SHARIF UNIVERSITY OF TECHNOLOGY
PHYSICS DEPARTMENT

Problem Set 6

Student name: *Ali Fayyaz - 98100967*

Course: *Computational Physcis - (Spring 2023)*
Due date: *April 7, 2023*

Exercise 6.1

Call the random generator of your programming language to generate digits 0 to 9. Put it in a loop of length N.

(*a*) Graph a histogram. If the generator is unbiased, you would expect to see each of the digits about $\frac{N}{10}$ times.

(*b*) Show that the deviation from the above number obeys the relation $\frac{\sigma}{N} \sim \frac{1}{\sqrt{N}}$

(*c*) Do you notice any similarities between this exercise and random ballistic deposition?

Answer. To answer this exercise, two methods were implemented. The first was exactly as described in the question. For different values of N, the histogram was plotted using *numpy.pyplot.bar()*. The second method was a more efficient, *Direct to Histogram* Method which passed a large *numpy* array of random values to *numpy.pyplot.bar()*.

As N grew in both the methods, $\frac{N}{10}$ became more apparent.

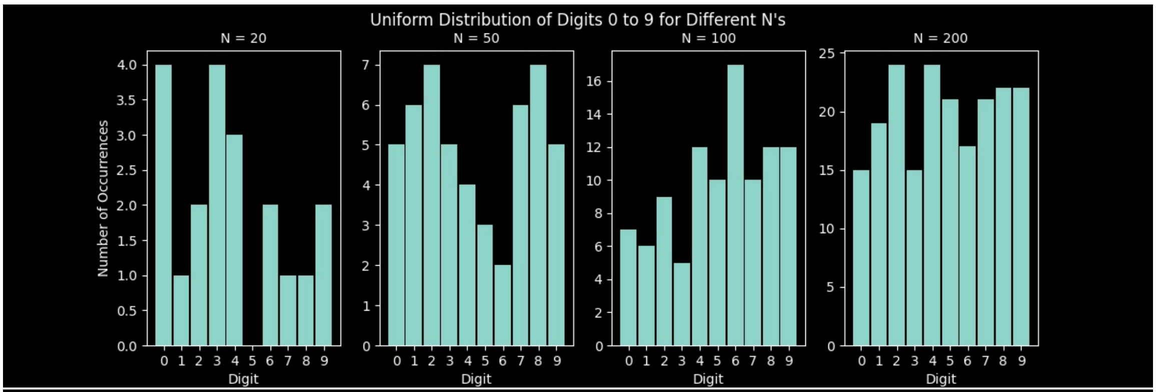
To test the relation, the second method was used for different ensemble sizes. Notice that

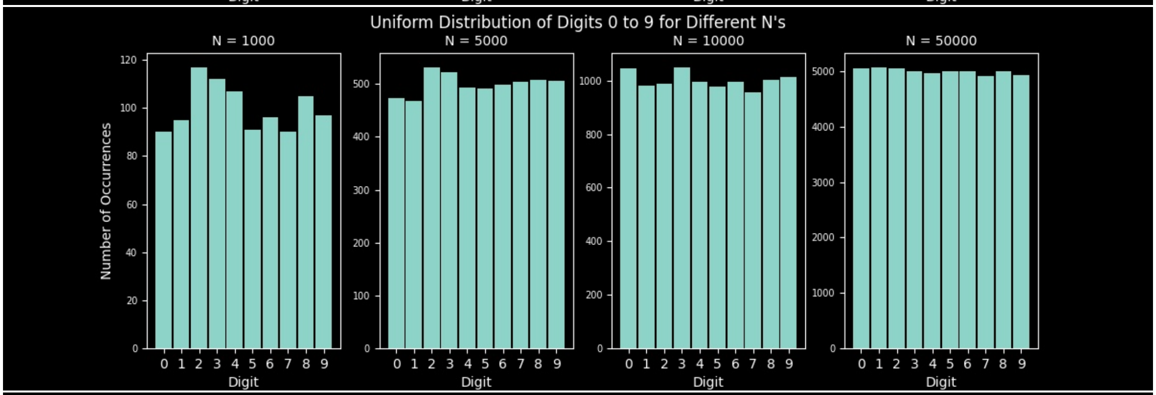
$$\sigma \sim \sqrt{N} \quad \Rightarrow \quad \log_{10}(\sigma) \sim \frac{1}{2}\log_{10}(N)$$

