Emile Goulard
uID: u1244855
CS3200
Assignment 2 Report

Question 1

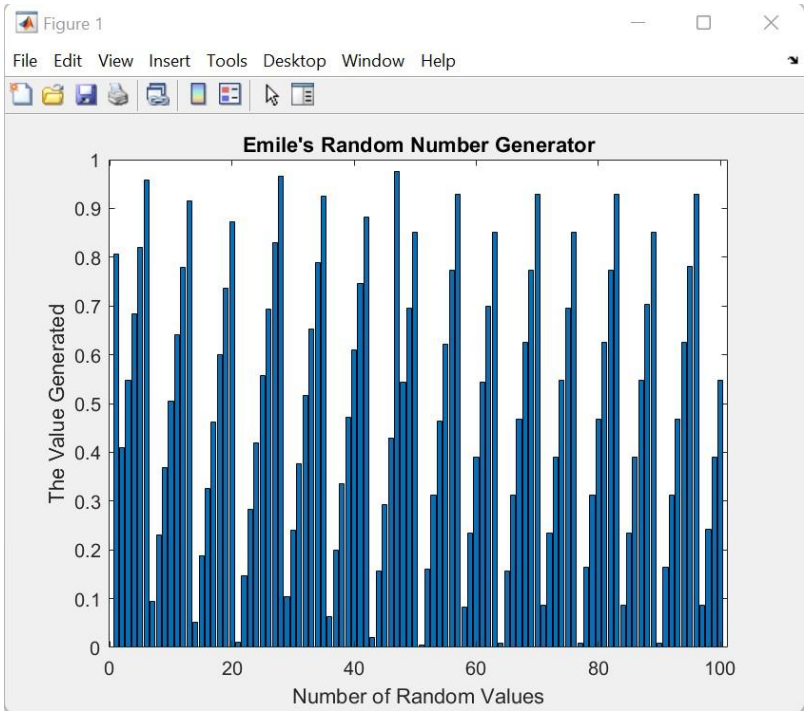
Context: The file: 'A2_RNGClass.m' contains the functions 'MyRNG' and 'BuiltInRNG' that are called in 'A2_RandomNumberGenerator.m' to compare and contrast my random number generator to MATLAB's built-in random number generator. How to run it is given in the README file.

a) My method for building a random number generator went as followed:

- I started with creating a 'start' variable given a 0 value. This will come into play during the for-loop which loops from 1 to n (number of random values).
- The important piece that makes the random numbers is using MATLAB's built-in date/time system. Conceptually, I wanted to always keep track of the current time clocked in the computer to generate random numbers as time is always incrementing. Thus, I created a 'dateSerialNumber' that holds the current date as a unique serial number.
- From there, inside the for-loop, I wanted to clamp the values between [0,1] without using modulo. Thus, I looked up a way to grab the decimal value of any number (i.e. "4.56" becomes "0.56") and found a formula using:
  `sign(dateSerialNumber) * (abs(dateSerialNumber) - floor(abs(dateSerialNumber)));`
  to store the decimal value of the current date multiplied by some constant.
- Afterwards, I reset the 'dateSerialNumber' back to a smaller value by multiplying the current date/time to itself. This avoids any unknown values being accounted for (since I'm incrementing 'dateSerialNumber')
- Lastly, at the random values array's current index, I check to see if the value is strictly 0 or 1 and then adjust the values by a decimal point. Although this makes random values plateau at larger n (as I'll explain later), it ensures that all the data has been plotted.

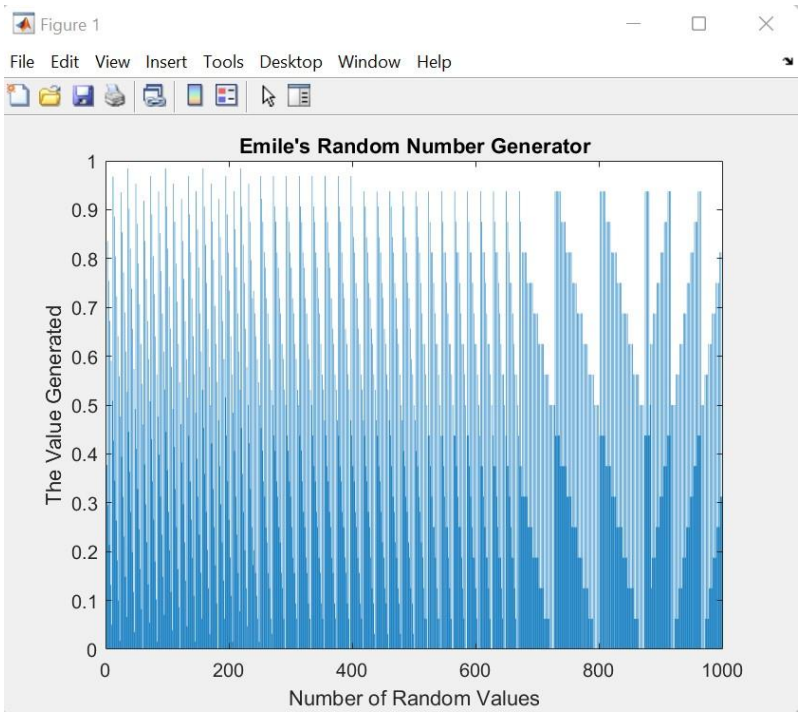
b) My RNG plots:

(Figure 1a)



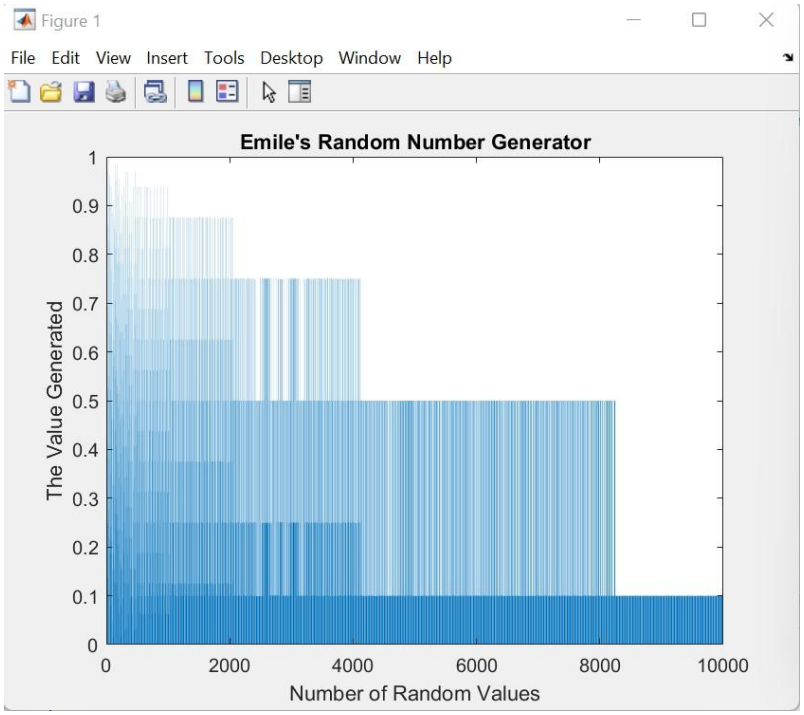
(n=100)

(Figure 2a)



