Statistical Properties of (Quasi) Monte-Carlo Methods

Ahmer Nadeem Khan

28 February 2025

1 Abstract

In this project, we will probe the theoretical reliability of pseudo-random and quasi-monte Carlo methods by focusing on the canonical problem: how well these numbers simulate the Uniform Distribution. We will describe first the statistical methods and tests at our disposal, and then perform experiments on various random number generators using these tests. We then present and compare our results. (Homework problems are marked in red).

2 Objectives

Our objective is to numerically test pseudo-random and low-discrepancy number generators described previously. We first give a description of the necessary mathematical notions and then we use the following statistics:

- Kolmogorov-Smirnov (KS) statistic
- χ^2 statistic
- Anderson-Darling statistic

We will focus on the following tests:

- Directly applying the KS statistic to the basic Fibonacci Generator.
- Maximum Test using the Anderson-Darling statistic (and the KS statistic) to the 1 dimensional Halton sequence (Van der Corput sequence) and a higher dimensional Halton sequence.
- Gap test on the Lagged Fibonacci Generator using the χ^2 statistic.

We focus on the basic problem of simulating the Uniform Distribution $Unif(0, 1)$. Each of these generators have either been described in detail before or will be described here, and will be used for the goal above. We also describe the methodology of the tests and the theory behind the relevant statistics. Finally, we perform specific experiments, and present numerical and visual results.

3 Random Number Generators

Linear Congruential Generators (LCG's), the Halton Sequence, the Mersenne Twister and a few other generators were described previously. We now describe in addition, the Fibonacci Generator and the Lagged Fibonacci Generator.

3.1 Fibonacci and the Lagged Fibonacci Generator

We can generalize the LCG in the following way to get a larger class of generators.

Definition 1. *A **Multiple Recursive Generator** has the form*

$$x_n \equiv a_1 x_{n-1} + \ldots + a_k x_{n-k} \pmod{p}$$

where p is a prime.

It is possible to find $a_1, \ldots, a_k$ such that the sequence has period length $p^k - 1$. Initial values $x_0, \ldots, x_{k-1}$ can be chosen arbitrarily, but not all zero. The **Fibonacci Generator** has the form

$$x_n = x_{n-1} + x_{n-2} \pmod{M}$$

This is known to fail several statistical tests, an example of which we will see. This generator has a problem where small numbers frequently follow small numbers. In particular, it is impossible to compute x_{i+1} such that $x_{i-1} < x_{i+1} < x_i$ or $x_i < x_{i+1} < x_{i-1}$, which is property that should occur with probability $\frac{1}{3}$. An improvement is the **Lagged Fibonacci Generator**

$$x_n = x_{n-r} + x_{n-s} \pmod{M}$$

where $n \geq r$, $0 < s < r$, and r and s are the lags. There are further modifications but will focus on this particular form. The maximum possible period of the Lagged Fibonacci Generator is $2^{k-1} \times 2^{M-1}$. Note that we need at least r seed values to start the lagged generator. We usually require at least one of these initial values to be odd, and there is also some theory about how to choose these values, which is beyond the scope of this project.

4 Statistics and Tests

A Martin-Lof sequence is a sequence that satisfies all Turing complete almost sure properties. In particular, these are infinite. A 'pseudo-random' sequence is such that it satisfies a finite number of properties (making it computable). These statistical tests form a criteria for these number sequences, i.e. their acceptability (which can be for various purposes). Knuth proposes that a sequence is "innocent until proven guilty" - we reject the sequence in the event that it fails our tests. We describe useful statistics first that are integral to the tests we want to perform.

4.1 χ^2 Goodness of Fit

Let a random experiment have k mutually exclusive and exhaustive outcomes, say $A_1, \ldots, A_k$. Put $p_i = P(A_i)$ and

$$p_1 + \cdots + p_k = 1.$$

The experiment is repeated n times, independently, and we let Y_i be the number of times the experiment results in A_i. In 1900, Pearson was able to show that the expression

$$Q_{k-1} = \sum_{i=1}^{k} \frac{(Y_i - np_i)^2}{np_i}$$

has an approximate χ^2 distribution with $k-1$ degrees of freedom, i.e $Q_{k-1} \sim \chi^2(k-1)$).

Definition 2. *X has a **chi-square distribution** with r degrees of freedom if*

$$f(x) = \frac{1}{\Gamma(r/2)2^{r/2}} x^{r/2-1} e^{-x/2}, \quad 0 \leq x < \infty$$

where $\Gamma(t) = \int_0^\infty y^{t-1} e^{-y}\, dy$, $t > 0$.

The mean, variance, and moment generating function of $\chi^2(r)$ are:

$$E[X] = r, \quad Var(X) = 2r, \quad M(t) = (1-2t)^{-r/2}, \quad t < 1/2.$$

Since the χ^2 statistic approaches the aforementioned distribution in the limit, we generally want to choose a large n. A good rule of thumb is to choose n such $np_i \geq 5$ for the smallest p_i. Note that Y_i is the sample frequency of event A_i, while np_i is the expected frequency. The statistic intuitively measures the relative L_2 distance between these values. This discussion leads to a better approach to implementing tests based on the χ^2 statistic.

4.2 Kolmogorov-Smirnov and Anderson-Darling Statistics

We begin by defining the true and sample (empirical) CDF.

Definition 3. *The **distribution function** induced by a probability measure $\mathbb{P}$ on $(\mathbb{R}, \mathcal{B})$ is the function*

$$F(x) := \mathbb{P}((-\infty, x]), \quad \textit{for all } x \in \mathbb{R}.$$

The sets of these kind generate the Borel σ-algebra. There is considerable probability theory concerning the distribution function (also known as the cumulative distribution), especially in regards to convergence. These are results underlying sampling techniques and methods, as well as inference. We state a few theorem, and also prove certain results we will use.

Theorem 4.1. *The distribution function F characterizes the Probability measure.*

This is not hard to show from first principles of probability theory. The main result of use to us is:

Theorem 4.2. *A function F is a distribution function of a unique probability measure $\mathbb{P}$ on $(\mathbb{R}, \mathcal{B})$ if and only if:*

1. *F is non-decreasing.*
2. *F is right-continuous.*
3. $\lim_{x \to -\infty} F(x) = 0$ *and* $\lim_{x \to +\infty} F(x) = 1$.

Proof. Proof. ($\to$) First, assume F is a distribution function.

1. If $y > x$, then $(-\infty, x] \subset (-\infty, y]$, implying

$$F(x) = \mathbb{P}((-\infty, x]) \leq \mathbb{P}((-\infty, y]) = F(y).$$

   This shows that F is non-decreasing. We have used here the monotonicity of the measure.

2. Let $(x_n)_{n \geq 1}$ be a sequence in $\mathbb{R}$ such that $x_n \downarrow x$ (i.e. $x_{n+1} \leq x_n$, and the limit is x). Then,

$$(-\infty, x] = \bigcap_{n=1}^{\infty} (-\infty, x_n].$$

   Note that $(-\infty, x_n]$ (for $n \in \mathbb{Z}^+$) is a decreasing sequence of events whose limit is $(-\infty, x]$.

   By a basic result (that we use without proof), we get

$$\mathbb{P}\left(\bigcap_{n=1}^{\infty} (-\infty, x_n]\right) = \lim_{n \to \infty} \mathbb{P}((-\infty, x_n]) = \mathbb{P}((-\infty, x]).$$

   This shows that

$$\lim_{x \to x_0^+} F(x) = F(x_0), \quad \forall x_0 \in \mathbb{R}.$$

   Hence, F is right-continuous.

