

ISyE 6644 - Simulation

— Second mini-project: Problem 11 —

Pablo González
pvaldivia6@gatech.edu

Abstract—The following paper aims to better understand how a Tausworthe pseudo-random number generator works. For this purpose, we implemented the algorithm for such a generator and performed relevant statistical tests on it to demonstrate that the given PRN's that are approximately i.i.d. $\text{Uniform}(0,1)$. Finally, using the Box-Muller transformation method, some $\text{Norm}(0,1)$ deviates were generated and plots of both distributions were showed for illustrative purposes.

1 INTRODUCTION

In this paper we analyze the obtained results after implementing a Tausworthe pseudo-random number generator. For this purpose, as previously indicated, we will conduct certain statistical tests to prove that the generated pseudo random numbers (PRN's) are approximately i.i.d. $\text{Uniform}(0,1)$ and, later in the paper, some i.i.d. $\text{Normal}(0,1)$ deviates will be generated using the Box-Muller transformation method.

For illustration purposes, a Jupyter Notebook document is included along with this report. However, all generated figures and implementation code can be found on Appendix A and B respectively.

2 PROBLEM BACKGROUND AND DESCRIPTION

The problem we are studying consists of understanding how, using a deterministic algorithm, we can generate pseudo-random numbers that appear to be random to us and share the statistical properties of independent and identically distributed uniform random variables.

Our choice of pseudo-random number generator is the Tausworthe generator, which has a rather direct implementation described as follows:

$$B_i = (B_{i-r} + B_{i-q}) \bmod(2) = B_{i-r} \text{ XOR } B_{i-q}, \quad \text{where } r < q$$

The period of this class of generator is equal to 2^{q-1} and sequences are transformed into Uniform(0,1) random variables by taking a sequence of l bits, computing its decimal value, and dividing it by 2^l . For our implementation, the selected values for the r , q and l parameters were 18, 31 and 35 respectively.

Additionally, the initialization of our Tausworthe generator was based on time (time since the epoch) in case that no seeds were given, thus generating different sequences every time it executed and, for replicaability, a capability to allow for the use of random seeds was also implemented by converting the given number to binary, adjusting its length and using to initialize our generator.

As per the problem description we are asked to perform statistical tests on the generator to see that it gives PRN's that are approximately i.i.d. Uniform(0,1) and then plot adjacent PRN's (U_i, U_{i+1}) on the unit square to observe whether there are any patterns. Thus, we will first apply a Chi-squared goodness of fit test, then a Runs test and, finally, we will check for serial correlation.

Regarding the generation of our i.i.d. Normal(0,1) PRN's, we first generated uniform PRN's, and used pairs of uniform numbers (U_1 and U_2) to employ the Box-Muller transformation described as follows:

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

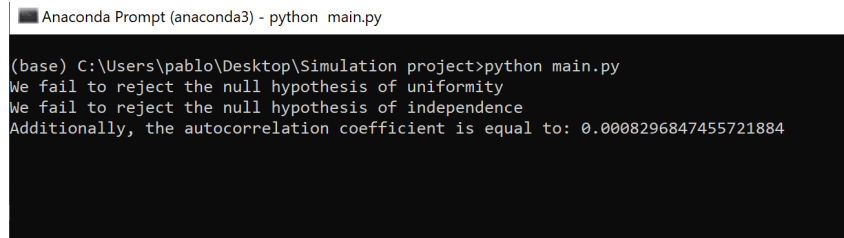
This method, although computer intensive, can generate us the same number of PRN's as the employed generator.

3 MAIN FINDINGS

Our implementation uses the Python programming language to generate the requested pseudo-random numbers and perform the required statistical analysis. A Jupyter Notebook document is submitted along with this paper to briefly show and explain the conducted procedure. However, the implemented program, which can be found on Appendix B, can be easily run from the console, to generate the figures used in this report, as follows:

```
PYTHONPATH=../:. python main.py
```

Running this command results in the following output and generates the same figures as the ones presented on Appendix A:



```
Anaconda Prompt (anaconda3) - python main.py
(base) C:\Users\pablo\Desktop\Simulation project>python main.py
We fail to reject the null hypothesis of uniformity
We fail to reject the null hypothesis of independence
Additionally, the autocorrelation coefficient is equal to: 0.0008296847455721884
```

The first step performed after implementing the algorithms consisted in generating a sequence of 1,000,000 Uniform(0,1) PRN's with the seed "6644" by using the previously generated "unif(n, a, b, seed)" function, which can generate n uniformly distributed PRN's between the numbers a and b and an initial random state (seed) of choice.

