Anthony Poerio (adp59@pitt.edu)
CS1538 – Assignment #2, Report
February 14, 2016

Random Number Generation

Generating truly random numbers is a longstanding problem in math, statistics, and computer science. True randomness requires true entropy, and in many applications—such as generating very large sets of random numbers very quickly—sufficient "true" entropy is difficult or impractical to obtain. (Often, it needs to come from the physical environment, sources such as radioactive decay, etc.) So, instead, we look to algorithmic **random number generators** for help.

Algorithmically generated random numbers will never be "truly" random precisely because they are generated with a repeatable algorithmic formula. But for purposes such as simulating random events – these "Pseudo-random" numbers can be sufficient. These algorithmic generators take a "seed value" from the environment, or from a user, and use this seed as a variable in their formula to generate as many random-like numbers as a user would like.

Naturally, some of these algorithms are better than others, and hundreds (if not thousands, or more) of them have been designed over the years. The output is always deterministic, and never "truly" random, but the ideal goal is to *approximate randomness* by generating numbers which:

1. Are uniformly distributed on the range of [0,1)
2. Are statistically independent of each other
   a. (That is, the outcomes of any given sequence do not rely on previously generated numbers)

The best random number generators will pass statistical tests for **both** uniformity and independence. In this analysis, we will subject three different random number generation algorithms to series of statistical tests and compare the outcomes.

The algorithms we will test are:

- Python's Built-In Random Number Generator
  - This algorithm is called the "Mersenne Twister", implementation details are available at: Python Docs for Random
  - Seed value: 123456789
- A Linear Congruential Generator
  - Seed value: 123456789
  - a=101427
  - c=m21
  - $m=2^{16}$
- A Linear Congruential Generator with RANDU initial settings

    - Seed value: 123456789
    - a=65539
    - c=0
    - m=2^{31}

The tests each algorithm will be subjected to are:

- **Uniformity tests**
    - Chi-squared Test for Uniformity
    - Kolmogorov-Smirnov Test for Uniformity
        - Null hypothesis for BOTH tests: The numbers in our data set **are** uniformly distributed
- **Independence tests**
    - Runs Test for Independence
    - Autocorrelation Test for Independence, (gap sizes: 2,3,5, and 5 will be used)
        - Null hypothesis for BOTH tests: The numbers in our data set **are** independent of each other

Test Result Summary

The exact implementation of each test can be viewed in the attached Python file named: “**lcg.py**”. The tests can be duplicated by anyone with Python installed on their system by running the command “*python lcg.py*”.

SUMMARY TABLE (2.1)

	X^2	**KS**	**Runs**	**Autocorrelation**
Algorithm	**TS/Result**	**TS/Result**	**TS/ Result**	**TS/Result**

Mersenne Twister (PyRand)	9.754
	Pass all
		0.079213
		Pass all
			-1.6605
			Reject at 0.8 and 0.9
			Pass at 0.95
				Gap=2: -1.089
				Gap=3: -0.924
				Gap=5: -0.559
				Gap=50: -0.228
				All Pass

LGC	6.288
	Pass all
		0.0537888
		Pass all
			0.5218
			Pass all
				Gap=2: 0.1753
				Gap=3: -0.394
				Gap=5: 0.872
				Gap=50: 1.335
				Reject at 0.8, Gap=50, others pass

LGC, w/ RANDU	12.336
	Pass all
		0.053579
		Pass all
			-0.2135
			Pass all
				Gap=2: -0.6591
				Gap=3: -0.9311
				Gap=5: 0.5788
				Gap=50:-0.9093
				All pass

The summary table above shows each algorithm tested, and which tests were passed or failed. More detailed output for each test and for each algorithm can be viewed in *Tables 1.1 – 1.3* in the appendix to this document.

The **Kolmogorov-Smirnov (or KS test)** was run at the following levels of significance: .90, 0.95, 0.99. The formulas for the critical value at these significance levels were taken from *table of A7* of Discrete-Event System Simulation by **Jerry Banks and John S. Carson II**. (Formulas for 0.80 could not be found, so I've used what was available.)

All other tests were run at the 0.80, 0.90., and 0.95 significance level.

Statistical Analysis and Expectations

Prior to generating the numbers for each test, I expected Python's random function to perform the best of all three algorithms tested, mostly because it's the library random function of one of the world's most popular programming languages. (Which means: thousands and thousands of code repositories rely on it—many of which are used by commercial and mission critical programs.) But in fact, it performed the worst, failing the **Runs Test** at both the 0.80 and 0.90 level of significance. These failings are NOT statistically significant at the alpha=0.05 level, but it's still surprising to see.

I expected the RANDU algorithm to perform the worst, and I thought it would perform especially badly on the autocorrelation test. This is because RANDU is known to have problems, outlined here. Specifically, it is known to produce values which fall along only a specific set of parallel planes (visualization in link above), which means the numbers should NOT be independent, when tested at the right gap lengths. It's possible that the gap lengths I've tested simply missed any of these planes, and as a result—RANDU performed the best of all the algorithms. It's the only algorithm that didn't fail *any* statistical tests at all.

