Implementation of the Tausworthe Random Number Generator in R

Evan Velasco
Georgia Institute of Technology
evelasco6@gatech.edu
Group 234

Abstract

Random numbers are a fundamental component of Monte Carlo experiments, simulations, and a wide variety of other computational techniques. There exist many modern methods which use deterministic algorithms to create pseudo-random numbers. This paper analyzes the implementation of the Tausworthe pseudo-random number generator in R. Unlike previous implementations, we will create a function which gives a user full control over the algorithm's parameters. Special attention is given to statistical analysis of the results generated from the algorithm to confirm their validity. Finally, we show the potential to extend our Tausworthe generator to simulate other distributions by implementing the Box-Muller method for generating Nor(0,1) observations.

1 Introduction

This paper explores the nuanced landscape of generating Uniform(0,1) pseudo-random numbers. Despite its conceptual simplicity, the implementation of pseudo-random number generation is crucial to obtaining usable results. The desire to generate a random outcome has been prevalent across civilization for ages. An early example can be found during the Roman Empire when a coin was flipped to generate independent random bits[1]. Over time a variety of methods arose involving dice, manual imputations, and physical devices. While many of these were successful to some degree in generating randomness, there was still much to be desired. The process to generate the numbers was usually quite tedious making it difficult to scale. Depending on the method it would also be difficult to reproduce sequences of numbers[1]. As computers grew in power, the notion of algorithms to generate pseudo-random numbers (commonly referred to as PRNs) arose. The term pseudo-random refers to the fact that these numbers are created through a deterministic algorithm but appear to be random. This paper will review the theory and implementation of one such algorithm known as the Tausworthe random number generator. The generator will be put to a variety of statistical tests to ensure it meets the criteria for statistically relevant pseudo-random numbers. That is to say the distribution of the PRNs is uniform and the observations are independent of one another. Finally, we will show how we can extend the usefulness of our PRNs by transforming them into Nor(0,1) observations via the Box-Muller method.

2 Motivation

PRNs have a wide variety of use cases in statistics, simulation, and other computational methods. One of the most powerful features of PRNs is the ability to transform values from Uniform(0,1) to a chosen distribution. This can be done through manipulations such as the inverse transform theorem or convolution. That means if we can successfully generate Uniform(0,1) PRNs, we have access to generating values from a wide variety of potential distributions. Having access to simulated data from a chosen distribution is an essential component to Monte Carlo experiments. Other use cases include using simulated data from a Exp(λ) distribution to model the interarrival times of a Poisson process. Validation methods such as k fold cross-validation can use PRNs to split the data into relevant folds. For many of these use cases, PRNs are not only an acceptable method to use but are in fact preferred. The true power of PRNs comes in the ability to replicate the random numbers used in a model. This property is instrumental when sharing results or validating others' approaches.

3 The Tausworthe Generator

Before we generate any PRNs using the Tausworthe generator, we will discuss the underlying theory behind how it works. Afterwards, we will review past attempts at creating the algorithm in R and how our method will differ. Finally, we will detail the implementation of our algorithm and give an example of calling the function in R.

3.1 Theory

To create our Tausworthe generator, we will define a sequence of binary digits $B_1, B_2, ..., B_n$ by

$$B_i = \left(\sum_{j=1}^{q} c_j B_{i-j} \right) \mod 2$$

where $c_j = 0$ or 1. We will use the following implementation in order to save computational power.

$$B_i = (B_{i-r} + B_{i-q}) \mod 2 \quad (1)$$
$$B_i = B_{i-r} \text{ XOR } B_{i-q} \quad (0 < r < q) \quad (2)$$

One of the most desirable properties of Tausworthe generators is that they have periods of length $2^q - 1$. A period is defined as the time it takes before numbers begin to repeat themselves. To illustrate this, we will create a generator with parameters $r = 2$ and $q = 3$. Initializing $B_1 = B_2 = B_3 = 1$ we obtain the following sequence of random binary digits.