3. Let $(x_n)_{n \geq 1}$ be such that $x_n \to +\infty$. Then

$$F(x_n) = \mathbb{P}((-\infty, x_n]) \uparrow \mathbb{P}(\mathbb{R}) = 1.$$

   Where the "up-limit" has the obvious meaning. Similarly, as $x_n \to -\infty$, we obtain $F(x) \to 0$.

The backward direction is slightly more involved, and is not useful to us. This, it is omitted. □

We now also define the sample CDF:

Definition 4. *Let $(X_1, \ldots, X_N)$ be independent, identically distributed real random variables with the common cumulative distribution function $F(x)$. Then the* ***empirical distribution function*** *is defined as*

$$F_N(x) = \frac{\textit{number of elements in the sample } \leq x}{N} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}_{\{X_i \leq x\}},$$

where $\mathbf{1}_A$ is the indicator of event A.

Theorem 4.3. *The expectation and variance of $F_N(x)$ satisfy:*

$$E[F_N(x)] = F(x), \quad \textit{and} \quad \mathrm{Var}(F_N(x)) = \frac{F(x)(1-F(x))}{N}.$$

Proof. We first note that the indicator function $\mathbf{1}_{(-\infty,x]}(X_i)$ follows a Bernoulli distribution:

$$\mathbf{1}_{(-\infty,x]}(X_i) \sim \mathrm{Ber}(F(x)),$$

i.e., it is a Bernoulli random variable with success probability $p = F(x)$.

Since $F_N(x)$ is the sample mean of N such Bernoulli trials, the sum

$$\sum_{i=1}^{N} \mathbf{1}_{(-\infty,x]}(X_i) \sim \mathrm{Bin}(N, F(x)),$$

follows a binomial distribution with parameters N (number of trials) and $F(x)$.

From properties of the binomial distribution, we compute the expectation:

$$E\left[NF_N(x)\right] = NF(x) \quad \Rightarrow \quad E[F_N(x)] = F(x).$$

Similarly, using the variance formula for a binomial variable,

$$\mathrm{Var}\left(NF_N(x)\right) = NF(x)(1-F(x)),$$

we obtain:

$$\mathrm{Var}(F_N(x)) = \frac{F(x)(1-F(x))}{N}.$$

□

This shows that the sample CDF is an unbiased estimator for the true distribution, and $\mathrm{Var}(F_N(x)) \to 0$ as $N \to \infty$. We also have convergence results. By the strong law of large numbers (SLLN), the estimator $F_N(x)$ converges to $F(x)$ as $N \to \infty$ almost surely, for every value of x:

$$F_N(x) \xrightarrow{\text{a.s.}} F(x).$$

which is equivalent to (stated without proof):
For any $\epsilon > 0$

$$\lim_{N\to\infty} \mathbb{P}\{ \sup_{-\infty<x<\infty} |F_N(x) - F(x)| > \epsilon\} = 0.$$

The asymptotic distribution can be further characterized in several different ways. First, the central limit theorem states that *pointwise*, $F_N(x)$ has an asymptotically normal distribution with the standard $\sqrt{N}$ rate of convergence:

$$\sqrt{N}\left(F_N(x) - F(x)\right) \xrightarrow{d} \mathcal{N}\left(0, F(x)(1-F(x))\right).$$

The first result can in fact be made stronger, in what is colloquially called the Fundamental Theorem of Statistics:

Theorem 4.4 (Glivenko-Cantelli Theorem, 1933)**.**

$$\|F_N - F\|_\infty = \sup_{x\in\mathbb{R}} |F_N(x) - F(x)| \xrightarrow{a.s.} 0.$$

This result gives us uniform convergence (not just almost sure convergence). It also defines the infinity norm or L_∞ norm of $F_N - F$, which is called the Kolmogorov Smirnov Statistic.

Definition 5. *The **Kolmogorov-Smirnov (KS) statistic** is defined as:*

$$D_N = \sup_{-\infty<x<\infty} |F_N(x) - F(x)|.$$

An asymptotic result about the distribution of D_N is:

$$\lim_{N\to\infty} P\left(\sqrt{N}D_N \le x\right) = 1 - 2\sum_{j=1}^{\infty}(-1)^{j-1}e^{-2j^2x^2} = H(x).$$

Note that numerical values for $H(x)$ are tabulated and the approximation is usually sufficient when $N > 35$. Also note that $H(x)$ is independent of the distribution $F(x)$.

We can also define more general statistics, in particular the **Cramer-von Mises Family** of statistics, defined as

$$Q = N\int_{-\infty}^{\infty} \left(F_N(x) - F(x)\right)^2 \psi(x)\, dF(x)$$

where $\psi(x)$ is a suitable function which gives "weights" to the squared differences. If

$$\psi(x) = \frac{1}{F(x)(1-F(x))}$$

then we get the **Anderson-Darling statistic** A^2, which can be computed as

$$A^2 = -N - \frac{1}{N}\sum_{i=1}^{N}(2i-1)\left(\log Z_{(i)} + \log(1 - Z_{(N+1-i)})\right)$$

where $Z_{(i)}$ are the ordered values $\left(Z_{(1)} \leq Z_{(2)} \leq \cdots \leq Z_{(N)}\right)$ of the sequence $Z_i = F(X_i)$, and $X_1, \ldots, X_N$ is the sample.

4.3 Design of Statistics

The test designs are methods of inference at a particular confidence level i.e. Hypothesis testing. For the χ^2 test, we get the following steps. We want to test a model that assigns the following probabilities to the outcomes of a random experiment:

	A_1	A_2	$\ldots$	A_k
Probabilities	p_1	p_2	$\ldots$	p_k

We proceed as follows:

- Simulate the random experiment n times and record Y_i, the number of times outcome A_i is observed.
- Compute $Q_{k-1} = \sum_{i=1}^{k} \frac{(Y_i - np_i)^2}{np_i}$.
- If Q_{k-1} is too small or too large, the model will be suspect.

The KS test is also generally designed as follows:

1. Obtain the sample: $X_1, \ldots, X_N$.
2. State the null hypothesis, H_0: *"The sample is from $F(x)$"*.
3. Test the null hypothesis: Compute D_N. Reject the hypothesis if D_N is too large.

We can also combine the two tests using blocks of the full sequence. The KS test is used when F is continuous. Just as for the Ks statistic, we generally find the 1%, 5%, and 10% significance levels, and if the computed value is bigger than the respective α, we reject the Null hypothesis at that confidence level. There is also analytic formula for computing the KS statistic that leads to a natural, fairly straightforward algorithm. Note that for this result, we require continuity of F, i.e. in addition to being right-continuous, F is also left-continuous. We derive that result as follows.

Theorem 4.5. *The Kolmogorov=Smirnov statistic is computed as:*

$$D_N = \sup_{-\infty<x<\infty} |F_N(x) - F(x)| = \max(D^+, D^-)$$

where D^+, D^- are defined below.

