Monte Carlo Methods
Homework 2

Daniel Gonzalez
27th of February, 2019

1. *Prove that*

$$\max\left\{F(X_1), \max_{k=1,\dots N-1}\left(F(X_{k+1}) - \frac{k}{N}, \frac{k}{N} - F(X_k)\right), 1 - F(X_N)\right\}$$
$$= \max\left\{\max_{k=1,\dots N}\left(\frac{k}{N} - F(X_k)\right), \max_{k=1,\dots N}\left(F(X_k) - \frac{k-1}{N}\right)\right\}.$$

Proof. Recall that we have ordered the samples $X_1 \leqslant X_2 \leqslant \dots \leqslant X_N$. Also, since F is a cumulative distribution function, it is non-decreasing and right-continuous (i.e. càdlàg). We will prove this equality by cases, relying on the two aforementioned properties.

In the first case, assume that the maximum value on the left-hand side is $1 - F(X_N)$. Then, we have

$$1 - F(X_N) \geqslant \max_{k=1,\dots N-1}\left(F(X_{k+1}) - \frac{k}{N}, \frac{k}{N} - F(X_k)\right)$$

and

$$1 - F(X_N) \geqslant F(X_1),$$

so that, for all $k \in \{1, \dots N\}$,

$$1 - F(X_N) \geqslant \max_{k=1,\dots N}\left(F(X_k) - \frac{k-1}{N}\right)$$

and

$$1 - F(X_N) \geqslant \max_{k=1,\dots N}\left(\frac{k}{N} - F(X_k)\right).$$

Thus,

$$1 - F(X_N) = \max\left\{\max_{k=1,\dots N}\left(\frac{k}{N} - F(X_k)\right), \max_{k=1,\dots N}\left(F(X_k) - \frac{k-1}{N}\right)\right\}.$$

The case when $F(X_1)$ is the left-hand maximum follows similarly to the this case.

In the final case, the maximum on the left-hand side is $\max_{k=1,\dots N-1}\left(F(X_{k+1}) - \frac{k}{N}, \frac{k}{N} - F(X_k)\right)$. This maximum must be a term either of the form $F\left(X_k - \frac{k-1}{N}\right)$ or $\frac{k}{N} - F(X_k)$ for some $k \in \{1, \dots N\}$. In the former case, the maximum becomes $\max_{k=1,\dots N}\left(F(X_k) - \frac{k-1}{N}\right)$, and in the latter case, the maximum becomes $\max_{k=1,\dots N}\left(\frac{k}{N} - F(X_k)\right)$. Thus, in either case, we have

$$\max_{k=1,\dots N-1}\left(F(X_{k+1}) - \frac{k}{N}, \frac{k}{N} - F(X_k)\right) = \max\left\{\max_{k=1,\dots N}\left(F(X_k) - \frac{k-1}{N}\right), \max_{k=1,\dots N}\left(\frac{k}{N} - F(X_k)\right)\right\}.$$

QED

2. *Consider the congruential Fibonacci generator* $x_{n+2} \equiv x_{n+1} + x_n \mod 2^{31}$, *where* $x_0 = x_1 = 1$. *Apply the Kolmogorov-Smirnov test to the first* 1000 *numbers of the sequence obtained by this generator. What are your conclusions?*

Using the code in `fibonacci.hs`, written in Haskell, the first 1000 elements of the given Fibonacci sequence were generated, and the Kolmogorov-Smirnov statistic for this finite sequence was computed using the formula

$$D_N = \max\left\{\max_{k=1,\dots N}\left(X_k - \frac{k-1}{N}\right), \max_{k=1,\dots N}\left(\frac{k}{N} - X_k\right)\right\}.$$

where $N = 1000$. The result was that $D_N = 2.144908972946 \times 10^9$, which is an unsettlingly high number. Just to confirm that everything was computed correctly, the same sequence and the same Kolmogorov-Smirnov statistic were implemented in `fibonacci.py` in Python. The result was the same $D_N = 2144908972.946$, leading to the conclusion that the random number generator should be rejected for non-uniformity.