$$111010011101001...$$

We can see that after the first $2^3 - 1 = 7$ numbers, the pattern begins to repeat itself. In order to go from our binary digits B_i to PRNs, we will use (l bits in base 2) / 2^l and convert to base 10. By performing this operation on our previously generated binary numbers with $l = 3$ we obtain the following PRNs.

$$0.875, 0.250, 0.375, 0.625, 0.125, ...$$

Now that we have seen how to construct PRNs with the Tausworthe generator, we will discuss methods to choosing the parameters. Tausworthe states that in order to properly implement the generator we must find a primitive polynomial $f(x)$ over GF(2) [3]. The polynomial that is easiest to implement is the simplest form of a trinomial. For example, the degree 7 polynomial $f(x) = x^7 + x^3 + 1$ gives us the parameters $q = 7$ and $r = 4$ [2]. For the purposes of this paper, we will use the degree 31 primitive trinomial $f(x) = x^{31} + x^3 + 1$ and set $q = 31, r = 4,$ and $l = 20$ [5].

3.2 Existing Methods

Currently, the primary package for implementing a Tausworthe generator in R is called SynthRNG. The key functionality of this package is to generate PRNs via the Tausworthe generator which will be synchronous across both R and Python [4]. This is particularly important in use cases such as comparing statistical model performance where models are constructed in both R and Python. However, one limitation of this tool is that it does not allow a user to input custom choices for the parameters $q, r,$ and l. As we have discussed, these values must be carefully chosen to ensure proper PRN generation. Due to this reason, leaving this out of a user's hand is best in most cases. However, producing poor PRNs and analyzing the results can be a helpful way to understand what can go wrong with Tausworthe generators. When developing our own method, we will give a user full control over all parameters.

3.3 Implementation

To implement the Tausworthe random number generator in R, we will create a function titled *tausworthe* that takes in 5 parameters: $seed, n, r, q,$ and l. The following psuedo code outlines the function.

1. Set B = *seed*
2. Determine number of PRNs to generate using $n = l * q$
3. Add $n - q$ additional bits to B using the XOR method outlined in equation (2) from the theory subsection
4. Create sequence of indices to split bits on to get n instances of l bits
5. Convert l bits into base 10 and divide by 2^l
6. Return a vector of length n of uniform PRNs

While the above tasks are all relatively straightforward to complete in R, an issue arose during step 5 when converting l bits into base 10. Originally, this step was completed using the *strtoi*() function in base R which takes in a string and converts it to an integer of a specified base. When passing a large number of l bits to the function, R returned NA values. To remedy this issue, a helper function called $bitToBase10()$ was created which manually computed the conversion and did not return errors on high values. All that remains is to call our function with the desired parameters. For example, the function call $tausworthe(seed = "111", n = 5, r = 2, q = 3, l = 3)$ would return the following PRNs.

$$0.875, 0.250, 0.375, 0.625, 0.125$$

A number of additional functions were created in order to perform statistical tests for goodness of fit and independence. Code documentation for all functions can be found in Appendix A.

4 Results

From our Tausworthe generator we obtain 10,000 pseudo-random numbers using the following function $tausworthe(seed = "01011010010010100101101010111111", n = 5, r = 2, q = 3, l = 3)$. A second set of PRNs was also produced for later using the same parameters but with $seed =$ "11011110011111100111101110110000" for later use in the Box-Muller method. Before we can truly use these PRNs we need to test the quality of our generator's output. We will begin our investigation by performing a visual analysis

of the results. Any patterns or strange behavior we can see with the naked eye will be a strong indicator that there is something wrong. Following this, we will perform more formal statistical tests to check for uniformity and independence. Finally, after confirming no issues with our PRNs, we will show how we can use our Tausworthe generator to create observations from a Nor(0,1) distribution.

4.1 Visual Analysis

To begin our visual analysis, we will plot our Tausworthe PRNs by index. What we hope to see is a random cloud with no apparent patterns.