Once this random numbers were generated, we set a significance level, α , of 0.05 and run a goodness of fit test using a k, number of bins, parameter of 10, obtaining a χ^2_0 statistic of approximately 5.225. This result was then compared with the theoretical value from a Chi-square distribution with 9 degrees of freedom ($\chi^2_{0.05,9}$) of 16.919, thus failing to reject the null hypothesis of uniformity.

To check for serial correlation, the auto-correlation coefficient was estimated, obtaining a value of approximately $-5e-4$, thus also being able to conclude that no significant serial correlation was to be found between adjacent PRN's.

Furthermore, using the generated results, we were also able to perform a "Runs" test, obtaining an observed Z score of 0.827, which, using a significance level of 0.05, was then compared to the expected value of 1.96, thus also failing to reject the null hypothesis of independence.

Finally, these results can also be visually analyzed on Appendix A, when it is also direct to conclude that the observed distribution approximates to an i.i.d Uniform(0,1) and no correlation pattern is to be appreciated.

4 CONCLUSION

In this paper we implemented a Tausworthe pseudo-random number generator capable of generating different PRN's at different times and using seeds for reproducibility and then analyzed the obtained results to prove that the generated PRN's were approximately i.i.d. Uniform(0,1). Additionally, to examine our results, histograms and kernel density plots were generated and adjacent PRN's were plotted on the unit square respectively to visually analyze for any patterns.

When conducting the former statistical analysis, using a significance level of 0.05, we failed to reject both the null hypotheses of independence and uniformity, thus concluding that our PRN's did share the properties of a uniform distribution. In other words, when analyzing these results, we can notice that the generated PRN's follow an i.i.d Uniform(0,1) distribution.

Finally, on an additional note, using the Box Muller transformation method, a function capable of generating Normal(μ, σ^2) PRN's was also implemented and, using a different seed, random standard normal numbers were generated.

5 APPENDIX A

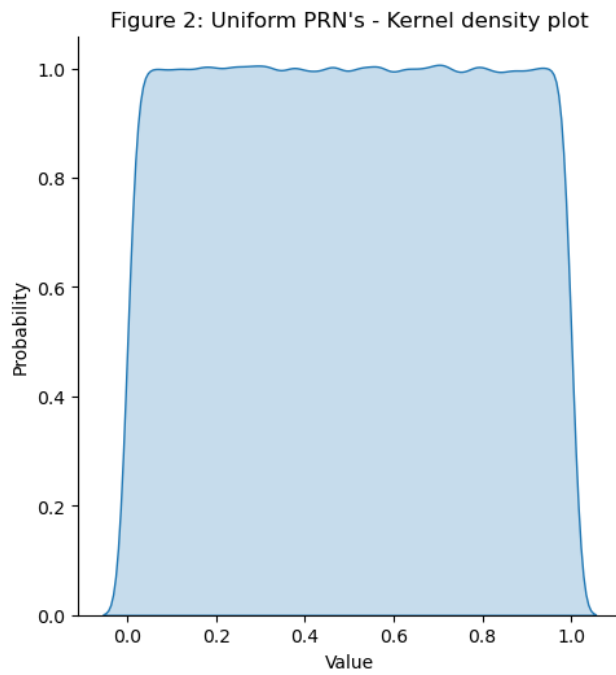
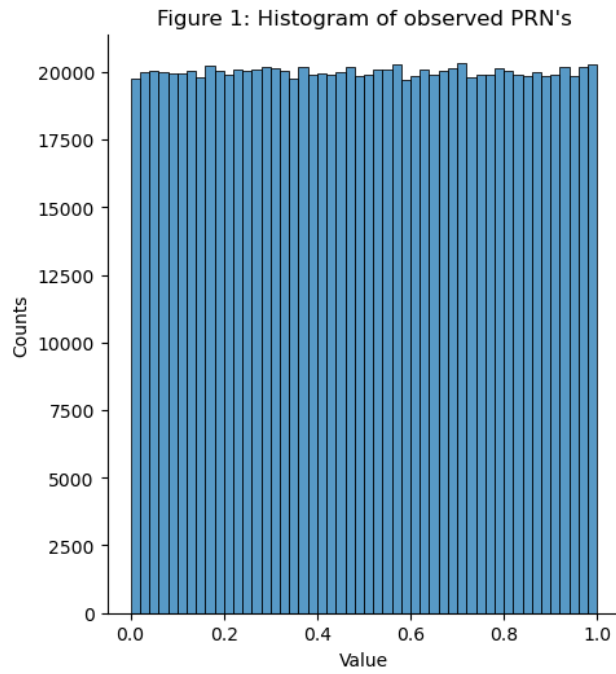