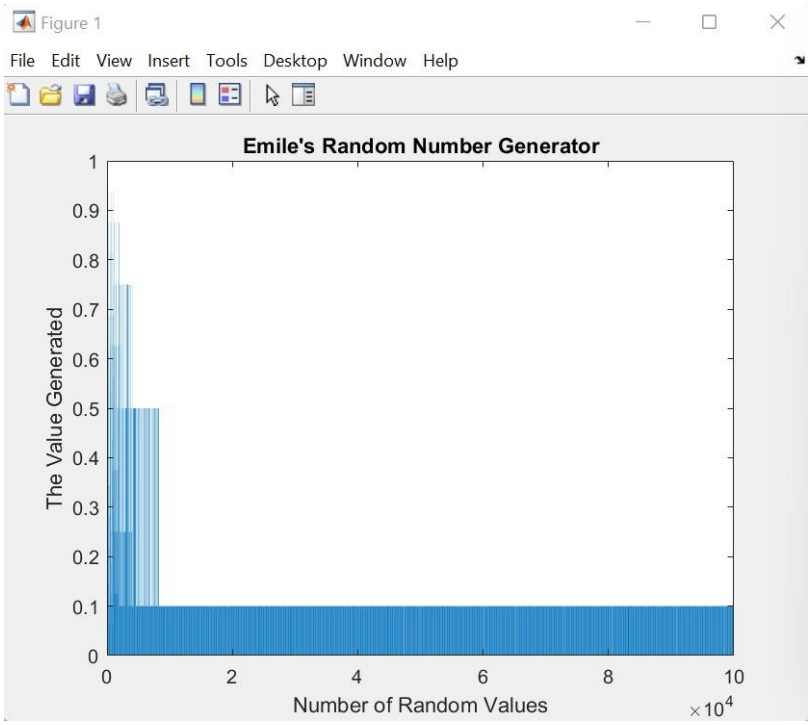
(n=1000)

(Figure 3a)



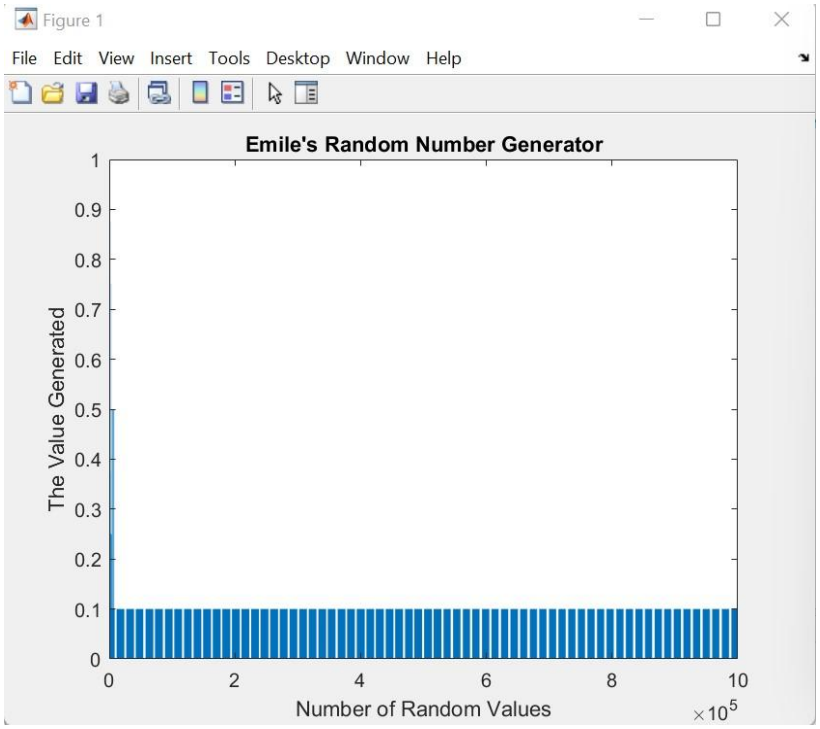
(n=10,000)

(Figure 4a)



(n=100,000)

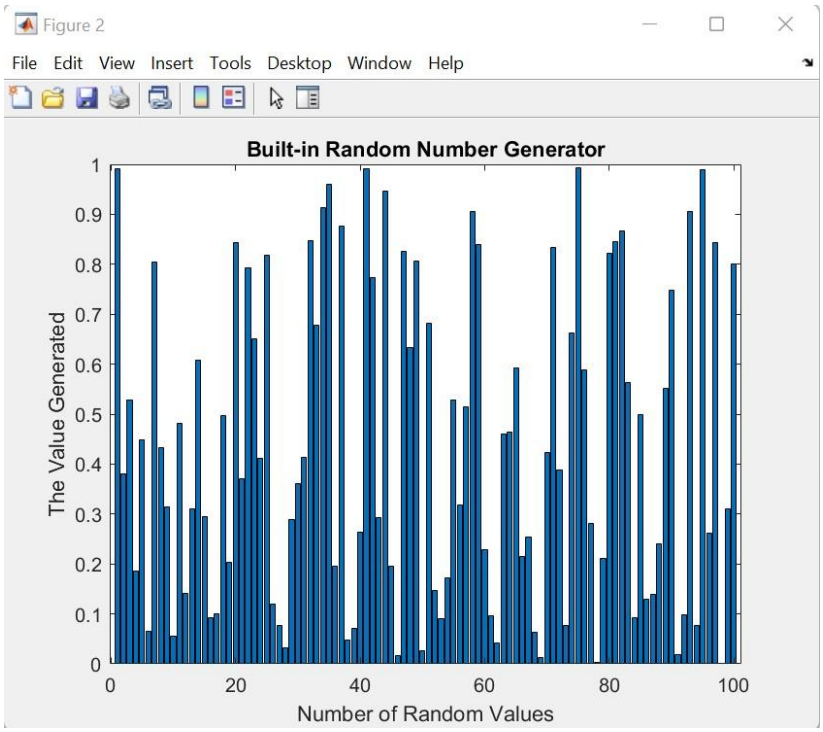
(Figure 5a)



(n=1,000,000)

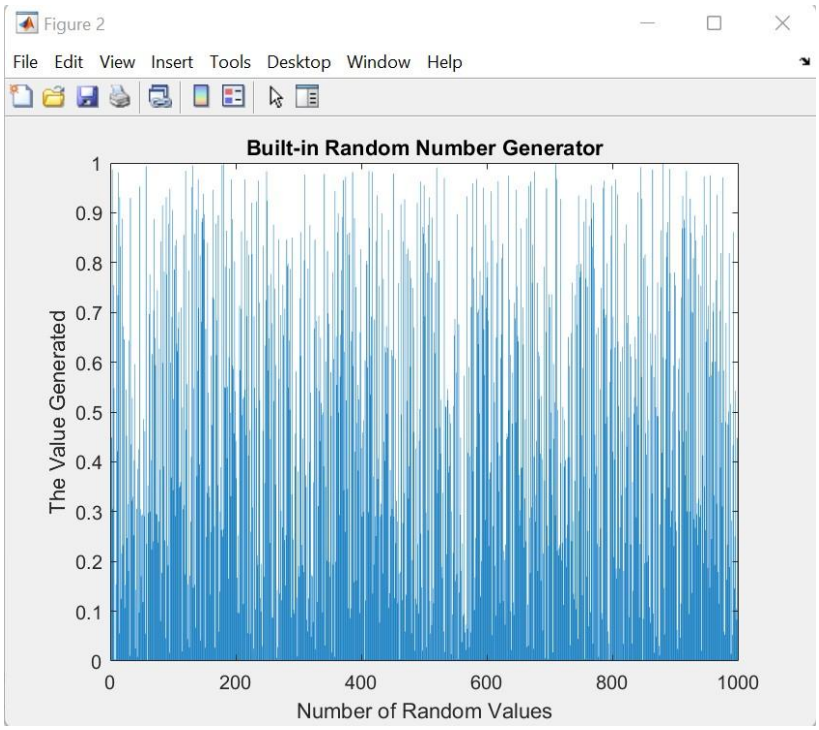
c) Built-In RNG plots:

(Figure 1b)



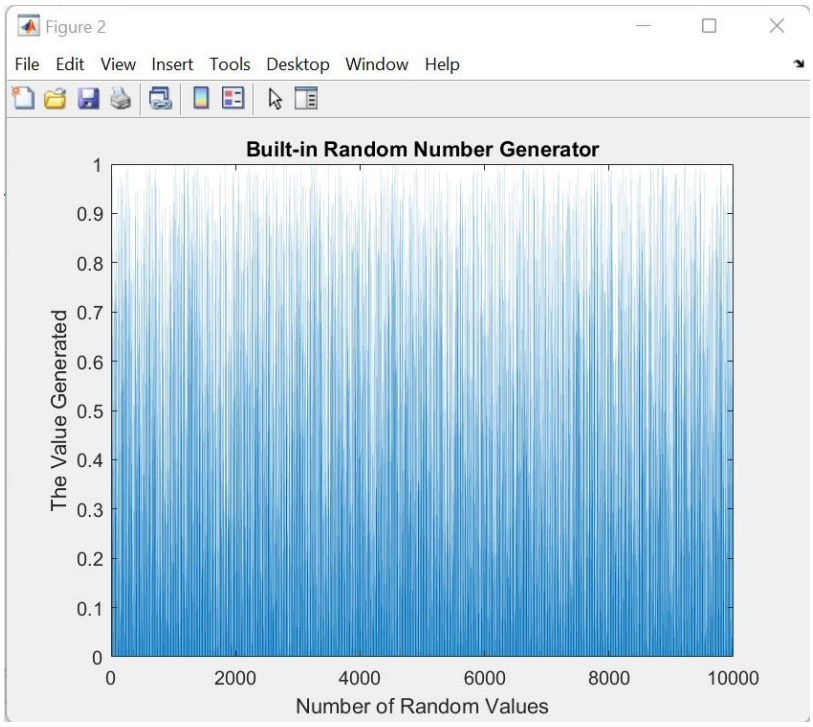
(n=100)

(Figure 2b)



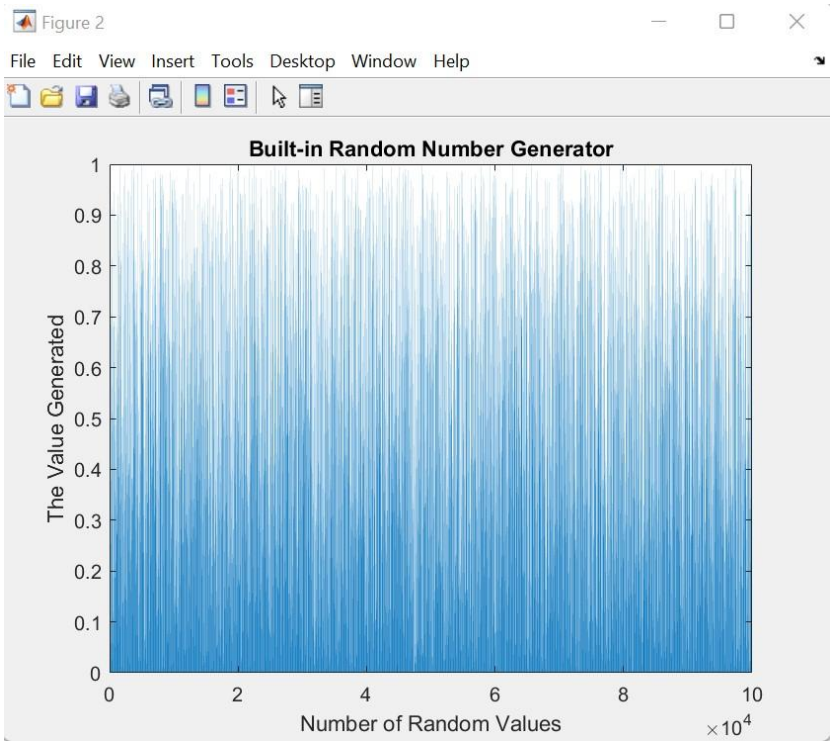
(n=1,000)

(Figure 3b)



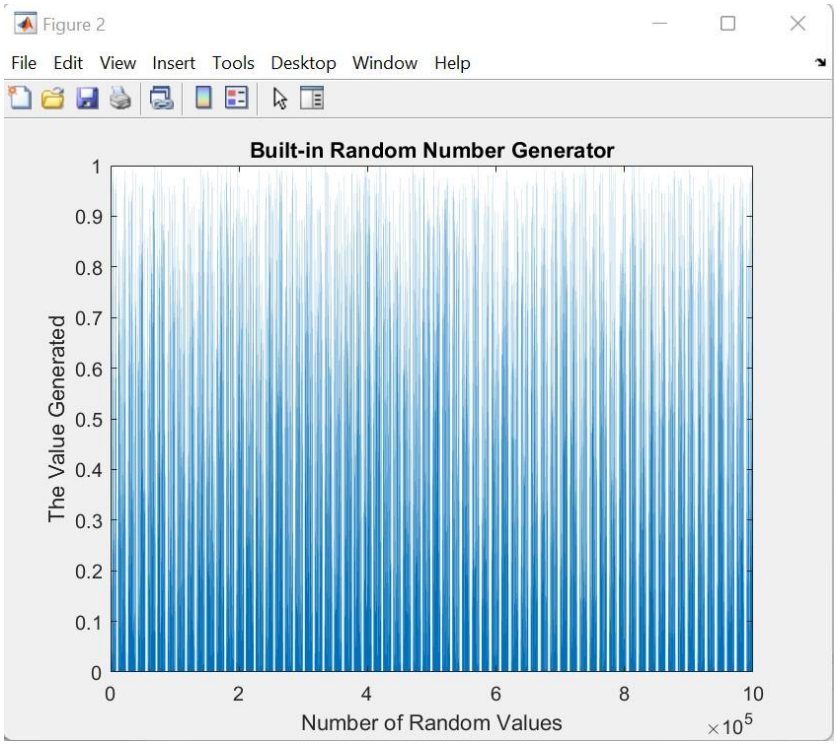
(n=10,000)

(Figure 4b)



(n=100,000)

(Figure 5b)



(n=1,000,000)

d) My RNG vs. Built-In RNG

As expected when running both programs, the Built-In RNG is much more uniform than My RNG. I didn't plot this on a histogram, instead I used a bar graph; however, it's still obvious to see that in My RNG it quickly plateaus with more random values being generated than the Built-In RNG. This means that there are more non-unique numbers being generated in My RNG and thus is non-uniform as expected. I was still surprised that, with relatively small size n, it could generate random numbers uniformly. I believe that it starts to plateau because my variables reach null eventually because they become so large such that I have to manually reset my date time value which creates noticeable patterns (as seen in Figure 3a and Figure 4a). My RNG, at smaller size n, also generates noticeable patterns (as seen in Figure 1a and Figure 2a) but can also generate uniform numbers upon running it a few times in MATLAB.

