Blake Van Dyken

Assignment 2

Methods Used:

- For my own random number generator, I used my own equation:
    - x_i = (2 * x_i + x_i) % (M-1)
    - x_i (or the seed) = 29089675
    - M = sys.maxsize - 1

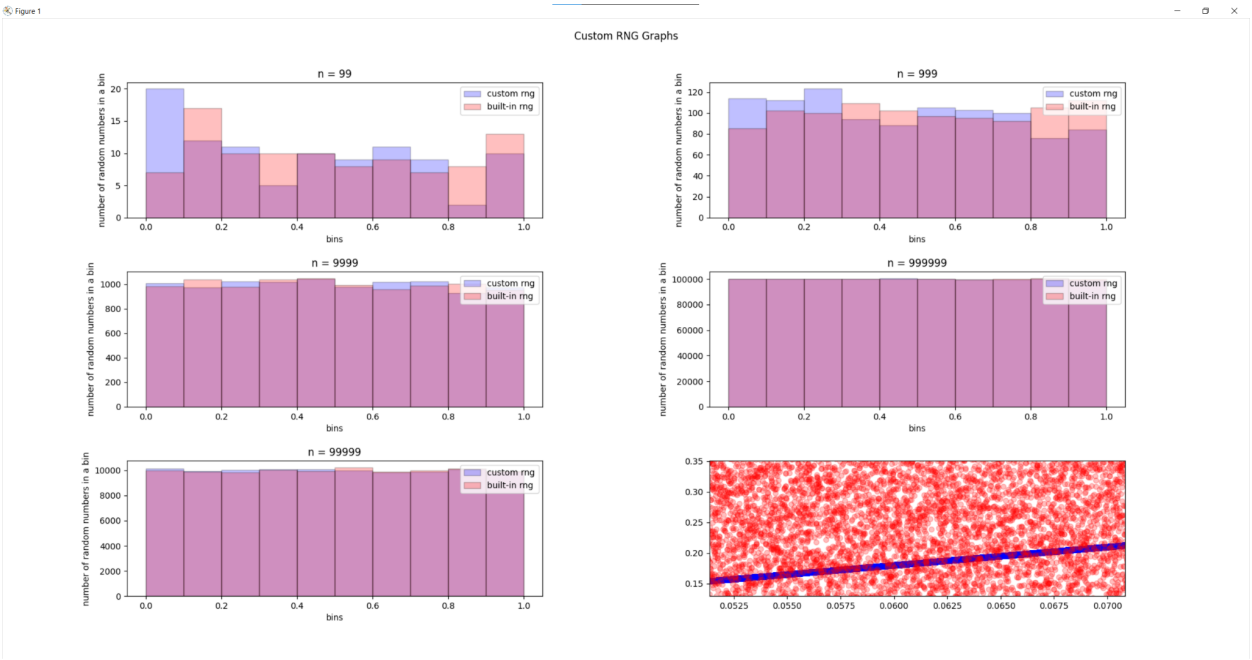
  Then I scaled the number to be between the interval [0, 1] by dividing the formula above by M - 1, thus the formula is: x_i / (M - 1).
- **Monte Carlo Method**
    - Used the range transformation equation $T(r_i) = a + \frac{b-a}{r_{max} - r_{min}} r_i =$

      Where r = the random number generated, a =0, b=1, rmax=1, rmin=0
    - Then took the function value, which was $\int_0^1 \ln(x^2)\ln((1-x)^2)\,dx$ without the integral
    - And proceeded to do an integral approximation using the equation

      $\frac{b-a}{n}\sum_{i=1}^{N} f(x_i)$, where the summation is the sum of all the previous results from f(x_i), where x_i was the value we got from the range transformation function. This gives us an approximation of the integral without using hard calculus
- **Volume of Cube and Sphere**
    - Used these equations to find necessary equations for calculating pi
    - The radius of the circle is equal to 2x the radius of the sphere

Results and Figures:

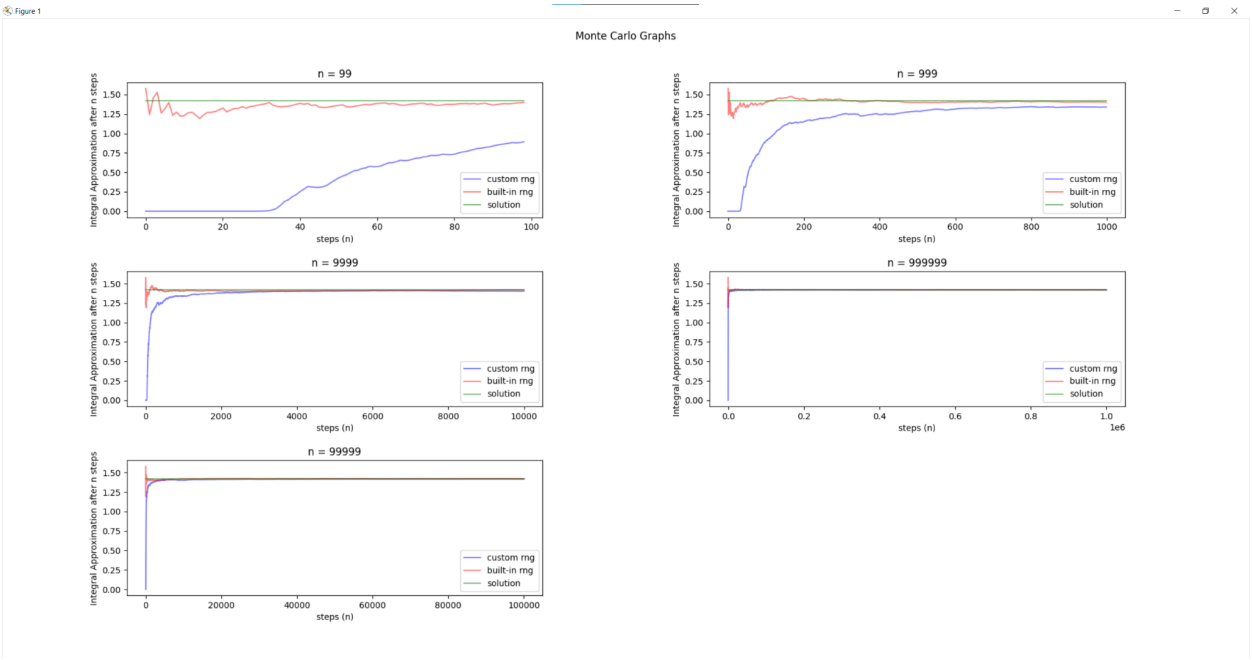
Question 1:

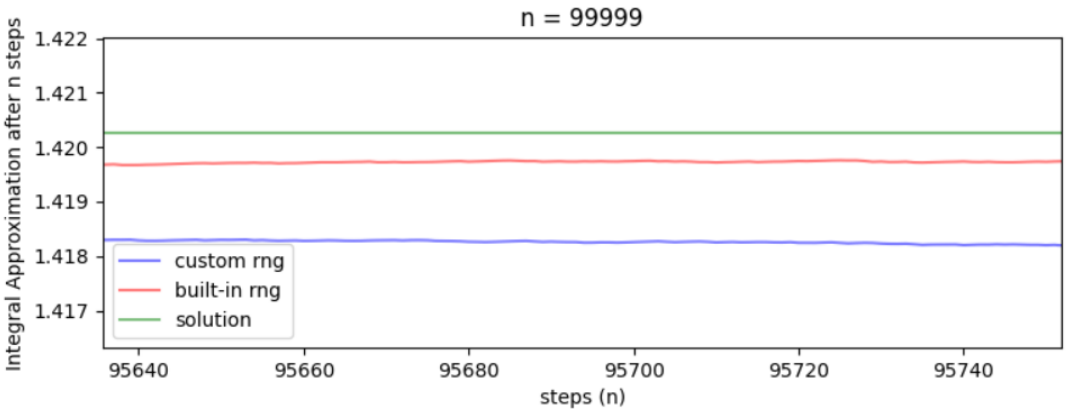


The way I generated my random number is described in the methods used section. Looking at the plots, the blue one is my custom rng method and the red is Python built-in random feature. Starting at n = 99, we can see that with so few numbers it is hard to cover an even distribution this is expected and we actually see that in the first graph as it is hard to tell which method is better. However, as we start to increase our n, we see we get more of an even distribution of numbers. In the next graph where n = 999, we see that the built-in rng does way better than mine, but as n increases, we get a better distribution and we see that they do about the same over a long period of time, however, the built-in one is more optimized and random and gets balanced results faster. We can see this in the scatter plot graph which I zoomed in on. I tried making various different equations but they all took a linear line like the one shown above. Thus we see my equation gives off more of a pattern which is terrible for randomness and

misses a very large portion of numbers on the interval I used. Thus we can see that it is very hard to come up with a random generator without using a linear congruential method.

Question 2:



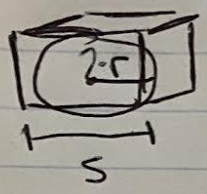


The analytical answer is $\text{Solution} = 8 - \frac{2\pi^2}{3}$, which is equivalent to roughly 1.42. My results using both my custom rng and pythons built-in one are very close to this answer after almost a million steps, then I get to around 1.42 as well. Using the Monte Carlo method as described on the first page, I graphed the result using my rng method in blue, and the built-in python rng method, and there is also a green line that represents

the solution. Looking at the first graph when n is very small (n = 99), we can see using this method over a short interval my rng method performs terribly, and the built in method is already close to the answer. As we increase N, we see that the built-in rng gets closer to the actual result faster or in fewer iterations of n. Also, over time as n gets very large as seen in the second picture, which is zoomed in on the graph of n = 999999, we see that the built-in rng gets closer to the actual solution whereas my custom rng method stays further away by a few more very small decimal places. Overall, after a certain amount of steps we see the graphs do not change very much but give us accurate results in a faster way and my custom rng method takes a very long and unreasonable amount of time to get close to the answer compared to the random method in python.

Question 3:

Blake #3



Vol. of Sphere $= \frac{4}{3}\pi r^3$ r = radius

Vol. of Cube $= (2r)^3$

$$\frac{\frac{4}{3}\pi r^3}{(2r)^3} = \frac{\frac{4}{3}\pi r^3}{8r^3} = \frac{\frac{4}{3}\pi}{8} = \frac{\pi}{6}$$

$$\frac{\pi}{6} = \frac{\#\text{ darts in circle}}{\#\text{ darts in square}}$$

$$\pi = \frac{6 \cdot (\#\text{ darts in circle})}{(\#\text{ darts in square})}$$

Sources:

All my code is submitted with the project and on GitHub here:

https://github.com/iPupkin/Theory-3200