Figure 3: Scatter plot of adjacent PRN's

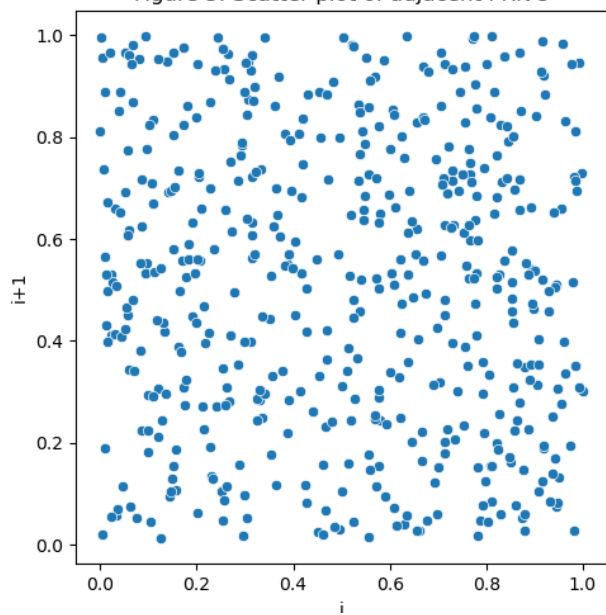
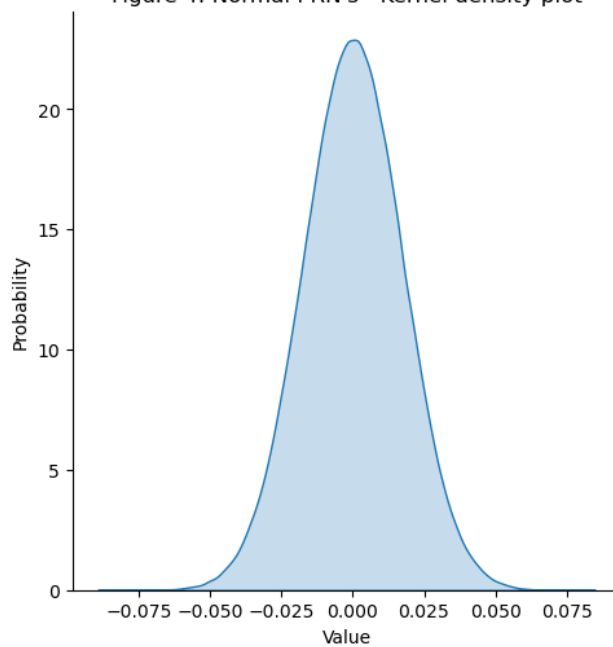