I find My RNG most similar to the Built-In RNG when at small sizes of n. In Figure 1a and Figure 2a, you can notice that there is some variation (even if Figure 1a is fairly non-uniform). This is more noticeable when running the program a few times on sizes between n=100 and n=1,000.

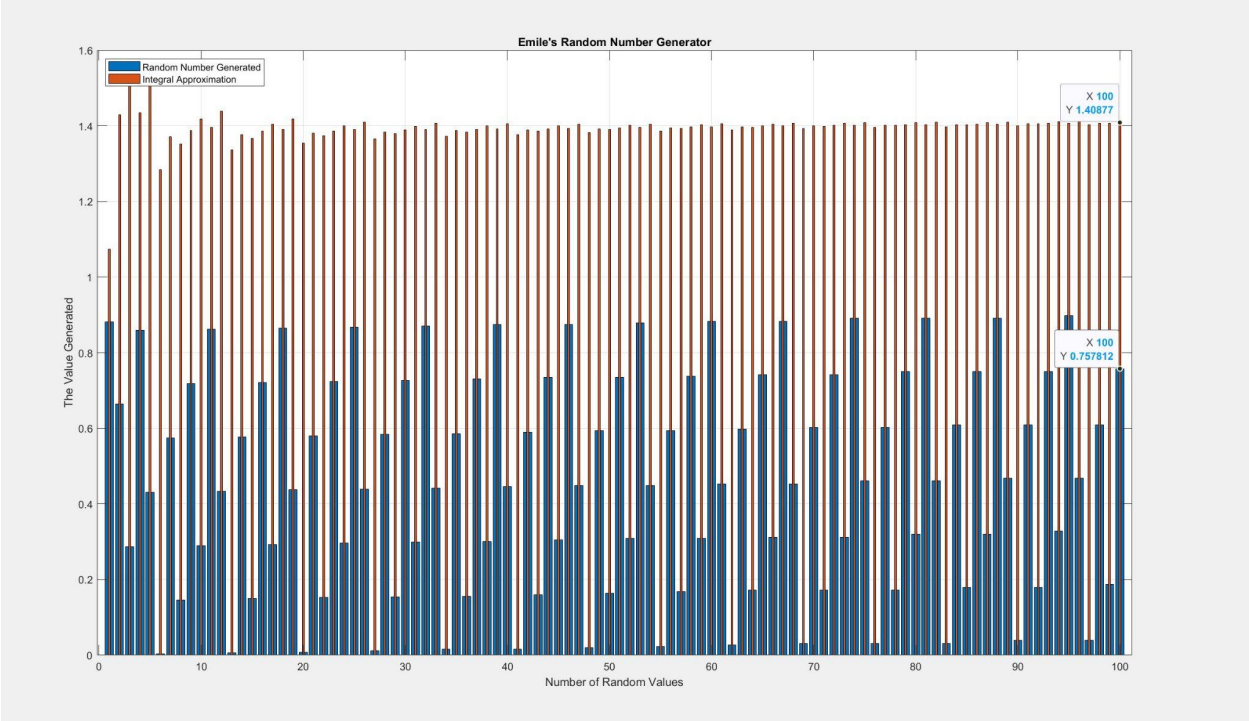
Despite this, it is without a doubt very difficult to make a good RNG from scratch because there are a lot of variables to consider such as calculations, variables, unknown variables, and, most importantly, how it handles RNG at larger values of n. It should be very apparent when comparing the graphs side by side (at the same n values) that my formulas start to fall apart where the Built-In RNG continues to excel.

Question 2

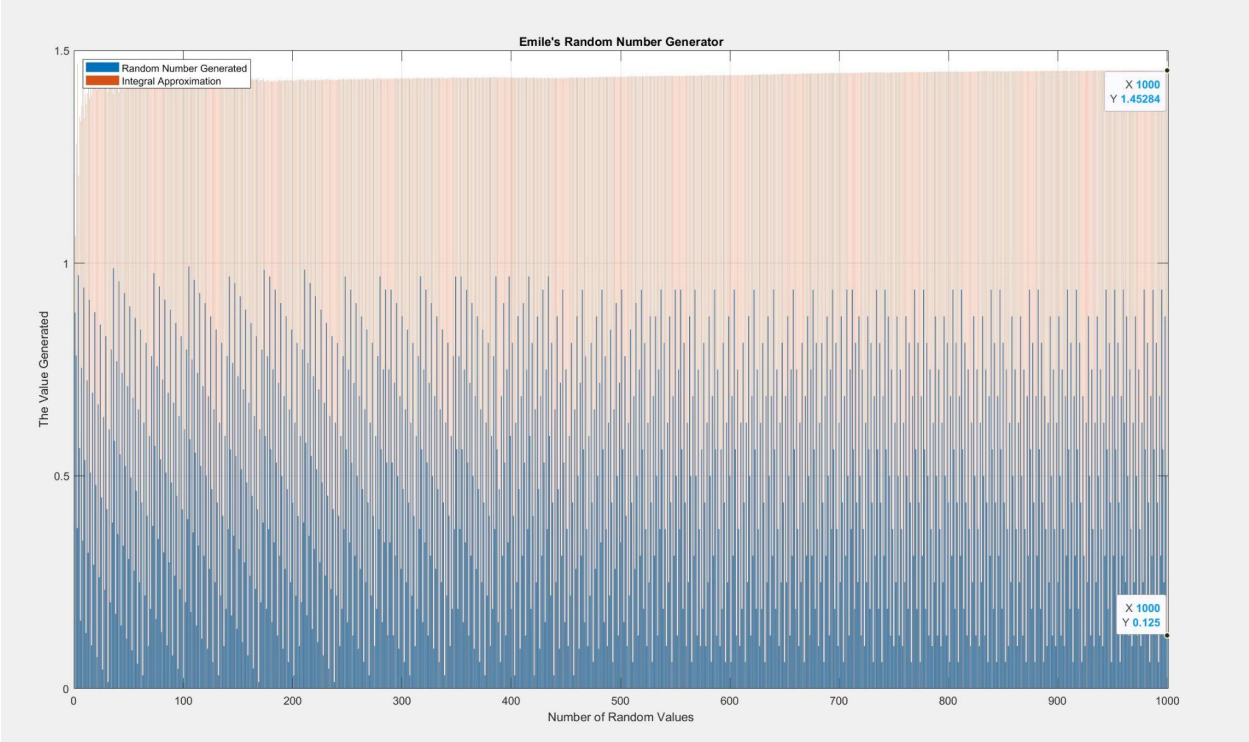
Context: The file: 'A2_MonteCarloMethod.m' should be ran to consider how My RNG and the Built-In RNG compare when evaluating the one dimensional integral using Monte Carlo Method. How to run it is given in the README file.

a) My RNG plots to calculate Monte Carlo Approximation:

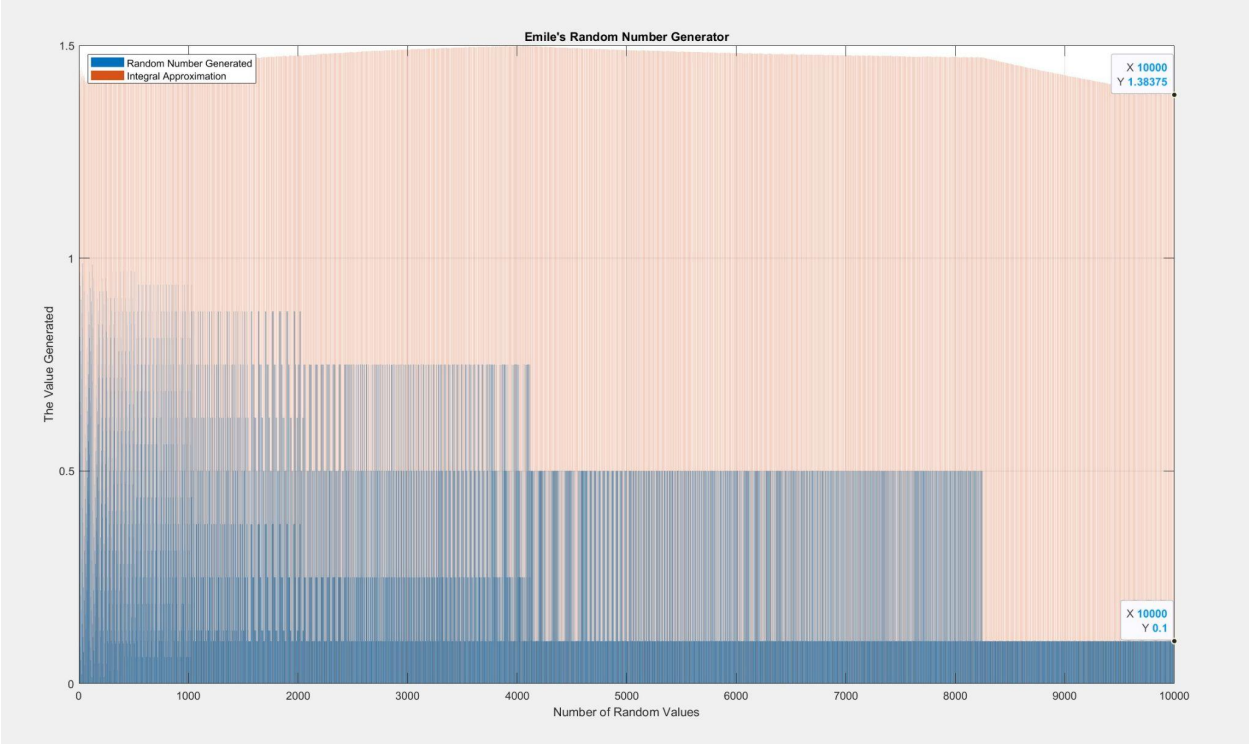
(Figure 1a) (n=100)



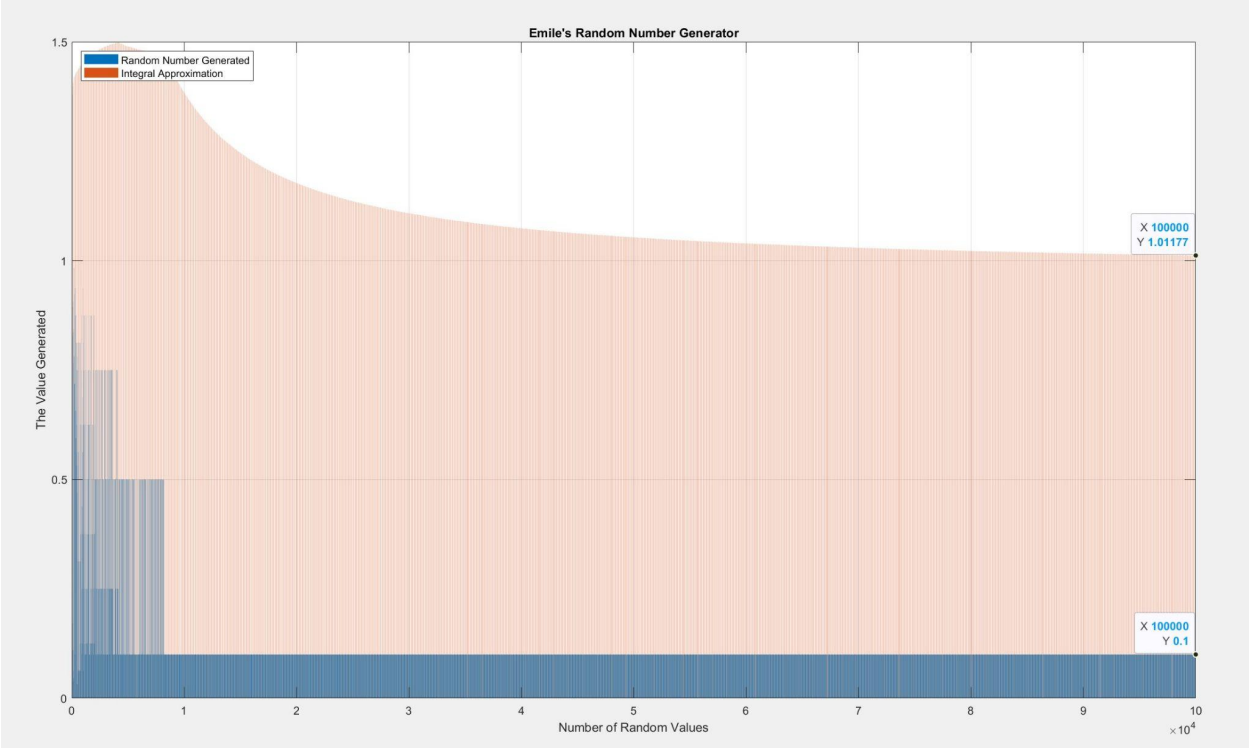
(Figure 2a) (n=1,000)