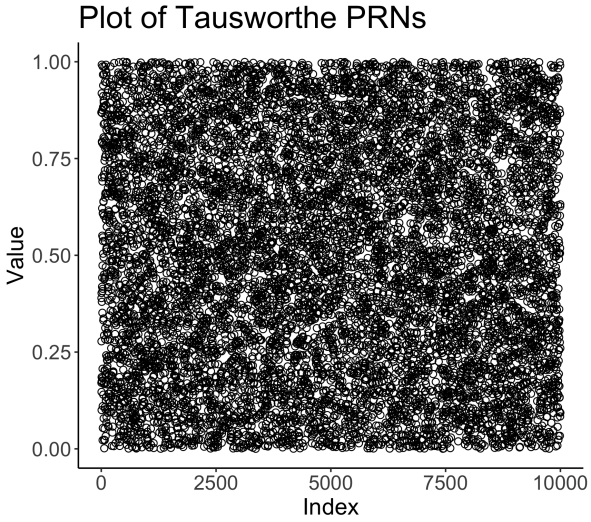

Figure 4.1: Plot of Tausworthe pseudo-random Numbers

The data in Figure 4.1 fits our visual test of randomness. We can also see that all values are bound by [0,1]. Next, we will look at the distribution of our PRNs.

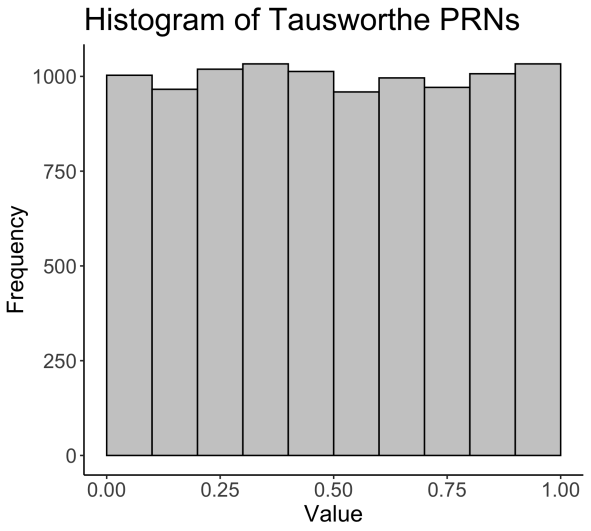

Figure 4.2: Histogram of Tausworthe pseudo-random Numbers

Figure 4.2 shows a histogram of our data. From visual analysis we see that the values roughly correspond to what we would expect from a Uniform(0,1) distribution. This is a good sign that our generator was successful. We will continue our visual analysis of the PRNs by plotting adjacent PRNs against one another. To accomplish this, we will plot the PRNs (U_i, U_{i+1}) on the unit square and (U_i, U_{i+1}, U_{i+2}) on the unit cube.