3. *Show that the serial correlation coefficient ρ is equal to -1 when $n = 2$, provided the denominator is non-zero.*

Proof. Recall that, given a finite sequence $u_1, u_2, \dots u_n$ with $u_{n+1} \coloneqq u_1$, the serial correlation coefficient ρ is defined by

$$\rho \coloneqq \frac{n\sum_{i=1}^{n} u_i u_{i+1} - \left(\sum_{i=1}^{n} u_i\right)^2}{n\sum_{i=1}^{n} u_i^2 - \left(\sum_{i=1}^{n} u_i\right)^2}.$$

When $n = 2$, provided the denominator is non-zero, we obtain

$$\begin{aligned}
\rho &\coloneqq \frac{2\sum_{i=1}^{2} u_i u_{i+1} - \left(\sum_{i=1}^{2} u_i\right)^2}{2\sum_{i=1}^{2} u_i^2 - \left(\sum_{i=1}^{2} u_i\right)^2} \\
&= \frac{2u_1u_2 + 2u_1u_2 - (u_1+u_2)^2}{2u_1^2 + 2u_2^2 - (u_1+u_2)^2} \\
&= \frac{2u_1u_2 + 2u_1u_2 - u_1^2 - 2u_1u_2 - u_2^2}{2u_1^2 + 2u_2^2 - u_1^2 - 2u_1u_2 - u_2^2} \\
&= -\frac{u_1^2 - 2u_1u_2 + u_2^2}{u_1^2 - 2u_1u_2 + u_2^2} \\
&= -1.
\end{aligned}$$

QED

4. *Using any good pseudorandom number generator, let $u(i)$ be the run-ups of length i for $i \in \{1, 2, 3\}$ and $u(4)$ be the run-ups of length 4 or more. Compute the serial correlation coefficient ρ_4 for $u(1), u(2), u(3), u(4)$. Repeat this procedure 100 times to obtain 100 correlation coefficients. Does the serial coefficient test indicate dependency? Can you explain the result intuitively? Based on your conclusions, explain whether one can apply the χ^2 test directly to the run-up counts (assuming that we know the probability of a run-up of length i) like it was applied in the gap test.*

The `run_test.py` file implements the run test as described above (counting runs of length 1, 2, 3, and $\geqslant 4$) using the standard Mersenne Twister in Python. The run-up counts are computed on pseudorandom sequences of length 15 from the Mersenne Twister, and 100 such sets of counts are computed. The serial correlation coefficients for these counts are computed using the definition $\rho_4 \coloneqq \frac{4\sum_{i=1}^{4} u(i)u(i+1) - \left(\sum_{i=1}^{4} u(i)\right)^2}{4\sum_{i=1}^{4} u(i)^2 - \left(\sum_{i=1}^{4} u(i)\right)^2}$.