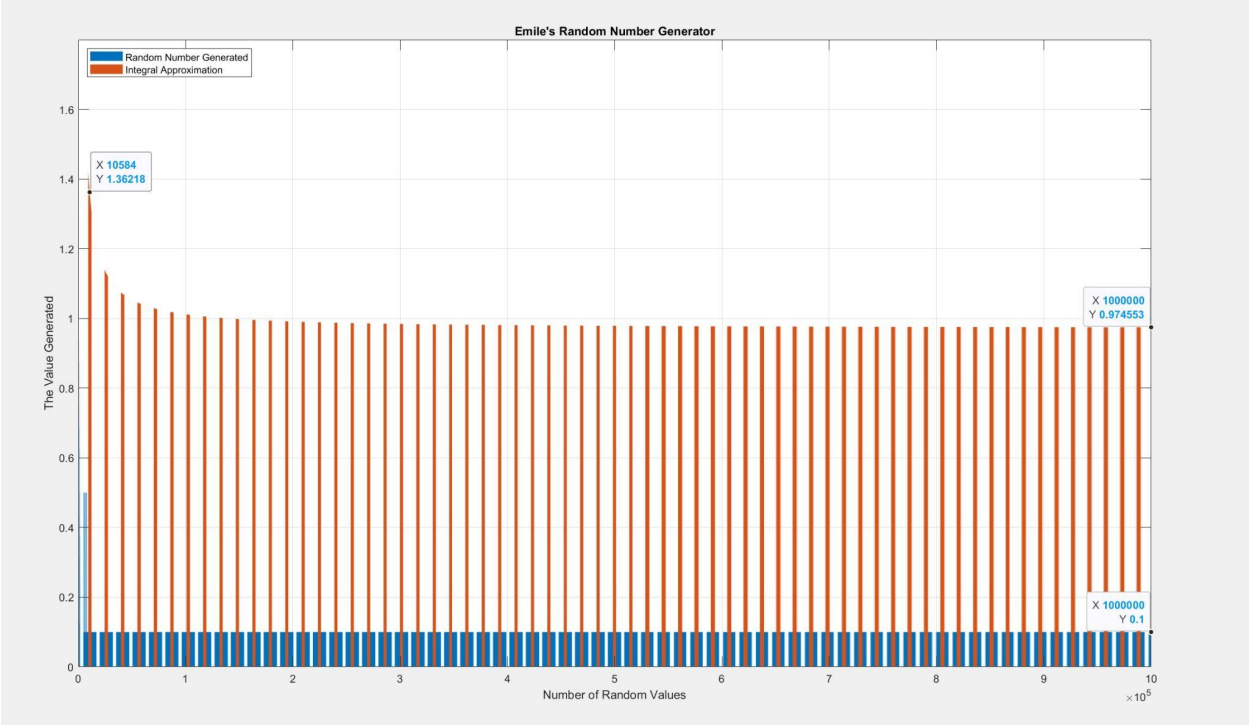
(Figure 3a) (n=10,000)



(Figure 4a) (n=100,000)



(Figure 5a) (n=1,000,000)



b) My Solution vs. Analytic Solution

To calculate the integral using the Monte Carlo approach, I used a for-loop that evaluates the f(x) function, calculates the sum, and uses the integral approximation function shown here:

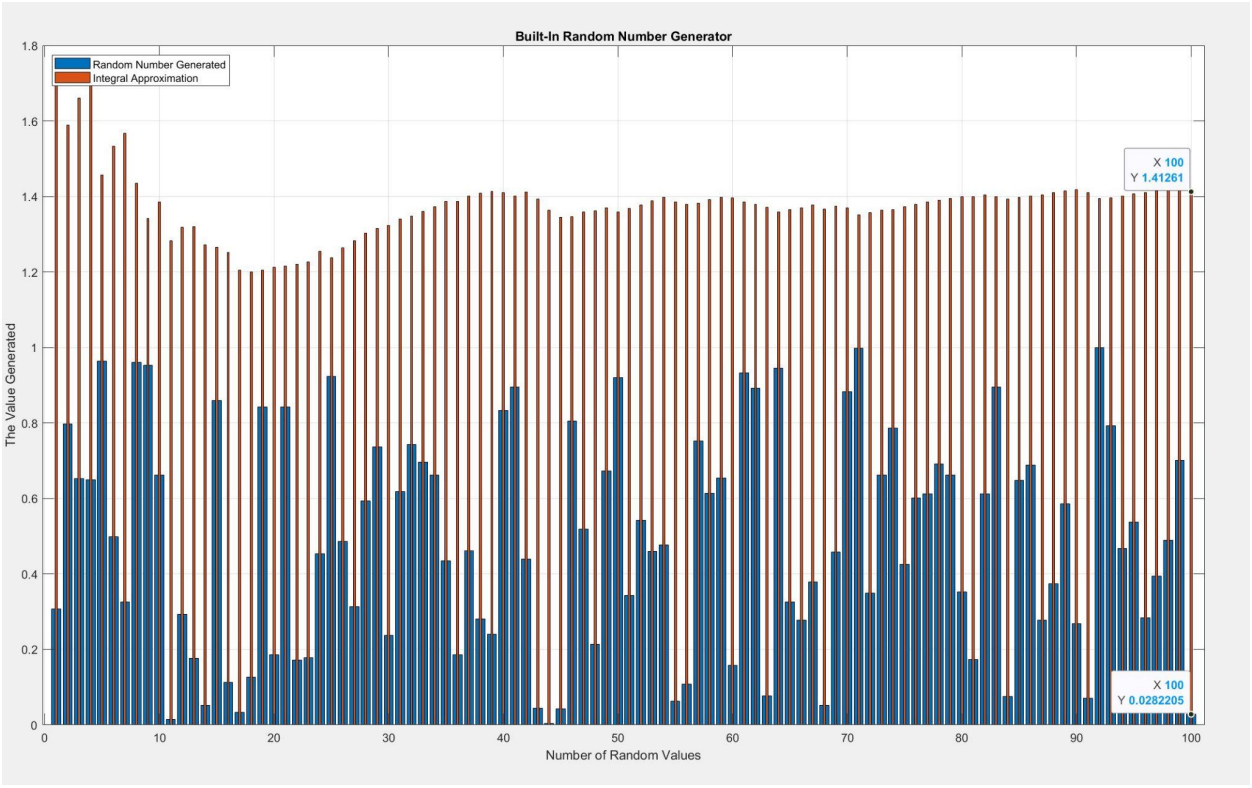
```
sum = 0;
    for i = 1:numOfRandValues
        x = vals(i);
        y = log(x^2) * log((1 - x)^2);
        sum = sum + y;
        integralApprox(i) = ((b - a) / i) * sum;
    end
```

(The analytic solution evaluates to 1.42026). My solution is accurate on smaller values of n as expected. In Figure 1a, it is the most accurate with a value of 1.40877, yet it continues to decline and gets less accurate to the analytical answer. In Figure 4a, it drops by 0.4085 points and in Figure 5a, it drops by 0.4457 points. Even though the solution to these values are correct (and more accurate with smaller size n), they significantly drop which is terrible for evaluating an approximation to the analytical answer.