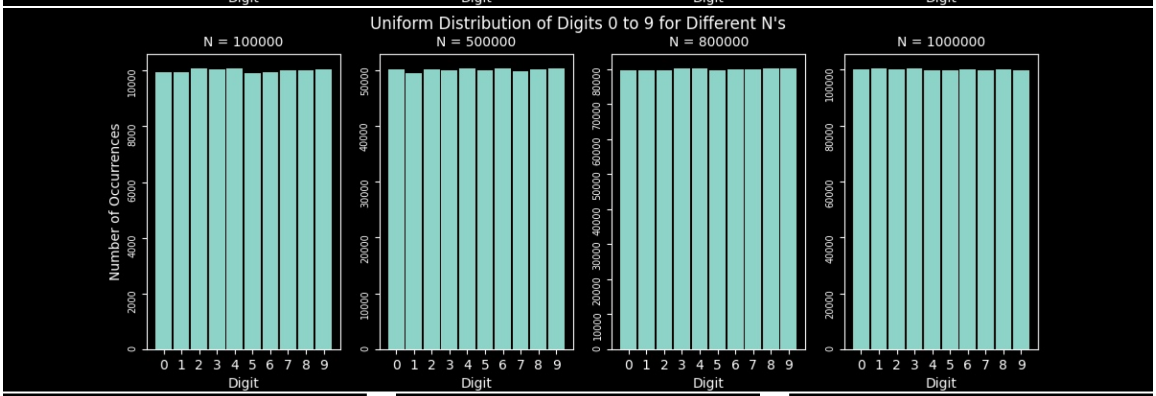
therefore a log-log graph should demonstrate a $\frac{1}{2}$ slope.

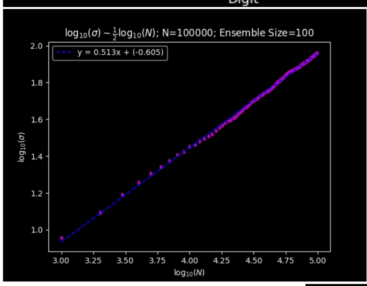
This exercise is basically random ballistic deposition with 10 bins. The surface roughness in random ballistic deposition grew in an identical fashion:

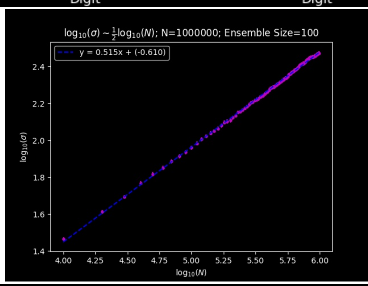
$$w(t) \sim \sqrt{t} \quad \equiv \quad \sigma \sim \sqrt{N}$$

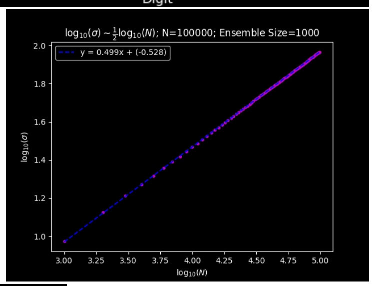

Uniform Distribution of Digits 0 to 9 for Different N's
N = 20
N = 50
N = 100
N = 200
Number of Occurrences
Digit