I anticipated the LGC function to perform 2nd best overall, and I was right about that—but the best and worst algorithm were the opposite of what I expected. Mostly, I thought that that Python's random generator would be nearly perfect, RANDU would be badly flawed, and the LGC would be just okay. Really, the LGC performed admirably: The only test it failed was autocorrelation at the 0.80 confidence level, and that isn't statistically significant by most measures.

Reviewing the data output into each .txt file directly, I don't see any discernible patterns in the numbers themselves. And with 10,000 data points, there's so much output to review that I can see why statistical measures are needed to effectively to determine what's really going on in the data. It's probably possible to find a few patterns, specifically related to runs and gap-sequences just by viewing the data directly, but tests are still needed to find out for sure.

With that said, I do **think the testing done in this experiment is sufficient**, because we have two tests for each measure that matters: 1) Uniformity; 2) Independence. The only improvement I would make for future tests is testing more gap-sequences, and starting them at different points. I'd do this mostly because I know that RANDU should fail gap-sequence tests given the right input, but there would be some trial and error involved in trying to find these sequences naively. So, sometimes, getting into math itself and working with proofs may still be the most effective method. Maybe sometime the old-fashioned way is still best.

TABLE 1.1 - Mersenne Twister (Python's Random Function, with seed: X_0 = 123456789)

Test Name	**Sample Size**	**Confidence Level**	**Critical Value**	**Test Statistic Found**	**Result**
Chi-Square	10,000	0.80	10118.8246	9.754	FAIL TO REJECT null
Chi-Square	10,000	0.90	10181.6616	9.754	FAIL TO REJECT null
Chi-Square	10,000	0.95	10233.7489	9.754	FAIL TO REJECT null
Kolmogorov-Smirnov	100	0.90	0.122	0.079213	FAIL TO REJECT null
Kolmogorov-Smirnov	100	0.95	0.136	0.079213	FAIL TO REJECT null
Kolmogorov-Smirnov	100	0.99	0.163	0.079213	FAIL TO REJECT null
Runs Test	10,000	0.80	1.282	-1.6605	**REJECT null**
Runs Test	10,000	0.90	1.645	-1.6605	**REJECT null**
Runs Test	10,000	0.95	1.96	-1.6604	FAIL TO REJECT null
Autocorrelation, GapSize=2	10,000	0.80	1.282	-1.089	FAIL TO REJECT null
Autocorrelation, GapSize=2	10,000	0.90	1.645	-1.089	FAIL TO REJECT null
Autocorrelation, GapSize=2	10,000	0.95	1.96	-1.089	FAIL TO REJECT null
Autocorrelation, GapSize=3	10,000	0.80	1.282	-0.92484	FAIL TO REJECT null
Autocorrelation, GapSize=3	10,000	0.90	1.645	-0.92484	FAIL TO REJECT null

Autocorrelation, GapSize=3	10,000	0.95	1.96	-0.92484	FAIL TO REJECT null
Autocorrelation, GapSize=5	10,000	0.80	1.282	-0.55929	FAIL TO REJECT null
Autocorrelation, GapSize=5	10,000	0.90	1.645	-0.55929	FAIL TO REJECT null
Autocorrelation, GapSize=5	10,000	0.95	1.96	-0.55929	FAIL TO REJECT null
Autocorrelation, GapSize=50	10,000	0.80	1.282	-0.227805	FAIL TO REJECT null
Autocorrelation, GapSize=50	10,000	0.90	1.645	-0.227805	FAIL TO REJECT null
Autocorrelation, GapSize=50	10,000	0.95	1.96	-0.227805	FAIL TO REJECT null
Test Name	**Sample Size**	**Confidence Level**	**Critical Value**	**Test Statistic Found**	**Result**

END TABLE 1.1

TABLE 1.2 – Linear Congruential Generator (X_0 = 123456789)

Test Name	Sample Size	Confidence Level	Critical Value	Test Statistic Found	Result
Chi-Square	10,000	0.80	10118.8246	6.288	FAIL TO REJECT null
Chi-Square	10,000	0.90	10181.6616	6.288	FAIL TO REJECT null
Chi-Square	10,000	0.95	10233.7489	6.288	FAIL TO REJECT null
Kolmogorov-Smirnov	100	0.90	0.122	0.05378	FAIL TO REJECT null
Kolmogorov-Smirnov	100	0.95	0.136	0.05378	FAIL TO REJECT null
Kolmogorov-Smirnov	100	0.99	0.163	0.05378	FAIL TO REJECT null
Runs Test	10,000	0.80	1.282	0.521889	FAIL TO REJECT null
Runs Test	10,000	0.90	1.645	0.521889	FAIL TO REJECT null
Runs Test	10,000	0.95	1.96	0.521889	FAIL TO REJECT null
Autocorrelation, GapSize=2	10,000	0.80	1.282	0.175377	FAIL TO REJECT null
Autocorrelation, GapSize=2	10,000	0.90	1.645	0.1753777	FAIL TO REJECT null
Autocorrelation, GapSize=2	10,000	0.95	1.96	0.1753777	FAIL TO REJECT null
Autocorrelation, GapSize=3	10,000	0.8	1.282	-0.39487	FAIL TO REJECT null