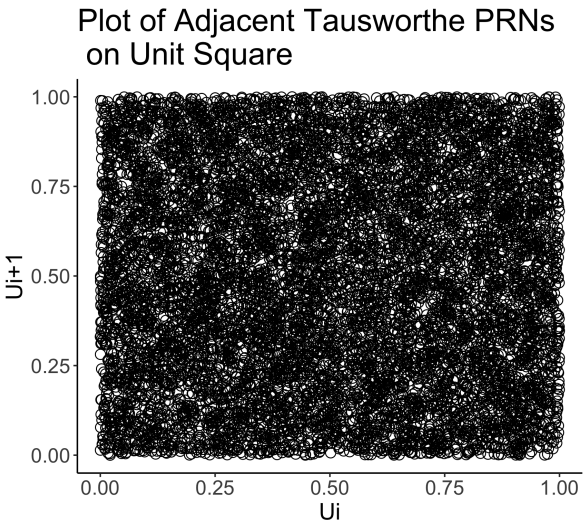

Figure 4.3: Plot of Adjacent Tausworthe pseudo-random Numbers in 2-dimensions

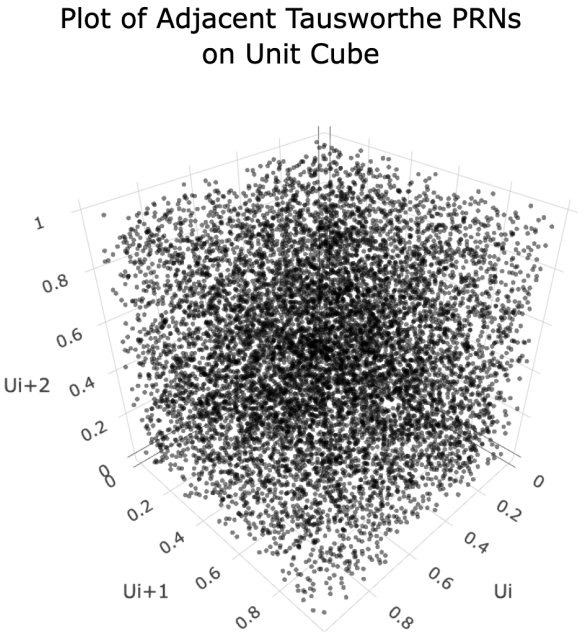

Figure 4.4: Plot of Adjacent Tausworthe pseudo-random Numbers in 3-dimensions

In Figure 4.3 and Figure 4.4 we see no strange behavior or patterns developed when analyzing the adjacent PRNs. For those wishing to see a more detailed view of Figure 4.4, please see the reference code in order to recreate the plot and pan to whatever view you would like. Based on a visual inspection, there are no glaring issues or patterns in our data.

4.2 Goodness of Fit Analysis

Now that our data has passed a visual inspection, we will perform more formal statistical tests to assess performance. To begin, we will perform a χ^2 Goodness of Fit test using $k = 5$ sub intervals. The respective hypothesis of this test will be

H_0: The observations are Uniform
H_A: The observations are Not Uniform

Recall that we can calculate our test statistic using

$$\chi_0^2 \equiv \sum_{i=1}^{k} \frac{(Oi - E_i)^2}{E_i}$$

and compare it to the appropriate $\chi^2_{a,k-1}$ quantile. Computing these values gives the following results for the χ^2 test at $a = 0.05$ level of significance.

χ_0^2	$\chi^2_{0.05,4}$	p-value
3.569	9.488	0.467

Figure 4.5: Results of χ^2 GOF test on PRN data

From the results shown in Figure 4.5 we fail to reject the null hypothesis, meaning we will continue with our assumption that the PRNs are Uniform(0,1).

4.3 Independence Analysis

Now that we have checked for uniformity, we will use a series of tests to check for independence. These tests are the Runs Up and Down, Runs Above and Below, and the Correlation Test. We will begin by defining each test statistic before computing all three tests and summarizing the results. All statistical tests in this subsection will be of the form

H_0: The observations are Independent
H_A: The observations are Not Independent

The Runs Up and Down test is defined as follows. Let A = total number of runs "up and down" out of n observations.

$$A \approx Nor\left(\frac{2n-1}{3}, \frac{16n-29}{90}\right)$$

We will compute our test statistic as

$$Z_0 = \frac{A - E[A]}{\sqrt{Var(A)}}$$

and reject H_0 if $|Z_0| > Z_{\alpha/2}$.

The Runs Above and Below test is defined as follows. Let B = total number of runs "above and below the mean" out of n observations. Let n_1 = number of runs above the mean and n_2 = number of runs below the mean. Then

$$B \approx Nor\left(\frac{2n_1n_2}{n} + \frac{1}{2}, \frac{2n_1n_2\left(2n_1n_2 - n\right)}{n^2\left(n-1\right)}\right)$$

We will compute our test statistic as

$$Z_0 = \frac{B - E[B]}{\sqrt{Var(B)}}$$

and reject H_0 if $|Z_0| > Z_{\alpha/2}$.

The Correlation Test is defined as follows. Let ρ be the lag-1 correlation of some uniform R_i's.

$$\widehat{\rho} \equiv \left(\frac{12}{n-1}\sum_{k=1}^{n-1} R_k R_{1+k}\right) - 3$$

and

$$\widehat{\rho} \approx Nor\left(0, \frac{13n-19}{(n-1)^2}\right)(under H_0)$$

We will compute our test statistic as

$$Z_0 = \frac{\widehat{\rho}}{\sqrt{Var(\widehat{\rho})}}$$

and reject H_0 if $|Z_0| > Z_{\alpha/2}$.