Proof. We start with a sample $X_1, \ldots, X_N$. Next, sort the sample so that $X_1 \leq X_2 \leq \cdots \leq X_N$ (here there is an abuse of notation since we use the same variables for both the original sample and the sorted sample.) We have

$$D_N = \sup_{-\infty<x<\infty} |F_N(x) - F(x)|$$

where the sample cdf is the following step-function (easy to check)

$$F_N(x) = \begin{cases} 0, & \text{if } x < X_1, \\ k/N, & \text{if } X_k \leq x < X_{k+1}, \\ 1, & \text{if } x > X_N. \end{cases}$$

Therefore,

$$D_N = \max\left[\sup_{x<X_1} |F(x)|, \sup_{X_1\leq x<X_2} \left|\frac{1}{N} - F(x)\right|, \ldots, \sup_{x>X_N} |1 - F(x)|\right].$$

The first and last terms are:

$$\sup_{x<X_1} |1 - F(x)| = F(X_1), \quad \text{and} \quad \sup_{x>X_N} |1 - F(x)| = 1 - F(X_N).$$

since F is non-decreasing by Theorem 4.2. For the other terms, by Lemma 4.6, we get that as $k = 1, \ldots, N-1$:

$$\sup_{X_k\leq x<X_{k+1}} \left|\frac{k}{N} - F(x)\right| = \max\left(F(X_{k+1}) - \frac{k}{N}, \frac{k}{N} - F(X_k)\right).$$

Thus,

$$D_N = \max\left[F(X_1), \max_{k=1,\ldots,N-1}\left(F(X_{k+1}) - \frac{k}{N}, \frac{k}{N} - F(X_k)\right), 1 - F(X_N)\right]$$

$$= \max\left[\underbrace{\max_{k=1,\ldots,N}\left(\frac{k}{N} - F(X_k)\right)}_{D^+}, \underbrace{\max_{k=1,\ldots,N}\left(F(X_k) - \frac{k-1}{N}\right)}_{D^-}\right].$$

where the last term was absorbed in the first expression and vice-versa. By definition we get that

$$D_N = \max(D^+, D^-).$$

□

We need the following result in the proof above. (Homework 2.1)

Lemma 4.6.

$$\sup_{X_k \le x < X_{k+1}} \left| \frac{k}{N} - F(x) \right| = \max\left(F(X_{k+1}) - \frac{k}{N}, \frac{k}{N} - F(X_k) \right).$$

for $k = 1, \ldots, N-1$.

Proof. We first have to show that the RHS is an upper-bound to the LHS, and then we show that it is the least upper bound. We can do this in cases:

1. If $\frac{k}{N} \le F(x_k)$, then since F is non-decreasing we get that $F(x_{k+1}) - \frac{k}{n}$ is the largest value of the absolute difference on this interval. In this case all values of $F(x) - \frac{k}{N}$ are positive, so we can multiply by -1 and remove the absolute values. Note then that the max of the LHS is exactly this number, i.e. $\max(D^+, D^-) = D+$ so this is the required upper bound. Now assume, on the contrary that $\exists\, l \in \mathbb{R}$ s.t.

$$\begin{aligned} &\left| \frac{k}{N} - F(x) \right| \le l < max(D^+, D^-) = D^+ \\ &\implies F(x) - \frac{k}{N} \le l < F(x_{k+1}) - \frac{k}{N} \\ &\implies F(x) \le l + \frac{k}{N} < F(x_{k+1}) \end{aligned}$$

   We then take the limit on both sides as $x \to x_{k+1}^-$, which gives us

$$F(x_{k+1}) \le l + \frac{k}{N} < F(x_{k+1})$$

   a contradiction, by left-continuity of F.

2. The case $F(x_{k+1}) \le \frac{k}{N}$ has an identical proof as Case 1 with D^- replacing D^+, and we remove the modulus directly since the values of the LHS are all positive. The contradiction is produced by right-continuity of F at x_k instead (note that this case does not use the continuity assumption on F).

3. If $F(x_k) \le \frac{k}{N} \le F(x_{k+1})$ then on the right of $\frac{k}{N}$ we can multiply by -1 and on the left the values of the LHS will be positive (I mean without the absolute values). Then we are free to remove the absolute values (this amounts to the required inequalities). Then to find the maximum magnitude, we again use the property that F is non decreasing, so on the left the maximum magnitude is

$$\frac{k}{N} - F(x_k)$$

   and on the right

$$F(x_{k+1}) - \frac{k}{N}$$

since we subtract the smallest possible number, or add the largest possible number. Then their maximum will be an upper bound (note these are both positive values, and they may be equal). In either case, we must have D^+ or D^- as the least upper bound, otherwise we get a contradiction by the same argument as in Case 1 and 2; we choose the largest subinterval and restrict to it to get the contradiction. In particular, if $max(D^+, D^-) = D^+$, then we restrict to the subinterval $[\frac{k}{N}, F(x_{k+1}))$ to get a contradiction by left continuity, and vice versa. If they are equal, we are free to choose either side/case.

□

We can also see a numerical example of computing the KS statistic with a very simple example. (Homework 2.2) Consider F to be the distribution function of the Uniform Random Variable i.e. $X \sim Unif(0, 1)$ (this would be our null hypothesis). Then consider the following sample (observations), with $N = 3$:

$$X = \{x_1 = 0.2, x_2 = 0.6, x_3 = 0.7\}$$

This gives us the following CDF:

$$F_3(x) = \begin{cases} 0, & \text{if } x < 0.2, \\ 1/3, & \text{if } 0.2 \leq x < 0.6, \\ 2/3, & \text{if } 0.6 \leq x < 0.7, \\ 1, & \text{if } x > 0.7. \end{cases}$$

while $F(x) = x$ for the Uniform distribution. We can then compute D^+ as

$$D^+ = \max_{k=1,2,3} \left(\frac{k}{3} - x_k \right) = \max_{k=1,2,3} \left(\frac{1}{3} - 0.2, \frac{2}{3} - 0.6, \frac{3}{3} - 0.7 \right) = \max_{k=1,2,3} \left(\frac{2}{15}, \frac{1}{15}, \frac{3}{10} \right) = \frac{3}{10}$$

and D^- as

$$D^- = \max_{k=1,2,3} \left(x_k - \frac{k-1}{3} \right) = \max_{k=1,2,3} \left(0.2 - 0, 0.6 - \frac{1}{3}, 0.7 - \frac{2}{3} \right) = \max_{k=1,2,3} \left(\frac{1}{5}, \frac{4}{15}, \frac{1}{30} \right) = \frac{4}{15}$$

Then

$$D_N = \max(D^+, D^-) = \max(0.3, \approx 0.2666) = 0.3$$

Intuitively, the numbers D^+ and D^- measure the maximum positive and negative difference at the values $x_1, \ldots, x_k$ of the empirical CDF from the true CDF. The positive difference is the how much bigger in magnitude the empirical CDF is, while the negative difference is how much smaller in magnitude the empirical CDF is. We then find the maximum possible error for each respective value, giving us the largest possible positive (sample CDF > F) and largest possible negative (sample CDF < F) difference/discrepancy. The maximum of these two values then measures how much our sample CDF is 'away' from the true values, in the infinite norm sense (largest possible error) i.e in magnitude. This also

becomes clear from the plot of both functions, where the largest magnitude error is at the positive side at $x_3 = 0.7$. Note that it is the discrete, step nature of the sample CDF, and the fact that both functions are non-decreasing that allows this decomposition.