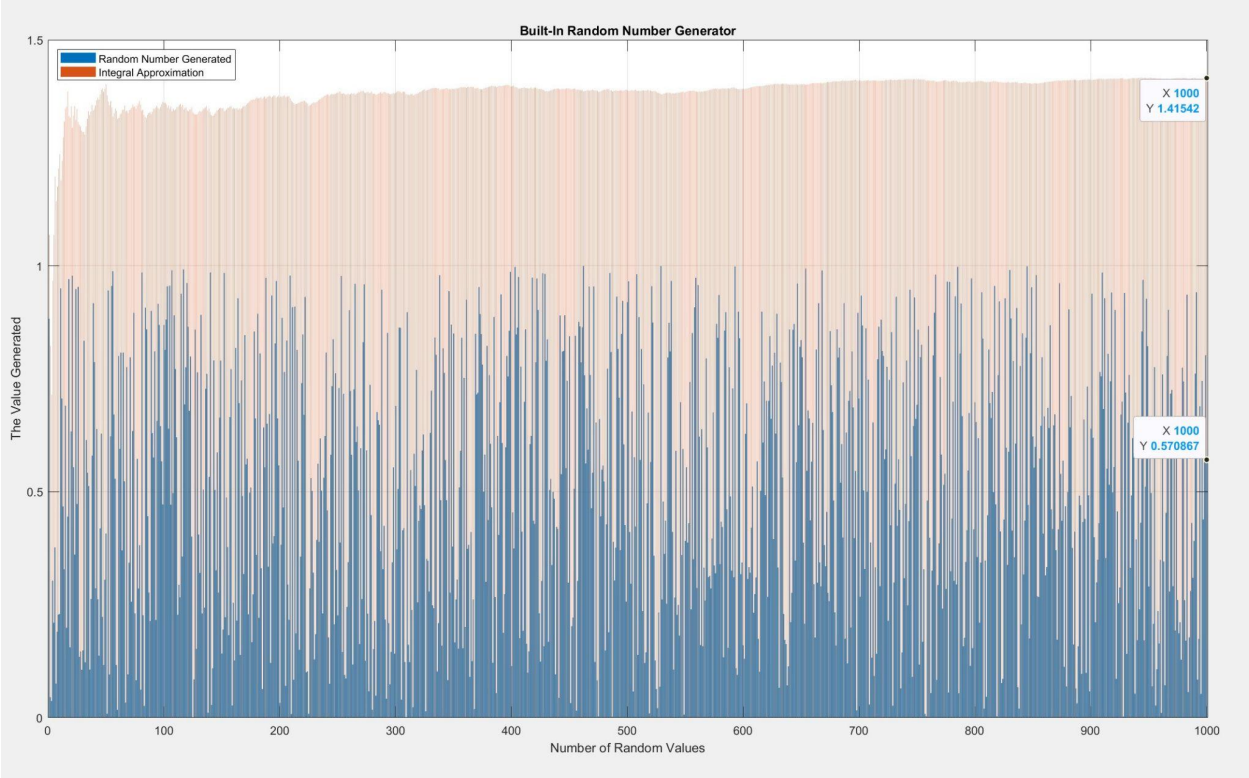
This again proves that making my own RNG is difficult and unnecessary because it won't calculate the Monte Carlo method effectively, especially on larger values of n.

c) Built-In RNG plots to calculate Monte Carlo Approximation:

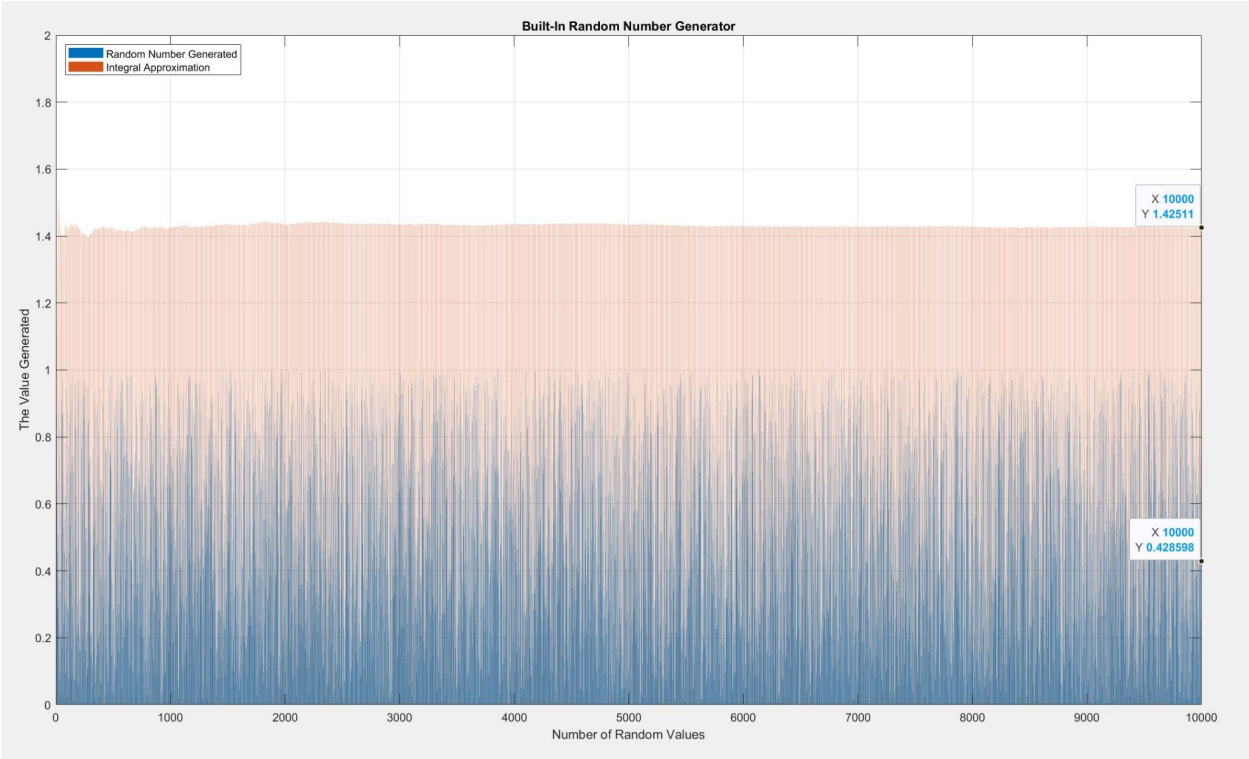
(Figure 1b) (n=100)



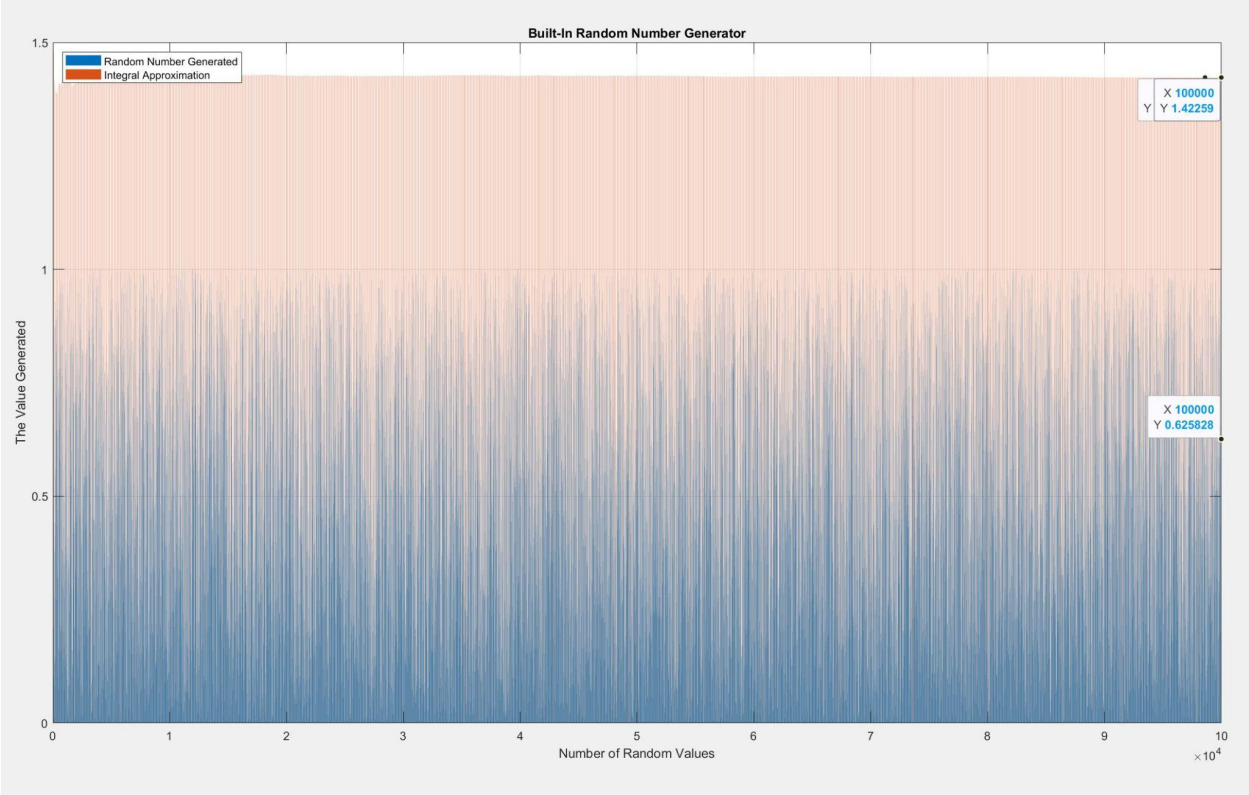
(Figure 2b) (n=1,000)