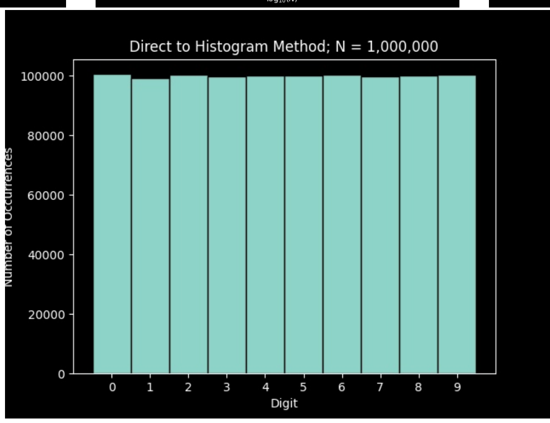

Uniform Distribution of Digits 0 to 9 for Different N's
N = 1000
N = 5000
N = 10000
N = 50000
Number of Occurrences
Digit


Uniform Distribution of Digits 0 to 9 for Different N's
N = 100000
N = 500000
N = 800000
N = 1000000
Number of Occurrences
Digit


$\log_{10}(\sigma) \sim \frac{1}{2}\log_{10}(N)$; N=100000; Ensemble Size=100
y = 0.513x + (-0.605)
$\log_{10}(\sigma)$
$\log_{10}(N)$


$\log_{10}(\sigma) \sim \frac{1}{2}\log_{10}(N)$; N=1000000; Ensemble Size=100
y = 0.515x + (-0.610)
$\log_{10}(\sigma)$
$\log_{10}(N)$

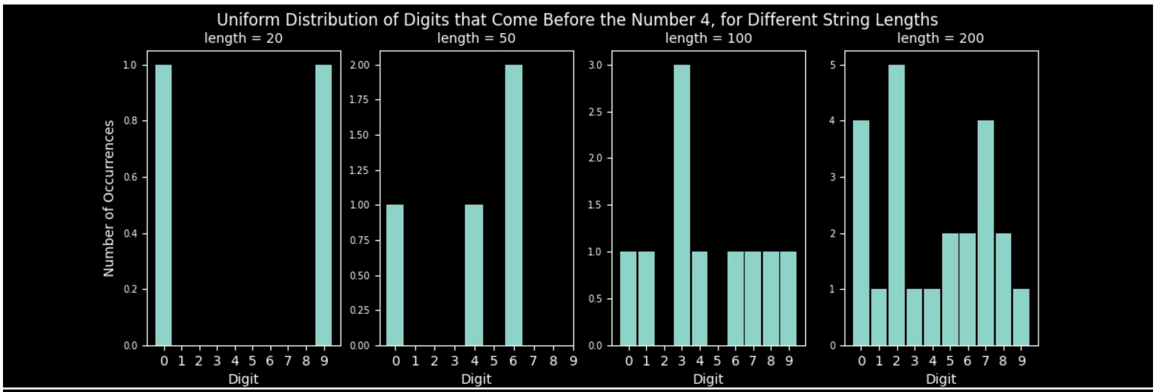

$\log_{10}(\sigma) \sim \frac{1}{2}\log_{10}(N)$; N=100000; Ensemble Size=1000
y = 0.499x + (-0.528)
$\log_{10}(\sigma)$
$\log_{10}(N)$