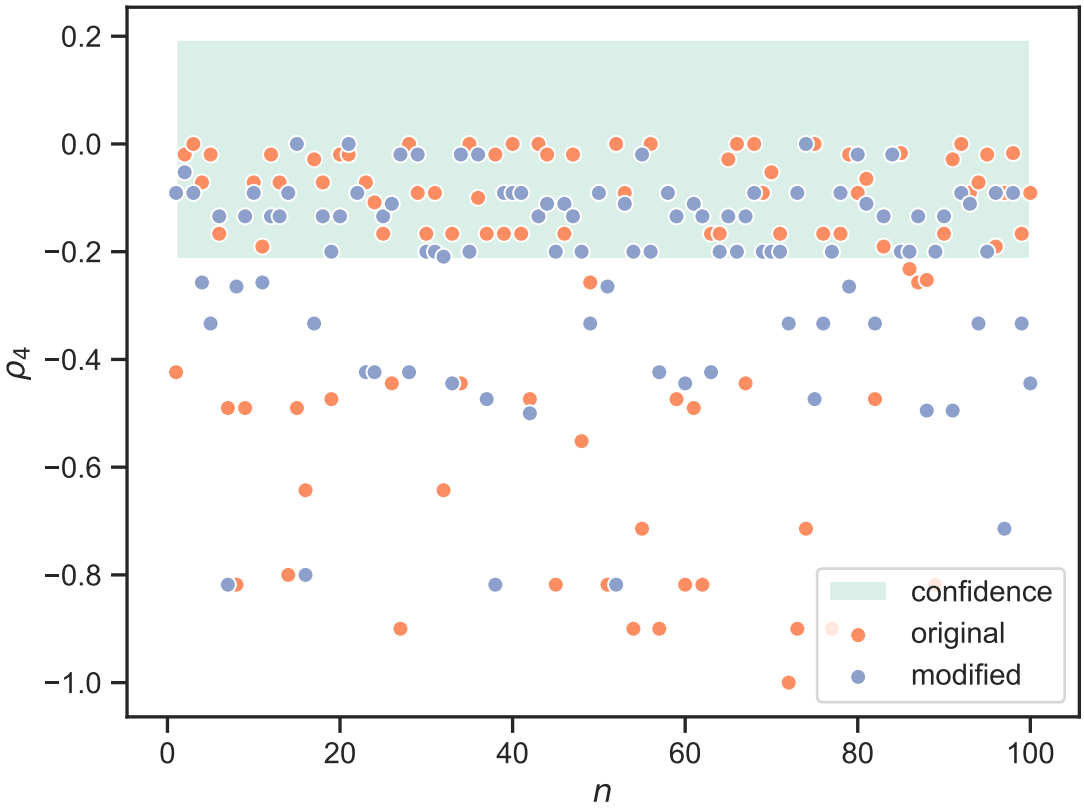
Table 1: Run Test Statistics

μ_{100}	-0.010101
σ_{100}	0.102035
sample mean	-0.239540
sample std deviation	0.238592
95% confidence interval	(-0.214172, 0.193970)

As we can see in Table 1, the serial correlation coefficient is on average not contained within the 95% confidence interval computed by $(\mu_{100} - 2\sigma_{100}, \mu_{100} + 2\sigma_{100})$, where $\mu_n = -\frac{1}{n-1}$ and $\sigma_n^2 = \frac{n^2}{(n-1)^2(n-2)}$. Therefore, we can reasonably conclude that the run-up counts are not uncorrelated enough to be realized by independent, identically distributed random variables. Since one of the major assumptions in the χ^2 test is that the counts $Y_1 = u(1), Y_2 = u(2), Y_3 = u(3), Y_4 = u(4)$ are independent, identically distributed random variables, and that assumption is violated by the serial correlation test, we should not use the χ^2 test directly on the run-ups $u(1), u(2), u(3), u(4)$ to analyze the randomness of the original sequence. This is corroborated by the visualization in Figure 1, which clearly demonstrates just how often the run test events fall outside of the confidence region, and the magnitude with which they deviate.

Intuitively, we should expect that the run-up counts would be negatively correlated since a high numbers of long-length run-ups would reduce the probability of mid-length run-ups and raise the probability of low-length run-ups significantly.

Figure 1: Run Test vs. Modified Run Test



5. *Consider the modified run test described in the assignment.*

   (a) *In the modified run test, argue whether or not the modified run-up events are independent and whether or not a simple χ^2 test can be applied.*

   As we can see in Figure 1, the counts in the modified run test appear to behave similarly to the ones in the unmodified run test. About half of the 100 samples of $\tilde{\rho}$ fall outside of the 95% confidence region, so we should be inclined to reject the hypothesis that the modified $\tilde{u}(1), \tilde{u}(2), \tilde{u}(3), \tilde{u}(4)$ are uncorrelated, and therefore we should not apply the χ^2 test directly to the modified run-up counts. This is further summarized in Table 2, where the average and standard deviation for $\tilde{\rho}$ is presented along with the expected mean and standard deviation and the associated confidence interval, clearly showing that the average modified $\tilde{\rho}$ lies outside the 95% confidence interval.

