $Z_i = (Z_{1i} - Z_{2i})(\text{mod } m)$ for some integer m, and finally set $U_i = Z_i/m$. It has a cycle length of about 2^{191}. Here's the official formula:

Initialize $X_{1,0}$ $X_{1,1}$, $X_{1,2}$, $X_{2,0}$, $X_{2,1}$, $X_{2,2}$. For $i \geq 3$, set

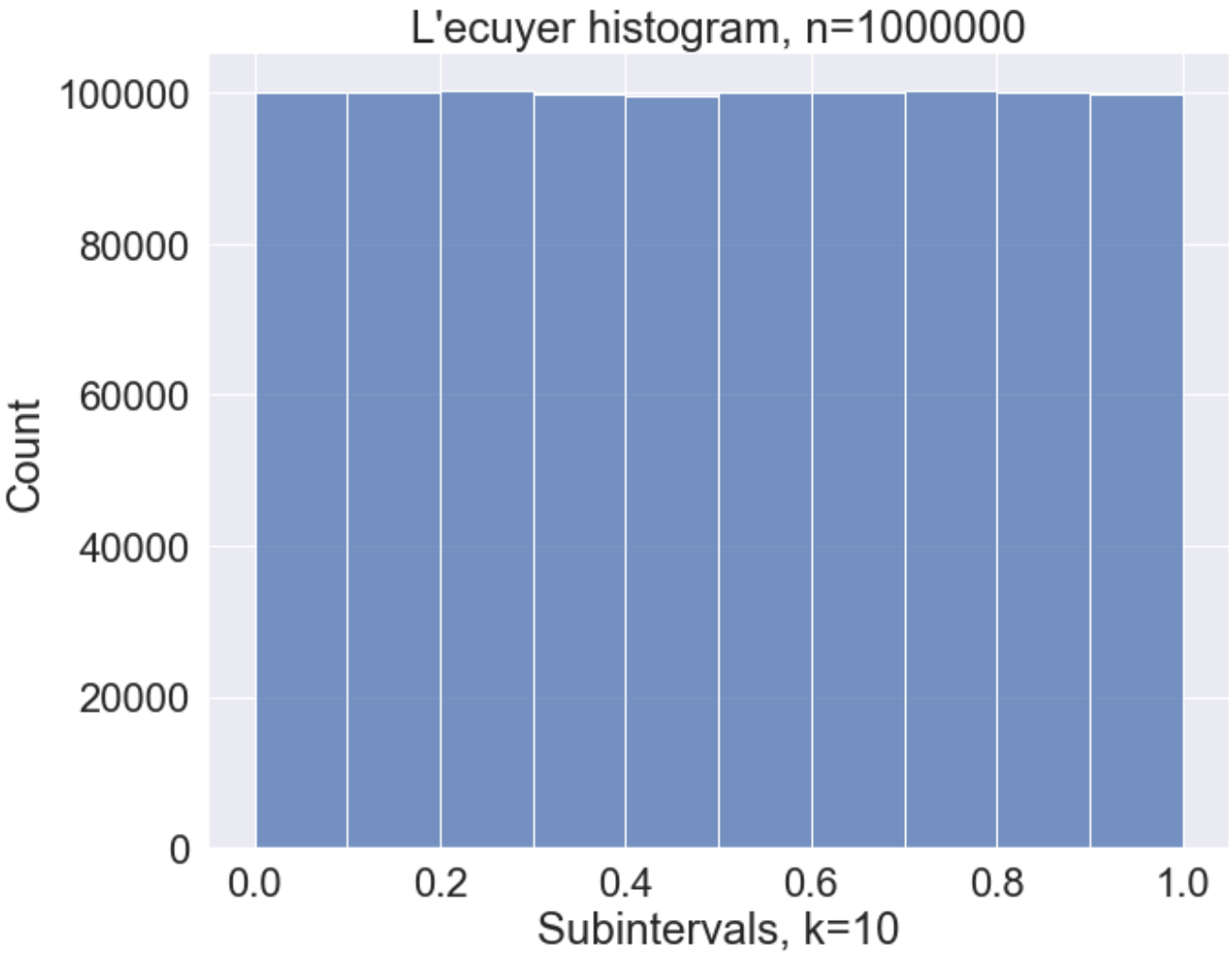
$$X_{1,i} = (1{,}403{,}580X_{1,i-2} - 810{,}728X_{1,i-3}) \text{mod}(2^{32} - 209)$$

$$X_{2,i} = (527{,}612X_{2,i-1} - 1{,}370{,}589X_{2,i-3}) \text{mod}(2^{32} - 22{,}853)$$

$$Y_i = (X_{1,i} - X_{2,i}) \text{mod}(2^{32} - 209)$$

$$R_i = Y_i/(2^{32} - 209)$$

I have defined a function below that implements the L'ecuyer formula and it only takes in one parameter: number of random numbers to generate. Also, note that running the function always yields a different initialization of 6, 5 digit numbers. Let's generate 1 million of these.


