MFE405 Project 1
Yong Jia (Justin) Tan

1. Generate Uniform Distribution Using LGM

Using the LGM method and the built-in function (uniform_real_distribution) to generate 10,000 uniformly distributed random numbers on [0,1], I find the following empirical results:

	LGM Method	**Built-In Function**
Mean	0.501677	0.501851
Standard Deviation	0.286702	0.290912

Comparing the results, it seems that the built-in function has slightly larger mean and standard deviation compared to my implemented LGM uniform random number generator. However, the differences are minimal, and this could just be arbitrary.

2. Generate Discrete Distribution

Using the 10,000 uniformly distributed random numbers from above, I generate 10,000 random numbers with the following discrete distribution:

$$X = \begin{cases} -1 \ with \ probability \ 0.30 \\ 0 \ with \ probability \ 0.35 \\ 1 \ with \ probability \ 0.20 \\ 2 \ with \ probability \ 0.15 \end{cases}$$

The histogram and empirical results for the simulated numbers from this distribution is shown below:

```
-1:     ************************************************************
0:      ********************************************************************************
1:      ***************************************
2:      ******************************
```

(Histogram is also generated in code as "discrete.txt".)

	Discrete
Mean	0.1966
Standard Deviation	1.02866

3. Generate Binomial(n=44, p=0.64) Distribution

Now I will generate 1,000 random numbers with a binomial distribution of n=44 and p=0.64. In order to generate these 1,000 numbers, I will need to first generate 44,000 random uniformly distributed numbers, using the same method as described earlier. Then, I simulate the binomial distribution as a sum of Bernoulli distributed random variables.

The histogram and empirical results for the binomial distributed random numbers are as below:

(Histogram is also generated in code as "binomial.txt".)