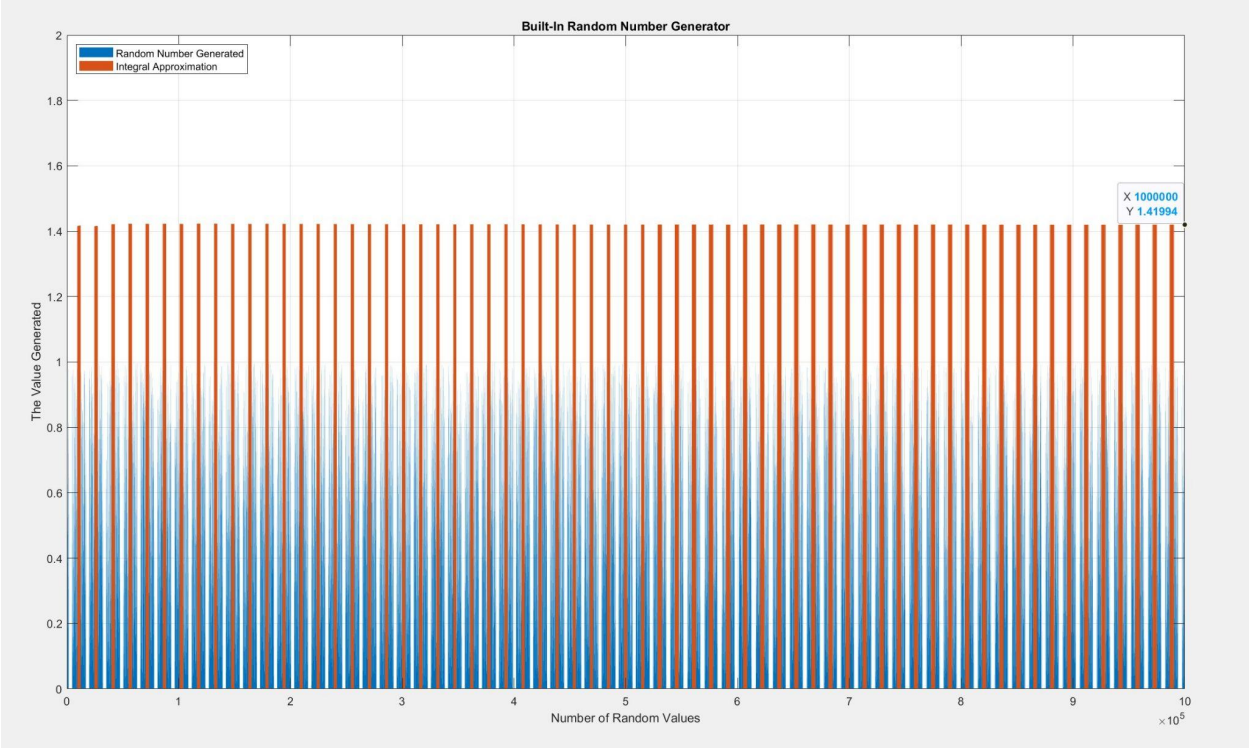
(Figure 3b) (n=10,000)



(Figure 4b) (n=100,000)



(Figure 5b) (n=1,000,000)



d) Built-In RNG's Solution vs. Analytic Solution (and comparison with my solution)

As you can see in Figures 1b - 5b, the solution is far more accurate to the analytical answer as it grows larger with n. It is also the most accurate in Figure 5b since the margin for error is 0.000324. In Figure 4b, the margin for error is 0.002326. This is to be expected because the Monte Carlo approach improves with more plot points as it gets closer to the analytical answer with larger n as proven in this experiment.

The differences between My Solution and the Built-In Solution is practically night and day. The Built-In RNG is far more accurate than mine. While I already knew this, I now understand for also getting closer to the Monte Carlo Approximation of the one dimensional integral, you need a good RNG formula to do so, otherwise the Monte Carlo approach will be pointless for larger n.

All in all, just use the RNG formulas built into most coding languages or use some already established theory on RNG creation using linear congruences, etc. Otherwise, it won't be possible to calculate large quantities of certain functions which means the margin for error will be far greater than need be.

Question 3

Solved on paper below

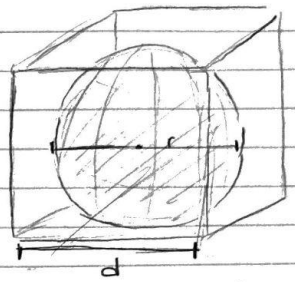
Question 3

Volume of a cube

$V = l \cdot w \cdot h$

Volume of a circle

$V = \frac{4}{3}\pi r^3$



- Because the sphere is within the cube, the length, width and height of the cube is the diameter of the sphere. Therefore, the diameter of the sphere is 2 · radius ($d = 2r$)
- Now, the volume of a cube is written as
  → $V = d^3 = (2r)^3$
- To start calculating π using Monte Carlo for a sphere inside a cube, we divide the volume of the sphere by the volume of the cube

$$\text{Volume Overlap} = \frac{\text{Volume of Sphere}}{\text{Volume of Cube}} = \frac{\frac{4}{3}\pi r^3}{(2r)^3} = \frac{\frac{4}{3}\pi r^3}{8r^3}$$

$$= \frac{\frac{4}{3}\pi}{8} = \frac{1}{6}\pi$$

- Lastly, to calculate π, we write ... $\frac{1}{6}\pi = \frac{\#\text{ of points in sphere}}{\#\text{ of points in cube}}$

such that ...

$$\pi = 6 \cdot \frac{\#\text{ of points in sphere}}{\#\text{ of points in cube}}$$

- With this, you can compare which points are within the sphere or outside the sphere by comparing if the point length (from center) is greater than or less than r.

Sources

This site was used to help find the decimal portion of a whole number. It is used to calculate my random number generator.

https://www.mathworks.com/matlabcentral/answers/21339-how-to-seperate-fractional-and-decimal-part-in-a-real-number

https://www.mathworks.com/help/matlab/ref/datenum.html

https://en.wikipedia.org/wiki/Monte_Carlo_method