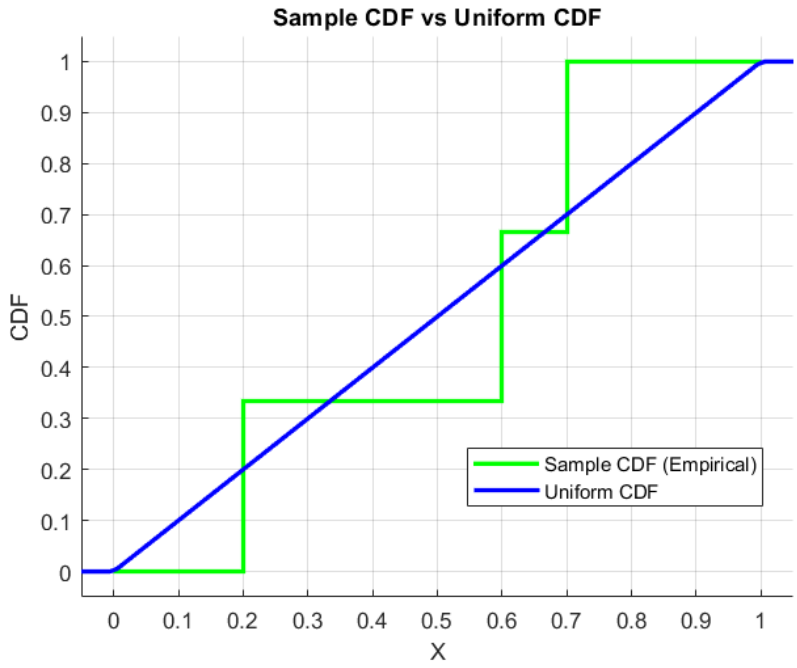


Figure 1: Sample CDF vs F

4.4 Statistical Tests

We can now begin to describe four statistical tests of interest here. Each test applies the goodness of fit in some way.

1. KS Test
2. The Correlation Test
3. The Maximum Test
4. The Gap Test

The KS test was described above. We only describe the basic definitions of the **Correlation test**, since we will not be using it for our experiments. We want to measure the correlation between U_i and U_{i+1} in a sequence of random numbers $U_1, U_2, \ldots$. The correlation coefficient of U and V, $\mathrm{Cov}(U,V)/\sqrt{\mathrm{Var}(U)\,\mathrm{Var}(V)}$, can be estimated from samples $U : u_1, \ldots, u_n$ and $V : v_1, \ldots, v_n$ by

$$\frac{\sum_{i=1}^{n}(u_i-\bar{u})(v_i-\bar{v})}{\sqrt{\sum_{i=1}^{n}(u_i-\bar{u})^2\sum_{i=1}^{n}(v_i-\bar{v})^2}}=\frac{n\sum u_iv_i-\sum u_i\sum v_i}{\sqrt{(n\sum u_i^2-(\sum u_i)^2)(n\sum v_i^2-(\sum v_i)^2)}}.$$

We will compute the correlation coefficient between the observations and their immediate successors, by letting

$$U: u_1, u_2, \ldots, u_{n-1}, u_n$$

$$V: u_2, u_3, \ldots, u_n, u_1$$

and compute

$$\rho=\frac{n\sum_{i=1}^{n}u_iu_{i+1}-(\sum_{i=1}^{n}u_i)^2}{n\sum_{i=1}^{n}u_i^2-(\sum_{i=1}^{n}u_i)^2}$$

where $u_{n+1}=u_1$.

(Homework 2.3) For $n=2$, we can show that the correlation coefficient ρ is always -1, provided the denominator in the expression is non-zero. Let $n=2$, which gives us

$$U: u_1, u_2$$

$$V: u_2, u_1$$

and then from the formula for ρ, we get

$$\begin{aligned}\rho&=\frac{2(u_1u_2+u_2u_1)-(u_1+u_2)^2}{2(u_1^2+u_2^2)-(u_1+u_2)^2}\\ \rho&=\frac{4u_1u_2-u_1^2-u_2^2-2u_1u_2}{2u_1^2+2u_2^2-u_1^2-u_2^2-2u_1u_2}\\ \rho&=\frac{-1(u_1^2+u_2^2-2u_1u_2)}{u_1^2+u_2^2-2u_1u_2}\\ \rho&=-1\end{aligned}$$

since the denominator is assumed to be non-zero and the factors cancel out.

The **Maximum test** is conducted as follows. Let $U_1, U_2, \ldots$ be i.i.d. random variables from the $U(0,1)$ distribution, and let $V_1=\max(U_1, U_2, \ldots, U_t)$, $V_2=\max(U_{t+1}, U_{t+2}, \ldots, U_{2t})$, etc. In other words, V_i is the maximum of a block of t uniform random variables, for some fixed t.

Observe that

$$P(V\leq x)=P(\max(U_1, U_2, \ldots, U_t)\leq x)$$

$$= P(U_1 \leq x, \ldots, U_t \leq x)$$

$$= P(U \leq x)^t$$

Since $P(U \leq x) = x$, the above expression simplifies to

$$P(V \leq x) = x^t.$$

Maximum test:

1. Given real numbers between 0 and 1: $u_1, u_2, \ldots$
2. Compute $v_1, \ldots, v_N$
3. Apply the KS-test to $v_1, \ldots, v_N$ with $F(x) = x^t$ for $0 < x < 1$

For a t-dimensional low-discrepancy sequence, we can take the blocks to be the entire vector.

For the **Gap test**, we consider a sub-interval $J = (\alpha, \beta)$. We say a gap of length r occurs whenever a subsequence

$$u_j, u_{j+1}, \ldots, u_{j+r-1}, u_{j+r}$$

such that $u_j, u_{j+1}, \ldots, u_{j+r-1} \notin J$ and $u_{j+r} \in J$ is observed.

- The probability P_i that a gap of length i occurs is

$$P_i = (1-p)^i p, \quad i = 0, 1, \ldots, t-1,$$

  where $p = \beta - \alpha$.

- The probability P_t that a gap of length t or more occurs is

$$P_t = (1-p)^t.$$

Gap test:

1. Pick a t (truncation) value. Then, pick a value for n such that $nP_i \geq 5$, verifying the rule of thumb for the χ^2-test.
2. The outcomes of the test are: "gap of length 0", "gap of length 1", ..., "gap of length $t-1$", and "gap of length t or more".
3. Apply the χ^2 statistic to this data

We also need to be careful about counting the gaps to ensure termination. It also helps to look at an example: Consider a subinterval of $(0, 1)$, say, $J = (0, 1/2)$ and consider the numbers $0.2, 0.3, 0.6, 0.7, 0.4, 0.9, 0.8, 0.7, 0.1$. The gaps become:

0.2	0.3	$0.6, 0.7, 0.4$	$0.9, 0.8, 0.7$	0.1
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
gap of length 0	gap of length 0	gap of length 2	gap of length 3	$\in J$

5 Results and Methodology

In this section, we discuss the specifics of our implementation and present results from our experiments. Finally, we present conclusions about our observations - these are present at the end of each section.

5.1 Implementation

I implemented the code as separate routines in .m function files on MATLAB. For general testing and experimentation, I used a live script file (.mlx) which I have also provided. The code is segmented into sections for particular tasks and experiments and can be run mostly independently. The functions used to plot are not included in the Appendix, but are present otherwise. I used generative AI (specifically ChatGPT 4-o) for some plotting, LaTeX outputs (e.g. tables) and code organization (for the larger experiment runs).

The experiments conduced in the "Gap Test" file and some experiments in the lecture notes have been reproduced. This was done for code validation. I also provide multiple statistics for comparison and completeness wherever possible.

I have also used MATLAB vectorization wherever possible to optimize the code. Most routines have been implemented very directly from algorithmic representations. I also rewrote the code to compute the Halton sequence in nay dimension in MATLAB. The implementation was optimized for MATLAB - I also used an online base expansion routine, which I have cited, and the seed generator was also taken from standard resources. The major implementation was for the gap routine. I chose to use a counter-based traversal algorithm to count the gaps (see Appendix).