L'ecuyer histogram, n=1000000
Count
100000
80000
60000
40000
20000
0
0.0
0.2
0.4
0.6
0.8
1.0
Subintervals, k=10

	Actual Number of Observations	Expected Number of Observations
[0.0, 0.1)	100119	100000
[0.1, 0.2)	100041	100000
[0.2, 0.3)	100277	100000
[0.3, 0.4)	99863	100000
[0.4, 0.5)	99549	100000
[0.5, 0.6)	100090	100000
[0.6, 0.7)	100141	100000
[0.7, 0.8)	100212	100000
[0.8, 0.9)	99952	100000
[0.9, 1.0)	99756	100000

Similar to RANDU and Desert Island, these observations ***appear*** to be uniformly distributed between 0 and 1, but we don't know for sure. Thus, let's conduct the same empirical tests (with the exactly the same test setup) to assess the first criteria (1).

Number of Cells	Chisquare Critical Value	Chisquare Test Statistic	Chisquare p-value	Reject Chisquare GOF test?	Runs Test Critical Value	Runs Test Statistic	Runs p-value	Reject Runs Test for Independence?
10	16.918978	8.808582	0.504573	Fail to reject	1.959964	-0.041607	0.489585	Fail to reject
100	123.225221	100.561100	0.467876	Fail to reject	1.959964	0.205540	0.541422	Fail to reject
1000	1073.642651	995.308120	0.523781	Fail to reject	1.959964	0.010012	0.473580	Fail to reject

L'ecuyer Combined Generator Test Results¶

Chisquared Goodness of Fit

The relevant chisquared critical values for k = [10, 100, 1000] are (16.9190, 123.2252, 1073.6427) and the corresponding chisquared test statistics and p-values are (8.8086, 100.5611, 995.3081) and (0.5046, 0.4679, 0.5238), respectively. Since all of chisquared test statistics are less than the critical values and the p-values are greater than 0.05, we **fail to reject the null hypothesis** and have sufficient evidence that the PRNs generated from L'ecuyer are uniformly distributed between 0 and 1.

Runs Test for Independence

The relevant runs test critical value for k = [10, 100, 1000] is 1.96 and the corresponding runs test statistics and p-values are (-0.0416, 0.2055, 0.0100) and (0.4896, 0.5414, 0.4736), respectively. Since all of the absolute value of the runs test statistics are less than the critical values and the p-values are greater than 0.05, we **fail to reject the null hypothesis** and have sufficient evidence that the PRNs generated from L'ecuyer were produced in a random manner and are independent.