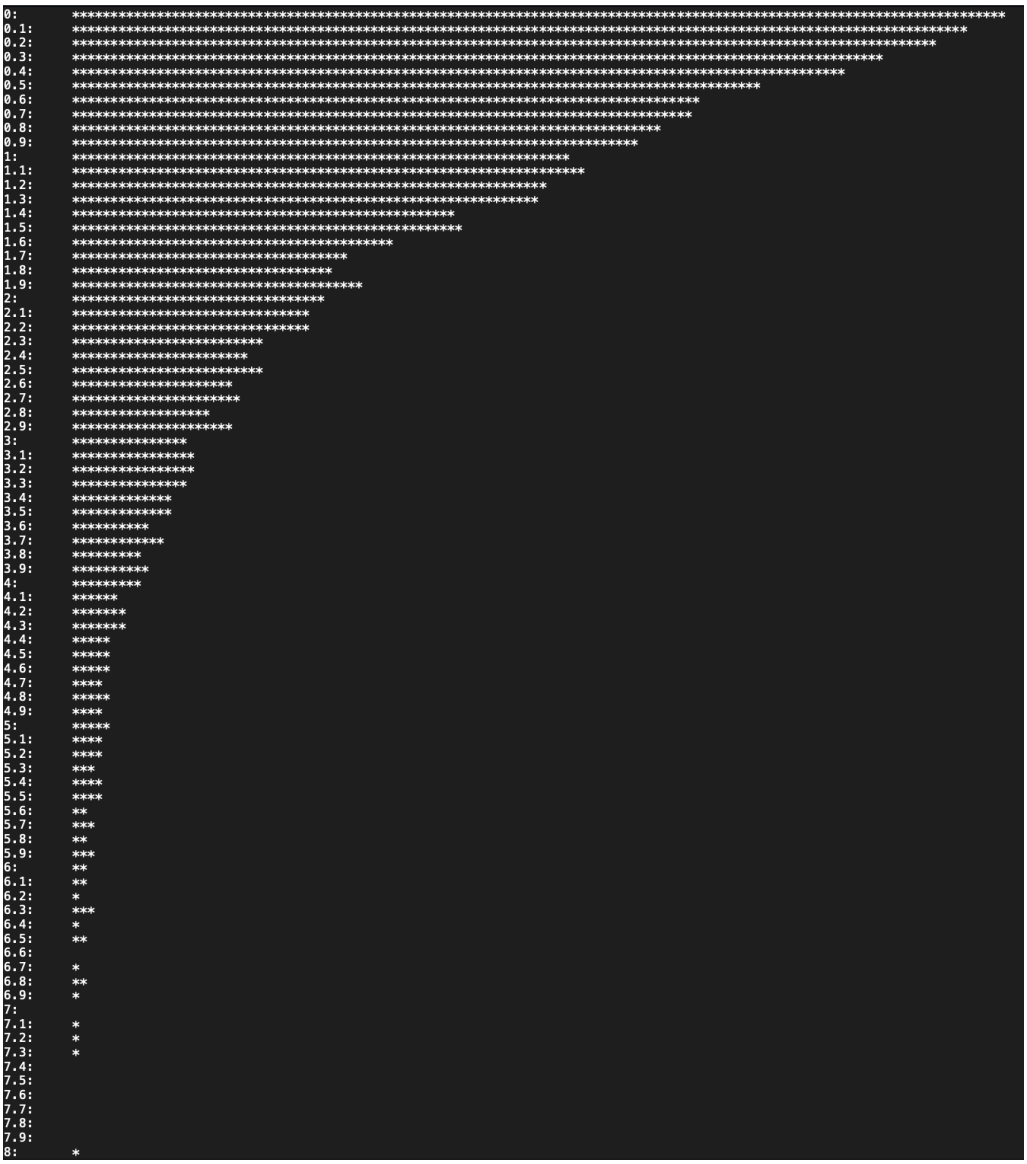
	Binomial
P(X ≥ 40)	0

From online resources, the exact calculated probability of P(X ≥ 40) is around 0.00008, which is very close to our result, 0, as well. If we simulate more than 1,000 random binomial numbers, we should approach this result.

4. Generate Exponential Distribution

Now, using the 10,000 generated uniformly distributed random numbers from the beginning of this project, I generate 10,000 exponentially distributed random numbers with parameter $\lambda = 1.5$. The empirical results and histogram are shown below:

	Exponential
Mean	1.5066
Standard Deviation	1.5137
P(X ≥ 1)	0.5169
P(X ≥ 4)	0.0684



(Histogram is also generated in code as "exp.txt".)

5. Generate Normal Distribution (Box-Muller,Polar-Marsaglia)

I now attempt to generate normally distributed random numbers. First, I generate 5,000 uniformly distributed random numbers using the same method as above. Then, I generate 5,000 normally distributed random numbers using the Box-Muller method. I now use the same 5,000 uniformly distributed random numbers to generate normally distributed random numbers using the Polar-Marsaglia method. However, since the Polar-Marsaglia has an acceptance-rejection criterion. Therefore, in order to compare execution times fairly, I generated a new set of 8,000 uniformly distributed random numbers and used them to generate 5,000 normally distributed random numbers using the Polar-Marsaglia method.

The empirical results, as well as the execution times in one arbitrary execution of the program, for all 3 sets of generated normally distributed random numbers, are in the table below:

	Box-Muller	**Polar-Marsaglia (< 5,000)**	**Polar-Marsaglia (= 5,000)**
Mean	0.00663234	-0.000764985	-0.00697809
Standard Deviation	0.983168	1.01838	1.0167
Execution Time (Microseconds)	433	286	315

In this execution instance, and in nearly all execution instances of generating 5,000 random normal variables, Polar-Marsaglia was faster than Box-Muller in execution time. If I change the code to generate larger amounts of normally distributed random numbers, the difference between execution times for Box-Muller and Polar-Marsaglia was even more obvious, with Polar-Marsaglia always outperforming Box-Muller (*amounts were adjusted to 10,000 and 20,000 for testing, however is not evident in final submission of code*).

Thus, I can conclude that in nearly all cases, especially when the amount of numbers to generate is fairly large, the Polar-Marsaglia Method is more efficient than the Box-Muller Method.