5.2 KS Test on the Fibonacci Generator

Here are the significance levels for the KS test (to be used as a reference).

		Significance levels		
KS-statistic	Modified statistic	10%	5%	1%
D_N	$D_N\left(\sqrt{N}+0.12+\frac{0.11}{\sqrt{N}}\right)$	1.224	1.358	1.628

Table 1: KS-statistic critical values at different significance levels

(Homework 2.4) We set $n = 1000$ and $m = 2^{31}$ for the Fibonacci generator. We then compute the KS Statistic as well as the Anderson-Darling Statistic:

Statistic	Value
KS Statistic (D_N)	0.0613
Modified KS Statistic	1.9452
Anderson-Darling Statistic (A^2)	18.3630

Table 2: Computed values for KS, Modified KS, and Anderson-Darling statistics

We can immediately conclude that the Fibonacci Generator is a bad generator. It fails at the smallest significance level, for both the KS test as well as the Anderson-Darling test, by a large margin. Thus we can reject the Null hypothesis that the Fibonacci Generator generates numbers from $Unif(0, 1)$. A larger experiment run gives us plots like these:

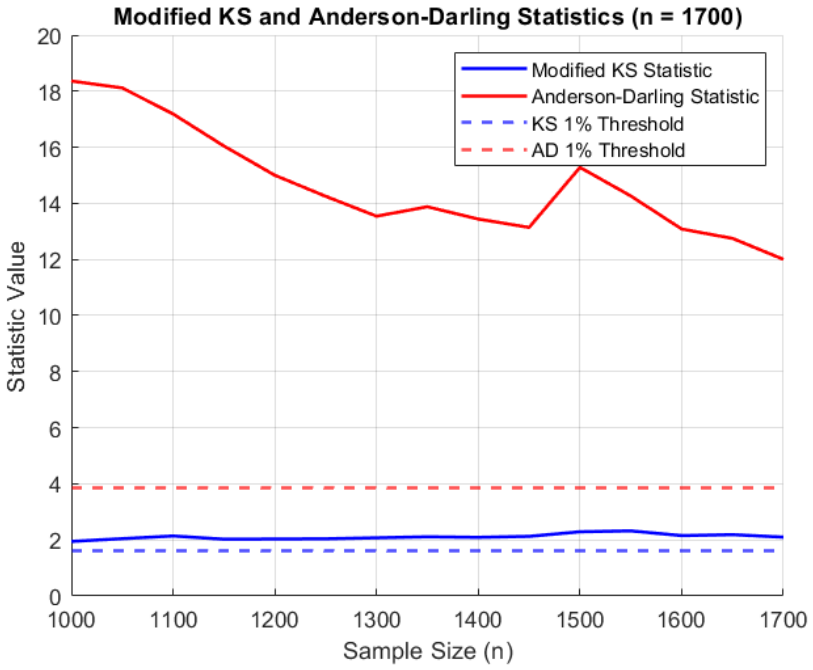


Figure 2: Statistics for the Fibonacci Generator

The Fibonacci generator generally tends to fail for both statistics, and only sometimes passes the 1% threshold for very large values of n. This is further evidence of our earlier hypothesis about this sequence; we expect this sequence to behave badly, due to the high amount of correlation in magnitude. This sequence does pass the basic χ^2 test (which is reproduced in the main testing code), which validates the need for multiple and strong tests.

5.3 Maximum Test on QMC Sequences

Here are the significance levels for the Anderson-Darling test (to be used as a reference).

	Significance levels		
Anderson-Darling statistic	10%	5%	1%
A^2	1.933	2.492	3.857

Table 3: Anderson-Darling statistic critical values at different significance levels

(Homework 2.5) We set $t = 4$, and compute a thousand values (vectors) for both the Van der Corput sequence (Halton in dimension 1) and the 4 dimensional Halton sequence (in prime bases 2,3,5,7). We then apply the maximum test to both, and compute both the KS and the Anderson-Darling Statistic, and present our results in a table.

Statistic	Van der Corput	Halton Sequence
KS Statistic (D_N)	0.3164	0.0074
Modified KS Statistic	10.0447	0.2346
Anderson-Darling Statistic (A^2)	108.9025	0.1306

Table 4: KS, Modified KS, and Anderson-Darling Statistics for Van der Corput and Halton Sequences

The results are very clear: the Van der Corput sequence fails at even the 1% level for both statistics, and the Halton sequence is accepted at at least the 10% confidence level. The value is quite small for the Halton sequence for both statistics. I also reproduced the KS values from the lecture notes for completeness. These results are in line with theoretical predictions. The Halton sequence was tested with blocks of 4, but since the Van der Corput sequence only has dimension 1, we expect it to fail on this test of uniformity. The Halton sequence in dimension 4 has the right size for this test, and each observation of the maximum is done on a single element of the sequence.

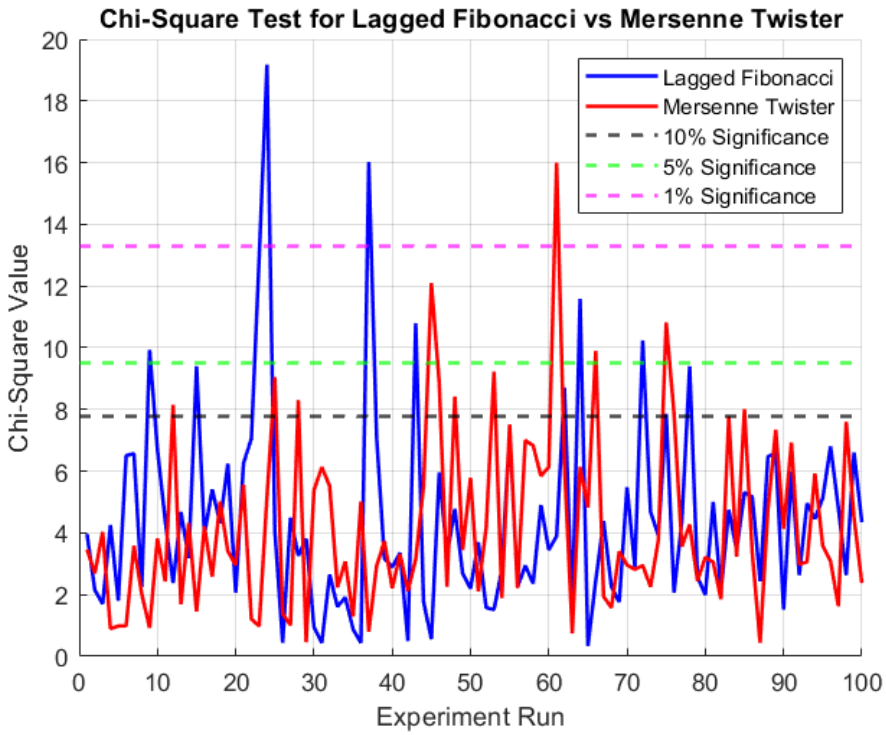
5.4 Gap Test on the Lagged Fibonacci Generator

(Homework 2.6) We want to set the following parameters for testing:

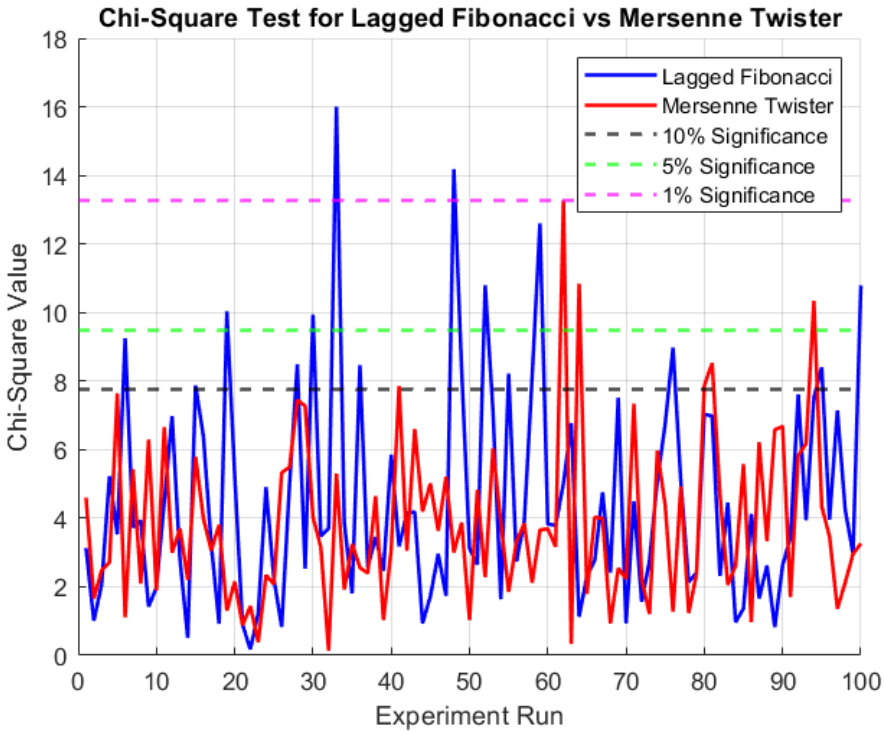
$$n = 2000, k_2 = 1, k_1 = 10, 16$$

for the Lagged Fibonacci Generator. To generate the seed values of size k_1, I did two implementations. One is a standard LCG and the other is the scaled Mersenne Twister. The LCG is the traditional choice for the seed for the Lagged Fibonacci. We then apply the χ^2 test and plot the values for various runs of the experiment (the seed choice is done randomly so we get different values).

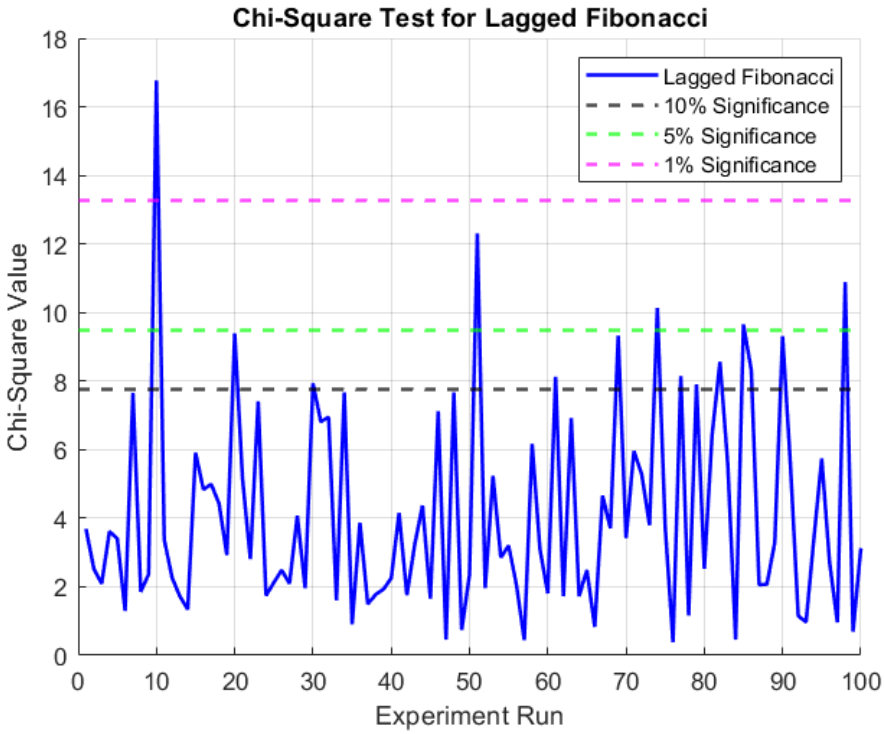
We can observe behavior of changing the parameter k_1, i.e. the larger lag. First we conduct the experiment for $k_1 = 10$, and also plot the required significance levels. We compare with the Gap Test applied to the Mersenne Twister:



If we change k_1 to 16, we get:



Removing the comparison with Mersenne Twister, we get (still at 16 for the lag):



It is clear that the choice of seed determines the behavior of the generator, and the parameter change has less of an effect. We explore this more by plotting both lagged generators on one plot, and comparing. I will make my choice of modulus for the LCG seed generator very small, i.e. 2^3 and then revert back to the standard 2^{31} to see the comparison. The modulus size affects the quality of the initial seed values. We get the following plots:

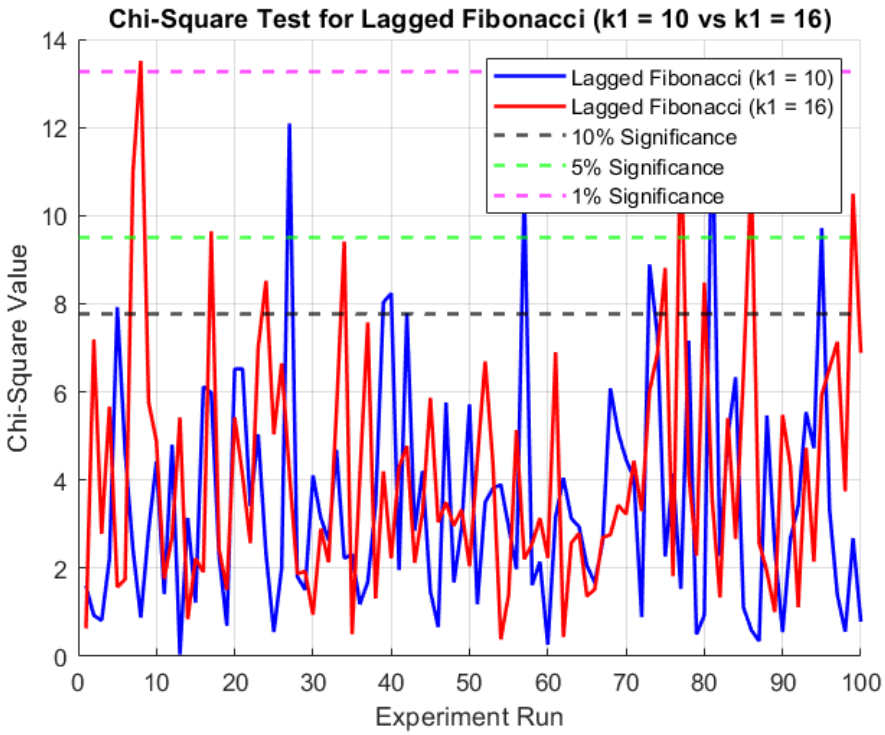


Figure 3: Small Modulus for the Seed

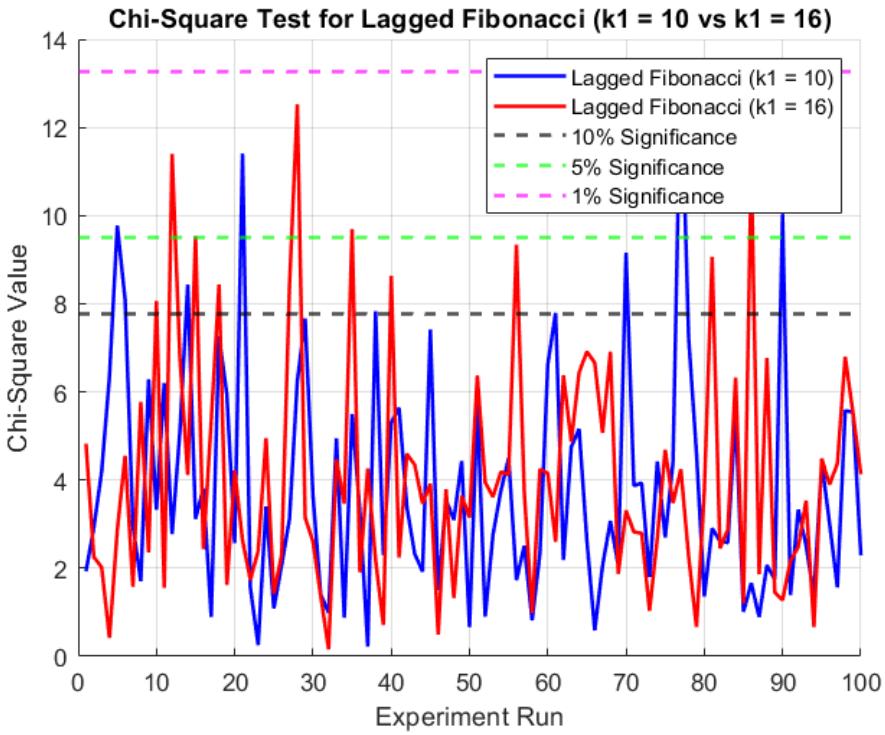


Figure 4: Larger Modulus for the Seed

It is hard to make discerning conclusions about the two parameter choices. We can however, make conclusions about the overall quality of the Lagged generator. In general, few experiment runs result in values for failure at a 1% level, and generally, there is average acceptance of the Null hypothesis at the 5% or 10% significance level for the χ^2 test. This means that the Lagged Fibonacci Generator is a considerable improvement on the standard Fibonacci generator. It also behaves similar to the Mersenne Twister, which has very good properties, so this is more encouraging. Careful choice of the lags may result in further improvement, as we can extrapolate from theoretical interest.

6 Conclusions

We tested and validated number generator routines using statistical tests. Different tests and statistics were used to numerically validate pseudo-random generators (and QMC in one instance), and corroborate theoretical predictions about the statistical properties about these sequences.

7 Code Appendix

7.1 Library Codes

```
function[a] = basexp(k,b)
% BASEXP     Base-b expansion of non-negative integer
% INPUTS   : k - expanded number, non-negative (decimal) integer
%            b - expansion base, decimal integer exceeding 1
% OUTPUTS  : a - 1*j array of expansion coefficients, with the
%                most significant coefficient in a(1)
% EXAMPLE  : basexp(0,2) = 0, basexp(1,2) = 1, basexp(2,2) = [1 1]
% SEE ALSO : DEC2BASE
% AUTHOR   : Dimitri Shvorob, dimitri.shvorob@vanderbilt.edu, 6/20/07
%            (Implementation of algorithm on p. 206 of Glasserman (2003))
if k ~= floor(k)||(k < 0)
   error('Input argument "k" must be a non-negative integer')
end
if b ~= floor(b)||(b < 2)
   error('Input argument "b" must be a positive integer exceeding 1')
end
if k
   j = fix(log(k)/log(b)) + 1;
   a = zeros(1,j);
   q = b^(j-1);
   for i = 1:j
       a(i) = floor(k/q);
       k = k - q*a(i);
       q = q/b;
   end
else
   a = 0;
end
```

[1]

7.2 Random Number Generator Routines

```
function vdc_sequence = vdc(size,base)
%VDC Generates a Van der Corput sequence
% Inputs:
%   size - Number of elements of the sequence to generate
%   base - Prime base of the seqence
% Outputs:
%   vdc_sequence - The Van der Corput sequence of size n and base b

vdc_sequence = zeros(1,size);
for i = 1:size
```

```
    base_exp = basexp(i,base);
    k = length(base_exp);
    vdc = 0;
    for j = 0:k-1
        vdc = vdc + (base_exp(k-(j)) / (base^(j+1)));
    end
    vdc_sequence(i) = vdc;
end
```

```
function halton = halton(size,dim)
%HALTON Generates a Halton sequence
% Inputs:
%   size - Number of elements of the sequence to generate
%   dim - Prime base of the seqence
% Outputs:
%   halton - The Halton sequence of size n and dimension d in the first d
%   prime bases

primes_array = zeros(1, dim);
counter = 0;
num = 2;

while counter < dim
    if isprime(num)
        counter = counter + 1;
        primes_array(counter) = num;
    end
    num = num + 1;
end

halton = zeros(dim, size);
for b = 1:dim
    halton(b, :) = vdc(size, primes_array(b));
end
end
```

```
function U = myfibonacci(size,x_0,x_1, m)
%MYFIBONACCI Generates a psuedorandom Fibonacci sequence
% Inputs:
%   size - Number of elements of the sequence to generate
```

```
%   x_0 - The first seed number
%   x_1 - The second seed number
%   m - The modulus of the generator
% Outputs:
%   U - The Fibonacci sequence of size n scaled to output in (0,1)

a = x_0;
b = x_1;

U = zeros(1, size);
for i = 1:size
    new_val = mod(a+b,m);
    a = b;
    b = new_val;
    if m ~= 0
        U(i) = new_val / m;
    else
        U(i) = new_val;
    end

end
end
```

```
function U = lagged_fibonacci(size, seed, k1, k2, m)
% LAGGED_FIBONACCI Generates a sequence using the Lagged Fibonacci Generator
% Inputs:
%   size - Number of numbers to generate
%   seed - Initial seed array of length max(k1, k2)
%   k1, k2 - Lags (k1 > k2)
%   m - The modulus of the generator
% Outputs:
%   U - The Lagged Fibonacci sequence of size n scaled to output in (0,1)

if k1 <= k2
    error('k1 must be greater than k2');
end
if length(seed) < k1
    error('Seed array must be at least of length k1');
end

state = seed;
U = zeros(1, size);
```

```
for i = 1:size
    new_val = mod(state(end-k1+1) + state(end-k2+1), m);

    U(i) = new_val / m;

    state = [state(2:end), new_val];
end
end
```

```
function seed = generate_seed(size, method)
% GENERATE_SEED Generates a seed array using 'mt' (Mersenne Twister)
                    or 'lcg' (Linear Congruential Generator)
% Inputs:
%   size - Number of seed values to generate
%   method - 'mt' for Mersenne Twister, 'lcg' for Linear Congruential Generator
% Output:
%   seed - Generated seed array of length 'size'

m = 2^31;

if strcmp(method, 'mt')
    seed = floor(rand(1, size) * m);

elseif strcmp(method, 'lcg')
    % Standard LCG
    X0 = randi([0, m-1]); % Random initial seed
    a = 1664525;          % Standard multiplier
    c = 1013904223;       % Increment
    seed = zeros(1, size);
    seed(1) = X0;

    for i = 2:size
        seed(i) = mod(a * seed(i-1) + c, m);
    end
else
    error('Invalid method. Use ''mt'' for Mersenne Twister
            or ''lcg'' for Linear Congruential Generator.');
end

end
```