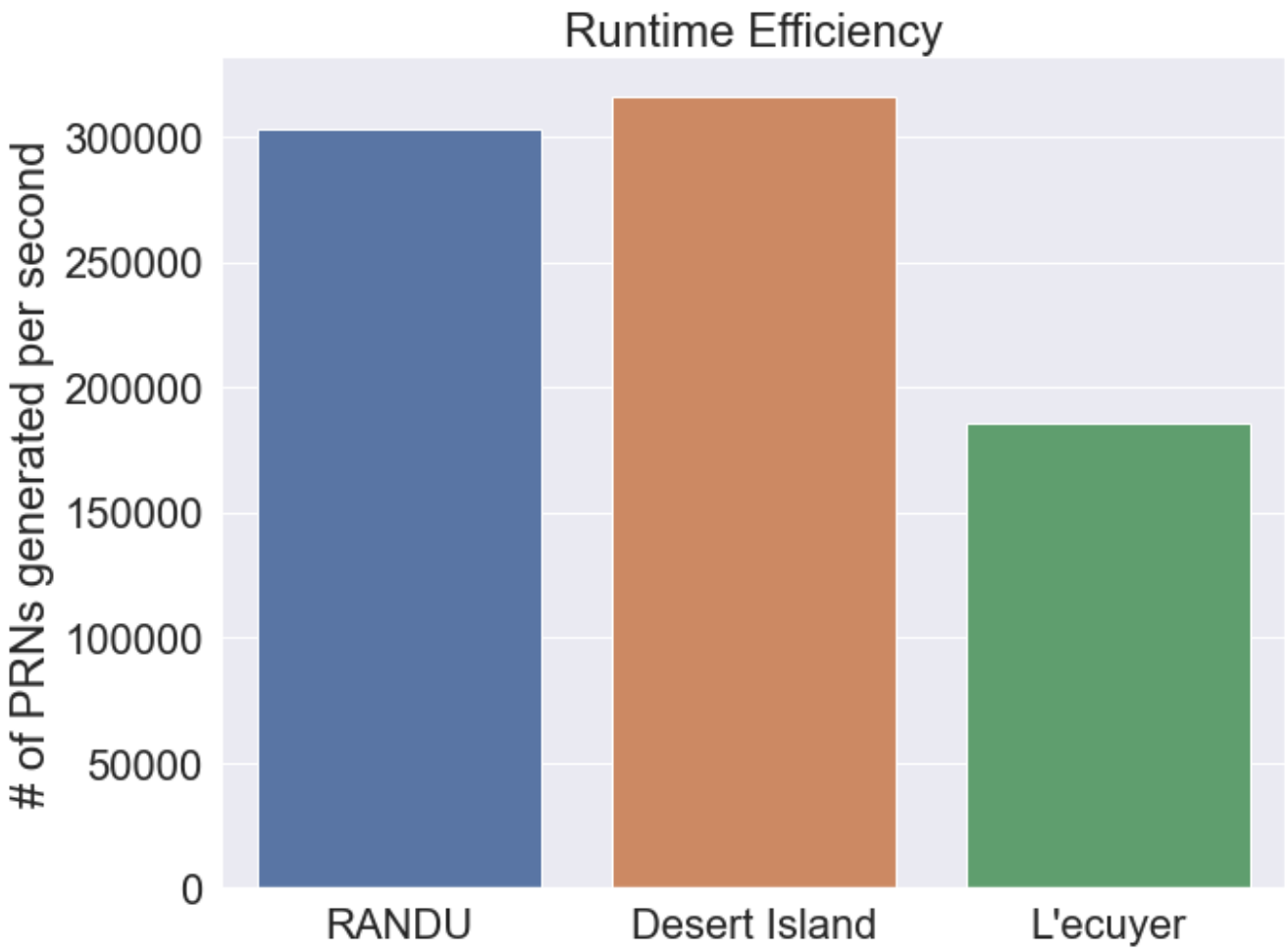
Conclusion

L'ecuyer passed both empirical tests so we can safely say that it satisfies criteria (1) in that L'ecuyer can generate values that are IID Unif(0,1). Similar to RANDU and Desert Island, notice how the average p-value for each number of subintervals is around 0.50 for both the chisquared goodness of fit test and the runs test, which effectively nullifies any Type 1 errors. Could this be a coincidence? Maybe.

Runtime efficiency (Criteria 2)

Another important aspect of an RNG is its runtime efficiency. When simulating a system, not only is it critical to obtain numbers that appear to be random, but it's also beneficial to obtain a lot of them since you may require millions (or even more!) numbers at a time. Let's see how many PRNs each generator produces **per second**.

	# PRNs generated per second
RANDU	303369.222913
Desert Island	315960.363634
L'ecuyer	185529.680695



RANDU generated **303,369** PRNs per second, Desert Island generated **315,960** PRNs per second, and L'ecuyer generated **185,529** PRNs per second, which makes **Desert Island** the most runtime efficient of the 3. Desert Island has a slight edge over RANDU, generating **12,591** more PRNs per second, which is equivalently **4.1% faster**. It's also worth noting that L'ecuyer can only generate **a little bit more than half** as many PRNs as the other two generators. This makes sense because L'ecuyer is a combination of ***exactly 2 LCG's***. So while L'ecuyer has the longest cycle length, it is the least runtime efficient generator.

Reproducibility (Criteria 3)

We would like to reproduce any given stream of random numbers for **2 main reasons**: the first reason is that it makes **debugging** easier, and the second reason is we might want to use ***identical*** streams of numbers when simulating two different systems to get a more precise comparison. For any generator, ensuring reproducibility is simple: we only have to remember the **starting seed used prior, Z_0**, and initiate the generator with this value again to obtain the same sequence of numbers.

Conclusion

Because random numbers used in computer systems are actually completely deterministic, the best thing we can do is hope that they appear to be truly IID Unif(0,1), which can be evaluated by conducting a chisquared goodness of fit test and a runs test. RANDU, Desert Island and L'ecuyer all passed these tests with flying colors, even though RANDU fails the spectral test. I also found it really interesting how all average p-values for each generator hovered around the 0.500 mark, which teased out any Type 1 errors and suggests that the number of subintervals for the chisquared of goodness of fit test is not that important. It could be a coincidence but further analysis is needed.

Desert Island was the most efficient generator, and any stream of random numbers can be reproduced as long as the same starting seed is used. If I were to do this project again, I would choose other popular RNGs out there, such as the **Mersenne Twister**, which has an astounding cycle length of $2^{19937}-1$, and use another goodness of fit test such as the **Kolmogorov–Smirnov test**.

References

ISYE OMSA-6644: Simulation and Modeling for Engineering and Science (Dr. David Goldsman)

Law, A. M., Simulation Modeling and Analysis, 5th edition, McGraw-Hill Education, New York, 2015.