Autocorrelation, GapSize=3	10,000	0.9	1.645	-0.39487	FAIL TO REJECT null
Autocorrelation, GapSize=3	10,000	0.95	1.96	-0.39487	FAIL TO REJECT null
Autocorrelation, GapSize=5	10,000	0.8	1.282	0.872668	FAIL TO REJECT null
Autocorrelation, GapSize=5	10,000	0.9	1.645	0.872668	FAIL TO REJECT null
Autocorrelation, GapSize=5	10,000	0.95	1.96	0.872668	FAIL TO REJECT null
Autocorrelation, GapSize=50	10,000	0.8	1.282	1.3352	**REJECT null**
Autocorrelation, GapSize=50	10,000	0.9	1.645	1.3352	FAIL TO REJECT null
Autocorrelation, GapSize=50	10,000	0.95	1.96	1.3352	FAIL TO REJECT null
Test Name	**Sample Size**	**Confidence Level**	**Critical Value**	**Test Statistic Found**	**Result**

END TABLE 1.2

TABLE 1.3 – Linear Congruential Generator with RANDU initial settings

Test Name	Sample Size	Confidence Level	Critical Value	Test Statistic Found	Result
Chi-Square	10,000	0.80	10118.8246	12.336	FAIL TO REJECT null
Chi-Square	10,000	0.90	10181.6616	12.336	FAIL TO REJECT null
Chi-Square	10,000	0.95	10233.7489	12.336	FAIL TO REJECT null
Kolmogorov-Smirnov	100	0.90	0.122	0.053579	FAIL TO REJECT null
Kolmogorov-Smirnov	100	0.95	0.136	0.053579	FAIL TO REJECT null
Kolmogorov-Smirnov	100	0.99	0.163	0.053579	FAIL TO REJECT null
Runs Test	10,000	0.8	1.282	-0.21350	FAIL TO REJECT null
Runs Test	10,000	0.9	1.645	-0.21350	FAIL TO REJECT null
Runs Test	10,000	0.95	1.96	-0.21350	FAIL TO REJECT null
Autocorrelation, GapSize=2	10,000	0.8	1.282	-0.65918	FAIL TO REJECT null
Autocorrelation, GapSize=2	10,000	0.9	1.645	-0.65918	FAIL TO REJECT null
Autocorrelation, GapSize=2	10,000	0.95	1.96	-0.65918	FAIL TO REJECT null
Autocorrelation, GapSize=3	10,000	0.8	1.282	-0.93113	FAIL TO REJECT null

Autocorrelation, GapSize=3	10,000	0.9	1.645	-0.93113	FAIL TO REJECT null
Autocorrelation, GapSize=3	10,000	0.95	1.96	-0.93113	FAIL TO REJECT null
Autocorrelation, GapSize=5	10,000	0.80	1.282	0.378881	FAIL TO REJECT null
Autocorrelation, GapSize=5	10,000	0.90	1.645	0.378881	FAIL TO REJECT null
Autocorrelation, GapSize=5	10,000	0.95	1.96	0.378881	FAIL TO REJECT null
Autocorrelation, GapSize=50	10,000	0.80	1.282	-0.90937	FAIL TO REJECT null
Autocorrelation, GapSize=50	10,000	0.90	1.645	-0.90937	FAIL TO REJECT null
Autocorrelation, GapSize=50	10,000	0.95	1.96	-0.90937	FAIL TO REJECT null
Test Name	**Sample Size**	**Confidence Level**	**Critical Value**	**Test Statistic Found**	**Result**

END TABLE 1.3

Table 2.1 – Test Result Comparisons

	X^2	KS	Runs	Autocorrelation
Algorithm	**TS/Result**	**TS/Result**	**TS/ Result**	**TS/Result**

Mersenne Twister (PyRand)	9.754
	Pass all
		0.079213
		Pass all
			-1.6605
			Reject at 0.8 and 0.9
			Pass at 0.95
				Gap=2: -1.089
				Gap=3: -0.924
				Gap=5: -0.559
				Gap=50: -0.228
				All Pass

LGC	6.288
	Pass all
		0.0537888
		Pass all
			0.5218
			Pass all
				Gap=2: 0.1753
				Gap=3: -0.394
				Gap=5: 0.872
				Gap=50: 1.335
				Reject at 0.8, Gap=50, others pass

LGC, w/ RANDU	12.336
	Pass all
		0.053579
		Pass all
			-0.2135
			Pass all
				Gap=2: -0.6591
				Gap=3: -0.9311
				Gap=5: 0.5788
				Gap=50:-0.9093
				All pass