We can easily perform these tests by calling the following functions: *runsUpDown()*, *runsAboveBelow()*, and *correlationTest()*. The results of these functions have been summarized in the table below.

Test	$\lvert Z_0\rvert$	$Z_{\alpha/2}$	p-value
Runs Up and Down	1.953	1.960	0.051
Runs Above and Below	0.425	1.960	0.671
Correlation Test	0.405	1.960	0.685

Figure 4.6: Aggregated results of statistical tests for Independence

Figure 4.6 shows that for all three tests we fail to reject the null hypothesis that the observations are independent. Therefore, we will keep our assumption that our Tausworthe pseudo-random number generator produces independent results.

4.4 Extensions

Now that we have successfully created Uniform(0,1) PRNs and tested their validity, we can use these PRNs to simulate other distributions. This allows us to extend the usefulness of our generator for a wide variety of purposes. In this case, we will focus on generating observations from a Nor(0,1)

distribution. The Box-Muller method states that if U_1 and U_2 are iid Uniform(0,1) then

$$Z_1 = \sqrt{2\ln(U_1)}\cos(2\pi U_2)$$
$$Z_2 = \sqrt{2\ln(U_1)}\sin(2\pi U_2)$$

where Z_1 and Z_2 are iid Nor(0,1). Using our Tausworthe generator, we can create two vectors of PRNs and plug them into Z_1 to get a normal distribution. To do this, we can use the function $boxMullerNorm(u1, u2)$. Doing so provides us with the following data.

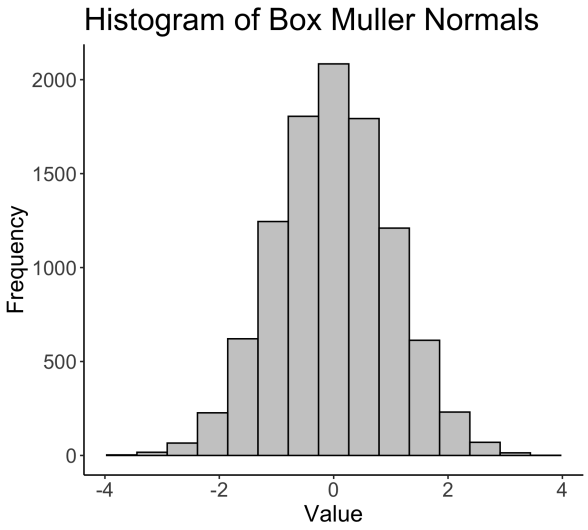

Figure 4.7: Plot of Box-Muller Normal Observations Generated from Tausworthe PRNs

The distribution of Figure 4.7 appears roughly normal with a mean centered around 0. To further investigate the underlying distribution we can use a Normal QQ plot.

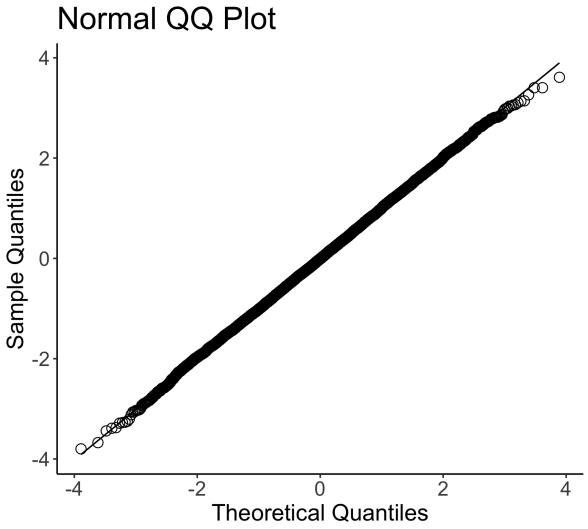

Figure 4.8: Normal QQ Plot of Box-Muller Normal Observations Generated from Tausworthe PRNs

Figure 4.8 shows that our simulated normal data very closely aligned with the theoretical quantiles of a Nor(0,1) distribution. Further evaluations such as the Kolmogorov-Smirnov test can be completed to formally test the quality of our simulated Nor(0,1) data. These additional tests are outside the scope of this paper but are left as an exercise to the curious reader.