Table 2: Modified Run Test Statistics

μ_{100}	-0.010101
σ_{100}	0.102035
modified mean	-0.222139
modified std deviation	0.184685
95% confidence interval	(-0.214172, 0.193970)

   (b) *Prove that, in the modified run test, the probability of having a run-up of length n is $\frac{1}{n!} - \frac{1}{(n+1)!}$ and the probability of having a run-up of length n or greater is $\frac{1}{n!}$.*

   Proof. Consider a sequence $x_1, x_2, \ldots x_n \in [0, 1]$ of distinct numbers and consider n sequential positions numbered $1, 2, \ldots n$. In order to have a run-up of length at least n, we would need the elements in positions 1 through n to be in increasing order. The only way for positions 1 through n to be in increasing order is for the largest number to be in position n, since otherwise the number positioned

after the largest number in the sequence will be smaller than it, resulting in a non-increasing sequence, which violates the definition of a run-up. Similarly, the number placed in position $n-1$ must be the second-largest element of the sequence. Continuing in this fashion, the number in position 1 must be the smallest element of the sequence. Therefore, there is only one arrangement of $x_1, x_2, \ldots x_n$ which creates an increasing sequence. Since the total number of possible arrangements of the sequence elements is $n!$, the probability of randomly arranging this sequence in increasing order is $1/n!$. Since all we need in order to have a run-up of length *at least* n is an increasing sequence of length n, with no regard to what comes after, we can conclude that the probability of having a run-up of length n or greater is $1/n!$.

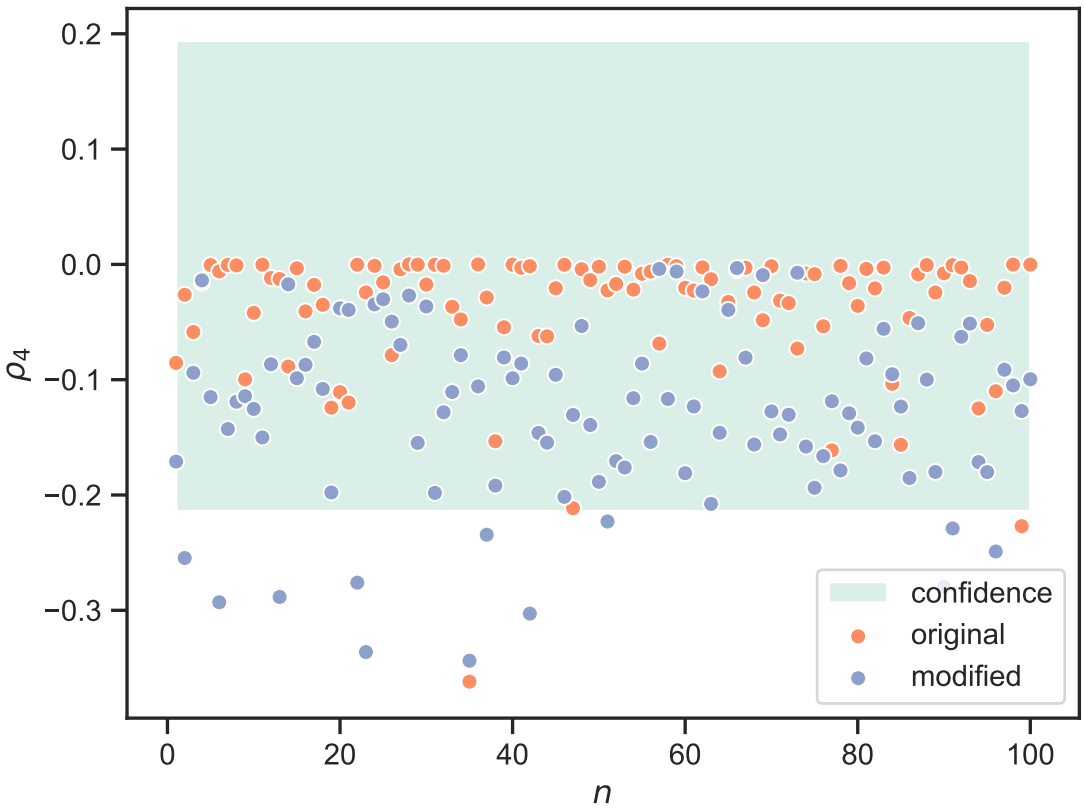
Now, the observing a run-up of length exactly n means observing a run-up of length at least n but *not* a run-up of length $n+1$ or greater. Since the probability of a run-up of length at least n is $1/n!$ and the probability of a run-up of length at least $n+1$ is $1/(n+1)!$, we can conclude that the probability of observing a run-up of length n exactly is $1/n! - 1/(n+1)!$.