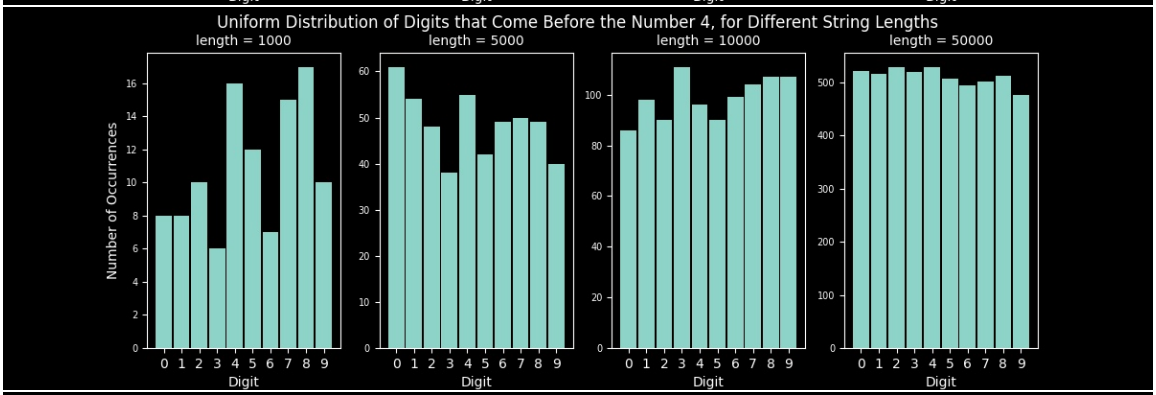

Direct to Histogram Method; N = 1,000,000
Number of Occurrences
Digit

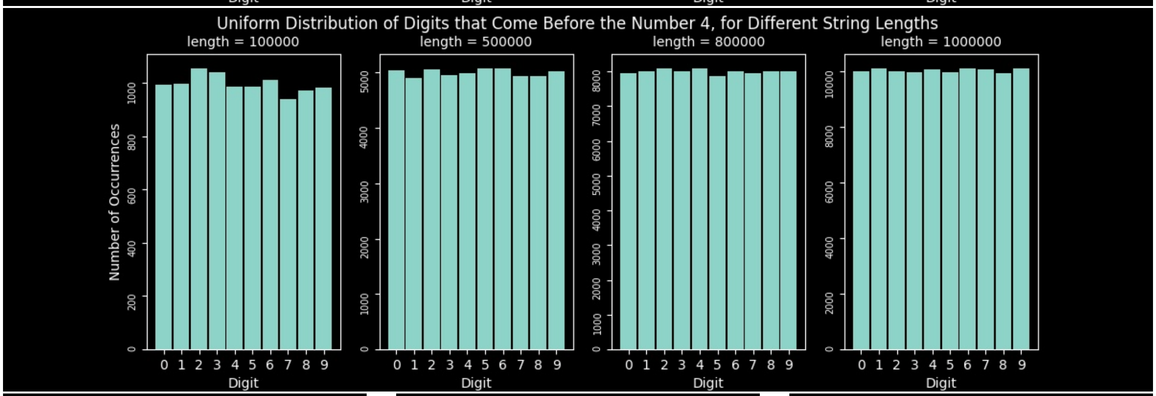
Exercise 6.2

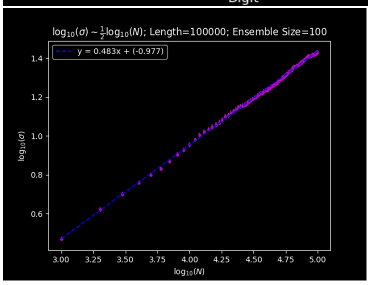
Repeat the previous exercise but this time use only the digits that appear before the number 4. Is this distribution uniform?

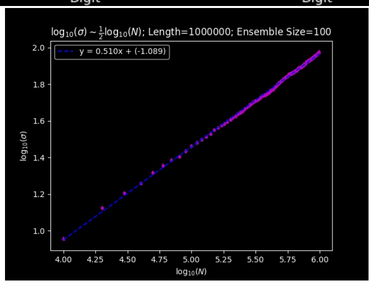
Answer. A large array of numbers was created using *numpy.random.randint()*. Afterwards, *numpy.unique()* and *numpy.where()* were used to count the digits that came before the number 4 in the said large array. The results were as expected ($\frac{N}{100}$ for each number), similar to the previous exercise. The relation $\sigma \sim \sqrt{N}$ also held, as evident in the log-log plots.