7.3 Test and Statistics Routines

```
function Q = chi_squared(Y, p, n)
% CHI_SQUARED Computes the chi-squared statistic
% Inputs:
%   Y - Vector of observed event counts (Y_i)
%   p - Vector of true probabilities (p_i) (should sum to 1)
%   n - Total sample size
% Output:
%   Q - Chi-squared test statistic

E = n * p;
Q = sum((Y - E).^2 ./ E);
end
```

```
function ks = KS_statistic(X,F)
% KOLMOGOROV_SMIRNOV Computes the Kolmogorov-Smirnov statistic
% Input:
%   X - Sample of points
%   F - Function handle for the cumulative distribution function (CDF)
% Output:
%   ks - Kolmogorov-Smirnov statistic

N = length(X);

X_sorted = sort(X);
F_sorted = F(X_sorted);

D_plus = max(((1:N) / N) - F_sorted);     % D^+ = max(k/N - F(X_k))
D_minus = max(F_sorted - ((0:N-1) / N));  % D^- = max(F(X_k) - (k-1)/N)

ks = max(D_plus, D_minus);
end
```

```
function modKS = modifiedKS(KS,n)
%MODIFIEDKS Returns the Modifed Kolmogorov-Smirnov Statistic
%   Takes in as input the regular KS and n, and returns the modified KS
modKS = KS*(sqrt(n) + 0.12 + (0.11/sqrt(n)));
end
```

```
function A2 = anderson_darling(X,F)
% ANDERSON_DARLING Computes the Anderson-Darling statistic
% Input:
%   X - Sample of data points
%   F - Function handle for the cumulative distribution function (CDF)
% Output:
%   A^2 - Anderson-Darling test statistic

N = length(X);

X_sorted = sort(X);
Z = F(X_sorted);

sum_term = sum((2 * (1:N) - 1) .* (log(Z) + log(1 - Z(N + 1 - (1:N)))));

A2 = -N - ((1/N) * sum_term);
end
```

```
function max_vals = block_max(X, t)
% BLOCK_MAX_GENERAL Computes max values in blocks of size t (for vectors)
% or max across columns (for matrices).
%
% Inputs:
%   X - Input vector or matrix
%   t - Block size (for vectors only, ignored for matrices)
% Output:
%   max_vals - Max values per block (vector case) or per column (matrix case)

if isvector(X)  % Case 1: X is a vector
    N = length(X);
    num_blocks = ceil(N / t);
    max_vals = arrayfun(@(i) max(X(((i-1)*t+1):min(i*t, N))), 1:num_blocks);

elseif ismatrix(X)  % Case 2: X is a matrix (treat each column separately)
    % Note that this makes t = k where k is the column length
    % This implicilty assumes that k=t for the Max test
    max_vals = max(X, [], 1);

else
    error('Input must be a vector or matrix');
end
end
```

```
function [gaps, P] = gaps(X,alpha,beta,t)
%GAPS Computes the gaps in the sequence for the Gap Test
% Inputs:
%   X - Sequence to compute gaps on
%   alpha, beta - Define the interval J = (alpha, beta)
%   t - Largest gap to count (t or more truncation)
% Output:
%   gaps - An array of frequency of gaps uptil t-1 and then t or more
%   P - Probability values for the distribution of gaps (theoretical)

gaps = zeros(1,t+1);
P = zeros(1,t+1);
p = abs(beta-alpha);
for i = 0:t-1
    P(i+1) = p*((1-p)^(i));
end
P(t+1) = (1-p)^t;

i = 1;
while (i < length(X))
    counter = 0;
    j = i;
    while (X(j) > beta) || (X(j) < alpha)
        counter = counter + 1;
        j = j+1;
        if(j > length(X))
            break;
        end
    end
    if(counter >= t)
        gaps(t+1) = gaps(t+1) + 1;
    else
        gaps(counter+1) = gaps(counter+1) + 1;
    end
    i = i+counter+1;
end
```

7.4 Experiments

```
Simple Routine Testing
I checked these by hand.
You may remove output suppression semi-colons to check code correctness.
I have included known example in the lecture notes wherever possible.
my_vdc = halton(10,1);
my_halton = halton(10,4);
```

```
mylagf = 4*lagged_fibonacci(4,[1,2,7],3,2,4);

myfib = myfibonacci(10,1,1,0);
Random Seed Generation (for Lagged Fibonacci)
mymtseed = generate_seed(4, 'mt');

mylcgseed = generate_seed(5, 'lcg');
Testing Gap Routine
% Example from lecture notes
X = [0.2,0.3,0.6,0.7,0.4,0.9,0.8,0.7,0.1];
A = gaps(X, 0, 1/2, 3)

% My own example
Y = [0,0,0,0.1,0.5,0.6,0.5,0.1,0.1,0.5,0.5,0.2,0.5];
B = gaps(Y,1/3, 2/3, 4);
clear;

Statistical Tests
Basic Fibonacci Generator with the KS statistic of the first 1000 values
n = 1000;
F_uniform = @(x) x; % Uniform CDF Function

fibonacci_rng = myfibonacci(n,1,1,2^31);
% Note the underflow in the starting values
% To see the double precision values, view the full variable

ks_fib = KS_statistic(fibonacci_rng, F_uniform)
modified_statistic = modifiedKS(ks_fib,n)
a2_fib = anderson_darling(fibonacci_rng, F_uniform)

Reproducing t=4 Max Test for KS statistic in the notes for Van der Corput
            and the Halton sequences

t=4
F_exp = @(x) x.^t;      % Exponential CDF (Max Test)

vdc = halton(t*n,1);
haltons = halton(n,t);
max_vdc = block_max(vdc,t);
max_halton = block_max(haltons);

ks_vdc = KS_statistic(max_vdc, F_exp)
modified_ks_vdc = modifiedKS(ks_vdc,n)
ks_halton = KS_statistic(max_halton, F_exp)
modified_ks_halton = modifiedKS(ks_halton,n)
Anderson-Darling Tests (for the same parameters)
```

```
a2_vdc = anderson_darling(max_vdc, F_exp)
a2_halton = anderson_darling(max_halton, F_exp)
Gap Test on the Lagged Fibonacci Generator

% Lagged Fibonacci Parameters
k1 = 16;
k2 = 1;
seed = generate_seed(k1,'lcg');
n2 = 2000;

% Gap Test Parameters
a = 0;
b = 3/4;
t=4;

% Test on Lagged Fibonacci
lag_fibonacci = lagged_fibonacci(n2,seed,k1,k2,2^35);
[gap_fib, Pvals_fib] = gaps(lag_fibonacci,a,b,4);
total_events = sum(gap_fib);
chi2_lagfib = chi_squared(gap_fib,Pvals_fib,total_events)

% Test on Mersenne Twister for comparison
mersenne = rand(1,n2);
[gap_mt, Pvals_mt] = gaps(mersenne,a,b,t);
total_events2 = sum(gap_mt);
ch2_mt = chi_squared(gap_mt, Pvals_mt,total_events2)


Example in the "Gap Test" File on Canvas

N =197;
m = 10^8 + 1;  % Modulus
z = 23;        % Multiplier
s = zeros(1, N);
s(1) = 47594118; % Initial seed

for n = 2:N
    s(n) = mod(z * s(n-1), m);
end

random_numbers = s / m;

my_t = 3;

[my_gaps, myP] = gaps(random_numbers,0,0.5,my_t);
```

```
my_chi2 = chi_squared(my_gaps,myP, sum(my_gaps))
```

References

[1] Dimitri Shvorob. Find base-b expansion of an integer, 2025. MATLAB Central File Exchange. Retrieved February 28, 2025.