QED

(c) *Design a modified run test using parts (a) and (b) and, apply it to the Mersenne Twister.*

Interestingly enough, despite our observations in part (a), the run test and the modified run test seem to start exhibiting independent behaviour if the length of the random sequence taken for computing the run-up counts $u(1), u(2), u(3), u(4)$ and $\tilde{u}(1), \tilde{u}(2), \tilde{u}(3), \tilde{u}(4)$ is long enough. In Figure 2, we can see a comparison between 100 realizations of the serial correlation coefficient for the original and modified run test, but this time, instead of computing the run-up counts on random sequences of length 15, they were computed on random sequences of length 300. We can clearly observe that we have significantly more data points than before falling inside the confidence interval; in fact, nearly ever single point for the unmodified run test fell comfortably inside the 95% confidence region. Therefore, we might think about applying the χ^2 test to the unmodified run-up counts using the probabilities computed in part (b).

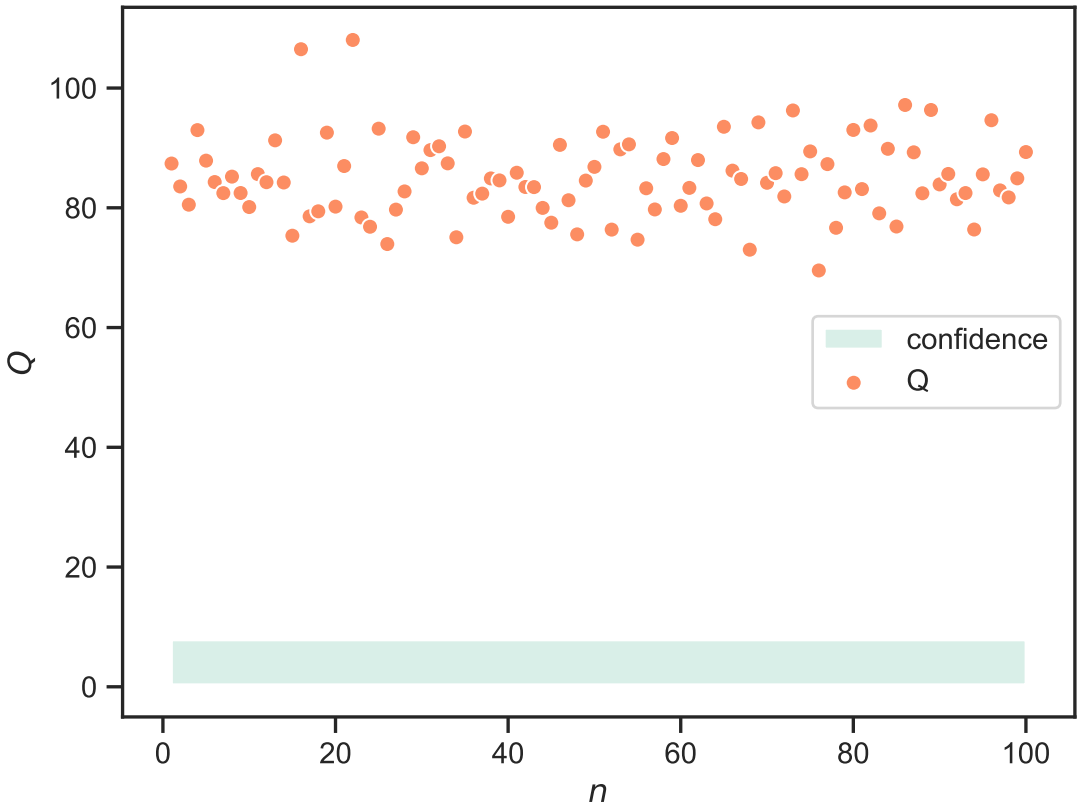
Figure 2: Long Run Test vs. Long Modified Run Test