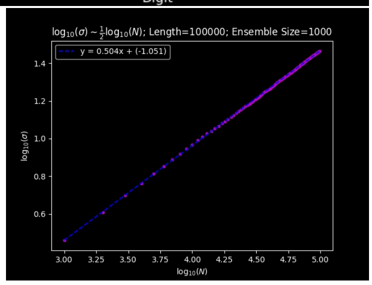








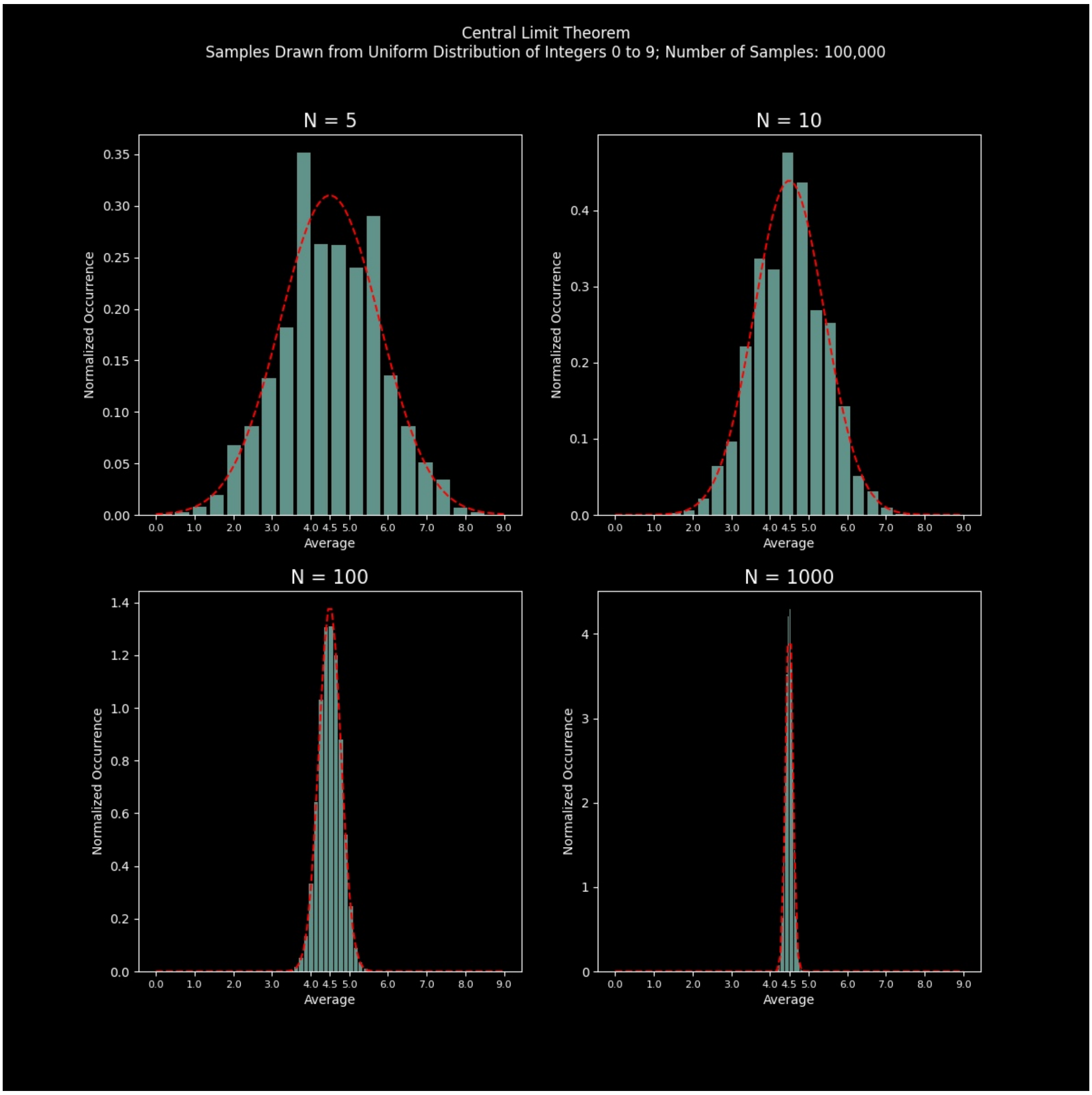




Exercise 6.3

Use the random generator of your programming language and test the Central Limit Theorem for $N \in \{5, 10, 100, 1000\}$. Do you notice any similarities among this exercise and random deposition and random deposition with relaxation?