5 Conclusion

Overall, we have been successful in implementing the Tausworthe random number generator in R to generate Uniform(0,1) psuedo-random numbers. We confirmed the quality of our generator using visual analysis, goodness of fit tests, and independence tests. Finally, we showed an example of possible extensions of our uniform PRNs to generate observations from a Nor(0,1) distribution. This is just a one example of how we can use PRNs to simulate more advanced distributions. Going further than just generation, these numbers can be used for a wide variety of Monte Carlo experiments, simulations, or analytical methods. The generator we have created in R will be particularly useful in understanding what happens when the parameters $q, r,$ and l are chosen poorly.

6 References

[1] P. L'Ecuyer. History of uniform random number generation. In *2017 Winter Simulation Conference (WSC)*, pages 202–230, 2017. doi: 10.1109/WSC.2017.8247790.

[2] J. Mellor-Crummey. Random number generation - rice university. URL `https://www.cs.rice.edu/ johnmc/comp528/lecture-notes/Lecture21.pdf`.

[3] R. C. Tausworthe. Random numbers generated by linear recurrence modulo two. *Mathematics of Computation*, 19(90):201–209, 1965. ISSN 00255718, 10886842. URL `http://www.jstor.org/stable/2003345`.

[4] G. van den Burg. *Package 'SyncRNG'*, 2023. R package version 1.3.2.

[5] N. Zierler and J. Brillhart. On primitive trinomials (mod 2). *Information and Control*, 13(6):541–554, 1968. ISSN 0019-9958. doi: https://doi.org/10.1016/S0019-9958(68)90973-X.

A Appendix A

A.1 R Function Documentation

adjacentPrns
Description: Returns a data frame that contains adjacent values in 2 or 3 dimensions.
Usage: adjacentPrns(x, dimensions=2)
Arguments:
x: vector of pseudo-random numbers
dimensions: integer value in [2,3] that corresponds to how many dimensions to compare adjacent numbers in.

bitToBase10
Description: Converts a series of bits into integer values in base 10.
Usage: bitToBase10(x)
Arguments:
x: string of bits to convert

boxMullerNorm
Description: Transforms two uniform psuedo random numbers into an observation from the normal(0,1) distribution.
Usage: boxMullerNorm(u1, u2)
Arguments:
u1: numeric representing the first uniform value to use in the normal calculation.
u2: numeric representing the second uniform value to use in the normal calculation.

correlationTest
Description: Performs the correlation test for independence.
Usage: correlationTest(prn, alpha=0.05)
Arguments:
prn: vector of pseudo-random numbers between 0 and 1.
alpha: numeric corresponding to the significance level to perform the test at.

runsUpDown
Description: Performs the runs "Up and Down" test for independence.
Usage: runsUpDown(prn, alpha=0.05)
Arguments:
prn: vector of pseudo-random numbers between 0 and 1.
alpha: numeric corresponding to the significance level to perform the test at.

runsAboveBelow
Description: Performs the runs "Above and Below the Mean" test for independence.
Usage: runsUpDown(prn, alpha=0.05)
Arguments:
prn: vector of pseudo-random numbers between 0 and 1.
alpha: numeric corresponding to the significance level to perform the test at.

tausworthe
Description: Performs psuedo random number generation using the Tausworthe method.
Usage: tausworthe(seed, n, r, q, l)
Arguments:
seed: a string of bits that represents the initialization of the first q binary digits in a Tausworthe generator.
n: an integer representing the number of random numbers for the generator to produce.
r: an integer value that corresponds to r in the Tausworthe algorithm
q: an integer value that corresponds to q in the Tausworthe algorithm
l: an integer value that corresponds to the length of bits used to calculate PRNs in the Tausworthe algorithm

uniformChiSq
Description: Performs chi square goodness of fit test for a uniform distribution with k=5.
Usage: uniformChiSq(x)
Arguments:
x: vector of numbers to test for fit against the uniform distribution.