In designing the χ^2 test, which is implemented by the `chi_squared` function in `run_test.py`, we actually iterated 100 independent χ^2 tests on Mersenne Twister sequences of length 300, and the number of times

each individual χ^2 test was accepted or rejected was recorded. The final decision to accept or reject was based on a majority vote, to take into account the fact that, in Figure 2, not all of the points fell inside the confidence region, so there might be some rare cases when the run-up counts would be serially correlated. However, as we can see in Figure 3, every single iteration of the test was rejected for being outside of the 95% confidence region given by $(0.352, 7.815)$, despite the fact that the Mersenne Twister sequence is known to be well-behaved and sufficiently uniform to pass the χ^2 test. This should let us reasonably conclude that the χ^2 test cannot be applied directly to the run test to test the uniformity of a pseudorandom sequence.

Figure 3: Long Run Test vs. Long Modified Run Test



6. *Design a statistical test for random number generators based on the result described in the assignment. Then, apply the test to any random number generator you want and explain the results.*

   The result described says that, given a Bernoulli random variable C with probability of success p and probability of failure $1 - p$, the random variable X that counts the number of trails necessary before as many successes have happened as failures has the probability density function

$$\mathbb{P}\{X = 2n\} = \frac{p^n(1-p)^n}{2n-1}\binom{2n}{n}.$$

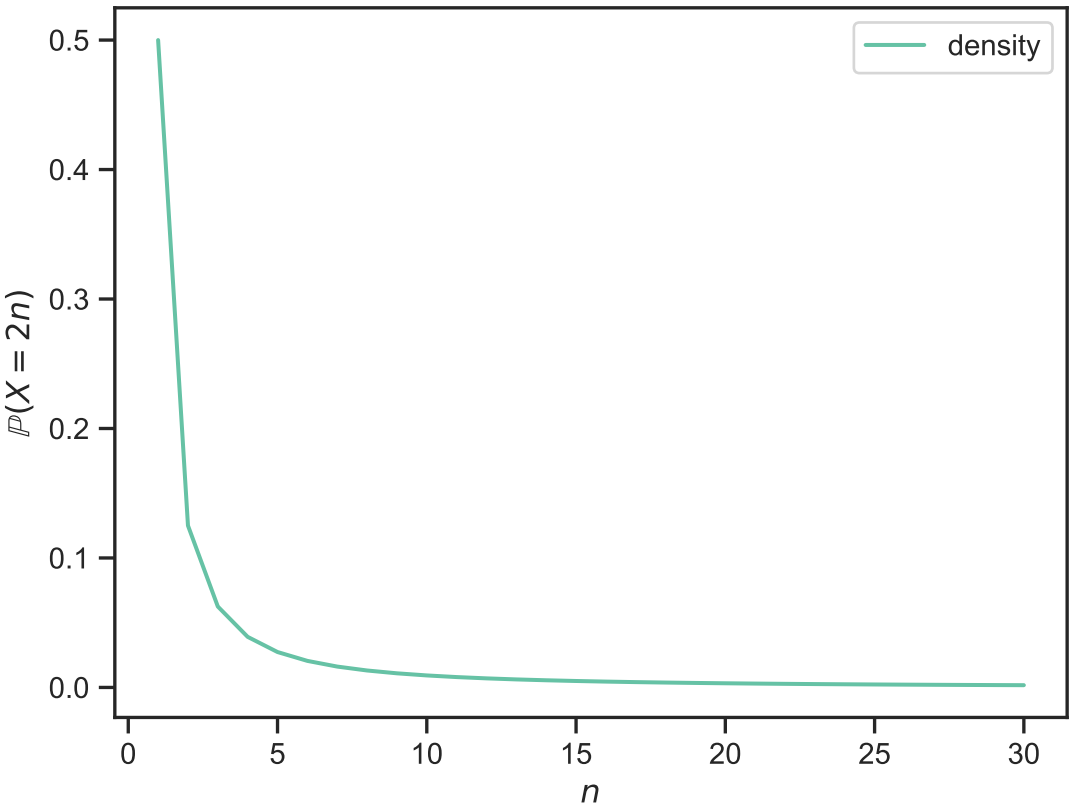
   We can then design a (weak) statistical test as follows: given a sequence $x_1, x_2, \ldots$ in $[0, 1]$, we define

$$C(i) := \begin{cases} \text{success} & \text{if } x_i \geqslant 0.5 \\ \text{failure} & \text{if } x_i < 0.5, \end{cases}$$

   thus obtaining a sequence of successes and failures given by $C(1), C(2), \ldots$ for which we can compute the value X described above with $p = 1/2$ in a test for uniformity. Basically, this will check that a pseudorandom sequence does not take too long to generate as many values in the interval $[0, 0.5)$ as in the interval $[0.5, 1]$; if a given sequence does take too long, then it is likely not uniform enough to be useful.