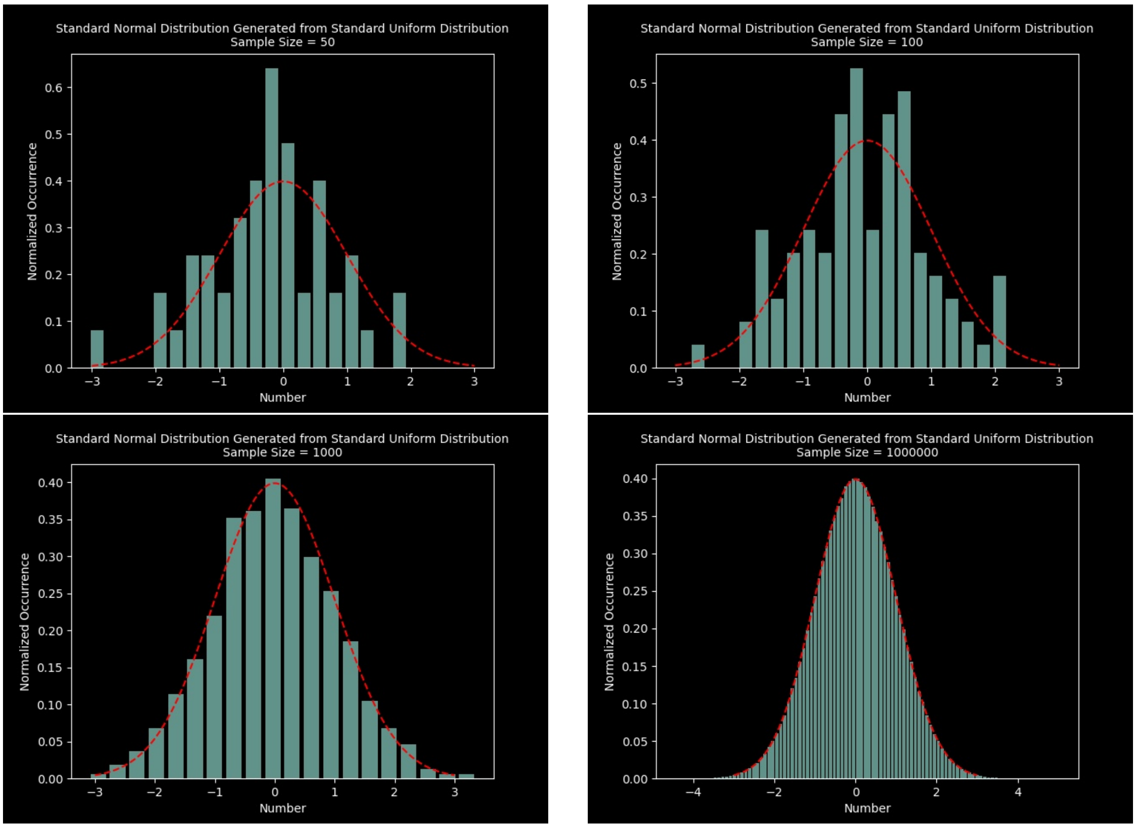
Answer. For each N, an ensemble of size 100,000 was created and the mean was calculated.*numpy* methods were used for efficiency. The plots show the normalized histogram of the resulting data and the corresponding Gaussian Distribution as a dashed lined. Notice that in these *normalized* histograms the area of the bins sum to 1, and not the heights.



Exercise 6.4

Use the method described in the textbook to create a random generator of a Gaussian Distribution. Plot the generated data to show that it works properly.

Answer. The random generator *numpy.random.rand()* uses a Standard Uniform Distribution. The approach explained in the textbook was followed for generating two sets of random numbers from Standard Uniform Distribution. The variance of the Gaussian Distribution was arbitrary, so it was set to be 1: the variance of the Standard Normal Distribution. The resulting histograms, accompanied by the corresponding Gaussain Distributions (the dashed lines) are as follows for different sample sizes:



Exercise 7.1

Using Simple Sampling Monte Carlo, and Intelligent Sampling Monte Carlo methods, calculate the integral $I = \int_0^2 e^{-x^2} dx$ and compare the resluts. For the Intelligent Sampling use $g(x) = e^{-x}$. Put the answer to the integral, statistical error, real error (checked with the likes of *MATLAB* or *Mathematica*), and the execution time for each method, and for different number of samples.