Figure 4: Normal PRN's - Kernel density plot



6 APPENDIX B

Main.py

```
import seaborn as sns
import numpy as np
import time

def tausworthe(n = 1, r = 3, q = 5, l = 4, seed = None):

    # Step 1: Initialize binary sequence
    binary = bin(seed)[2:] if seed else bin(clock())[2:]
    storage = ( binary*( 1 + q//len(binary) ) )[:q]
    position = q

    # Step 2: "The period of 0-1 bits is always 2**q - 1"
    period = 2**q - 1
    if n > period:
        n = period
        print(f"You have exceeded the period, \
              {period} numbers will be returned")

    # Step 3:
    for i in range(l*n):
        # B_i = (B_{i-r} + B_{i-q} mod 2)
        B_i = (int(storage[position-r]) +
               int(storage[position-q]))%2
        storage += str(B_i)
        position += 1
    storage = storage[:-q]
    storage = [ int(storage[i:i+l],2)/2**l \
                for i in range(0, len(storage), l)]
    return storage
```

```

def clock():
    temp = round(time.time()*100000)
    inverse = int(str(temp)[::-1])
    time.sleep(0.00001)
    return inverse

def unif(n = 1000000, a = 0, b = 1, seed = None):
    # This function returns n uniform random variables
    # between a and b.
    temp = [(b-a)*i + a for i in tausworthe(n,18,31,35, seed)]
    return temp

def norm(prns, mu = 0, var = 1, seed = None):
    # Using the Box-Muller transformation, this function
    # returns the inputted uniform random variables
    # as normal rvs with mean "mu" and variance "var"
    x,z = unif(n, seed = seed), []
    for i in range(0, len(x),2):
        # To reduce computation time (i.e. given the logs),
        # the calculations are first separated to then
        # only change whether we apply a sin/cos function
        p1, p2 = np.sqrt(-2*np.log(x[i])), 2*np.pi*x[i+1]
        z1 = p1*np.radians( np.cos(p2) )
        z2 = p1*np.radians( np.sin(p2) )
        z1, z2 = var*z1+mu, var*z2+mu
        z = np.append(z, [z1,z2])
    return z

def gof(prns, nbins = 10):
    # Goodness of fit test: H0 of uniformity.
    bins = np.array(list(range(0,nbins,1)))/nbins
    expected = [len(prns)/nbins]*nbins
    observed = np.bincount(np.digitize(prns,bins))[1:]
    chi0 = sum( np.divide( (observed-expected)**2, expected) )
    return chi0 > 16.919 # Do we reject?

```



```

def correlation(prns):
    return np.corrcoef( prns[:-1], prns[1:] )[1,0]

def runs(prns):
    # Runs test: H0 of independence.
    temp = np.sign( np.diff( np.sign( np.append([0], \
        np.diff(prns,1)) ) ) )
    runs, n = np.count_nonzero(temp), len(prns)
    A_mean, A_var = (2*n-1)/3, (16*n-29)/90
    z0 = (runs-A_mean)/np.sqrt(A_var)
    return abs(z0) > 1.96 # Do we reject?

def tests(prns):
    if gof(prns) == False:
        print("We fail to reject the null hypothesis " +
            "of uniformity")
    else:
        print("We reject the null hypothesis of uniformity")
    if runs(prns) == False:
        print("We fail to reject the null hypothesis " +
            "of independence")
    else:
        print("We reject the null hypothesis of independence")
    cor = correlation(prns)
    print(f"Additionally, the autocorrelation coefficient" +
        "is equal to: {cor}")
    return None

```

```

if __name__ == '__main__':

    seq1 = unif(1000000, seed = 6644)
    tests(seq1)
    hist = sns.displot(seq1, bins = 50)
    hist.set(xlabel = "Value", ylabel = "Counts",
             title = "Histogram of observed PRN's")
    hist.savefig("uniform_hist.png")
    hist.fig.clf()

    displot = sns.displot(seq1, kind = "kde", fill = True)
    displot.set(xlabel = "Value", ylabel = "Probability",
                title = "Uniform PRN's - Kernel density plot")
    displot.savefig("uniform_kde.png")
    displot.fig.clf()

    seq2 = seq1[:1000]
    a = [seq2[i] for i in range(len(seq2)) if i%2==1]
    b = [seq2[i] for i in range(len(seq2)) if i%2==0]
    dots = sns.scatterplot( x = a, y = b , legend = None)
    dots.set(xlabel = "i", ylabel = "i+1",
             title = "Scatter plot of adjacent PRN's")
    dots.figure.subplots_adjust(top=.95)
    dots.figure.savefig("uniform_sct.png")
    dots.figure.clf()

    dev = norm(500000,0,1, seed = 12345)
    displot2 = sns.displot(dev, kind = "kde", fill = True)
    displot2.set(xlabel = "Value", ylabel = "Probability",
                 title = "Normal PRN's - Kernel density plot")
    displot2.savefig("normal_kde.png")
    displot2.fig.clf()

```