The probability density function of X is shown in Figure 4 in the case that $p = 1/2$. As we can clearly see, the probability density function is monotonically decreasing. To see what value of X gives us a 95% significance level, a function called `significance` was implemented in `X_test.py` which uses the density formula for X with $p = 1/2$ and iterates n until the residual probability (i.e. 1 minus the sum of the previous probabilities) is below 0.05. The algorithm thus give us the value of n for which $\mathbb{P}(X > 2n) < 0.05$, which was computed to be $n = 128$.

Figure 4: Density of X when $p = 1/2$



Therefore, in our statistical test, we can compute the value of X for the sequence $C(1), C(2), \ldots$ and check to see how it compares to the significance value $n = 128$ under the null hypothesis that the sequence $C(1), C(2), \ldots$ has probability $p = 1/2$ of success. If $X \geqslant 128$, then we know with at least 95% confidence that the probability of success for C must not have been $p = 1/2$, and so the values in the sequence $x_1, x_2, \ldots$ must be biased towards $[0, 0.5)$ or $[0.5, 1]$, instead of being uniformly distributed among the two intervals.

This statistical test was implemented in the `X_test.py` file and applied on the standard Mersenne Twister in Python. The test was performed 100 times (using different seeds), and statistics regarding the tests are summarized in Table 3. As we can see, the Mersenne Twister was accepted 93% of the time with a median value of 2 and an average value well within the 95% confidence interval.

Table 3: X Test Statistics

Number of Acceptances	93
Number of Rejections	7
Sample Average of X	72.94
Sample Median of X	2

Therefore, the Mersenne Twister passes our test, and we can conclude that the Mersenne Twister sequence is, in the sense described above, uniformly distributed among the intervals $[0, 0.5)$ and $[0.5, 1]$.