Answer. The two functions **montecarlo_ss()** and **montecarlo_is()** were defined for the Simple Sampling and the Intelligent Sampling methods, respectively. The parameters are similar: the range of the integral a to b, and the number of iterations n. An array of x values containing n numbers is then created. For the Simple Sampling method, the values come from a uniform distribution in the range (a, b), whereas for the Intelligent Sampling method, as described in the textbook, the values come from the converted distribution of *g(x)*. *f(x)* and *g(x)* were defined in place, instead of seperately, for efficiency. The real answer for this specific integral was taken from Wolfram Alpha to be:

$$I \simeq 0.882081$$

pandas was used to generate the tables. The results are also saved to csv files.

As expected, the Intelligent Sampling method generally takes longer to execute and has smaller errors. The table on the left corresponds to Simple Sampling and the one on the right correponds to Intelligent Sampling.

n	INTEGRAL ANSWER	STATISTICAL ERROR	REAL ERROR (checked with wolfram alpha)	EXECUTION TIME (s)
100	0.875365	0.034186	-0.006716	0.000020
200	0.883775	0.024262	0.001694	0.000030
500	0.881478	0.015389	-0.000603	0.000010
800	0.879015	0.012160	-0.003066	0.000040
1000	0.882791	0.010901	0.000710	0.000030
10000	0.881731	0.003445	-0.000350	0.000121
100000	0.882088	0.001090	0.000006	0.001069
1000000	0.882026	0.000345	-0.000056	0.020180
10000000	0.882115	0.000109	0.000034	0.207601

n	INTEGRAL ANSWER	STATISTICAL ERROR	REAL ERROR (checked with wolfram alpha)	EXECUTION TIME (s)
100	0.887419	0.032468	0.005338	0.000020
200	0.882636	0.023299	0.000555	0.000030
500	0.884130	0.014723	0.002049	0.000045
800	0.881170	0.011698	-0.000911	0.000035
1000	0.883520	0.010423	0.001439	0.000055
10000	0.882035	0.003305	-0.000046	0.000200
100000	0.881942	0.001045	-0.000139	0.002465
1000000	0.882077	0.000330	-0.000004	0.035727
10000000	0.882087	0.000105	0.000006	0.364646

Exercise 7.2

The mass density of a sphere decreases linearly from top to bottom. If the density is ρ_0 at the top and $\frac{1}{2}\rho_0$ at the bottom, find its center of mass.

Answer. Suppose the center of the sphere is on the center of coordinates. Cylindrical coordinates are used. With simple mathematics, one can find the density ρ as a function of height z (R is the radius):

$$\rho(z) = \frac{1}{4}\frac{\rho_0}{R}z + \frac{3}{4}\rho$$

thus the total mass is:

$$M = \pi R^3 \rho_0$$

Given the two integrals below,

$$I = \int_V \rho(z) z dV$$

$$M = \int_V \rho(z) dV$$

Following the convention of the previous exercise, Intelligent Sampling Monte Carlo can be used with the function $\rho(z)$ as the distribution function $g(z)$ as follows:

$$I = \int_{z=-R}^{z=R} f(z)\rho(z)dz \qquad ; \qquad f(z) = z\pi(R^2 - z^2)$$

$$M = \int_{z=-R}^{z=R} h(z)\rho(z)dz \qquad ; \qquad h(z) = \pi(R^2 - z^2)$$

The center of mass R_{cm} can now be found:

$$R_{cm} = \frac{I}{M}$$

Analytically, $R_{cm} = \frac{1}{15}R \simeq 0.067R$. The integrals were calculated similar to the previous exercise and the results were put in the following table (for $R = 1$ and $\rho_0 = 1$):

n	INTEGRAL ANSWER	STATISTICAL ERROR
100	-0.068	0.926
200	-0.068	0.926
500	-0.068	0.926
800	-0.068	0.926
1000	-0.068	0.926
10000	-0.068	0.926
100000	-0.068	0.926
1000000	-0.068	0.926
10000000	-0.069	0.926