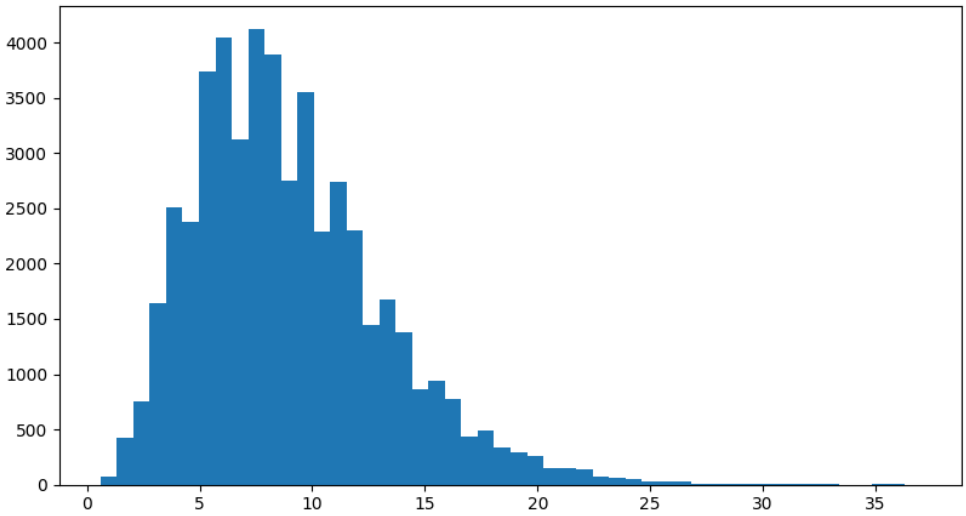
All of the code in the exercise runs using python3 with numpy, matplotlib and scipy as dependencies.

1

Code is at chi.py

Collecting the statistics over 50,000 runs of M=10, N=100 we get:
`Mean: 8.972368000000001, Median: 8.399999999999999`

Indeed the mean is M - 1 as stated by theory, which means python's random.random method is a good RNG.
The histogram received from it is:



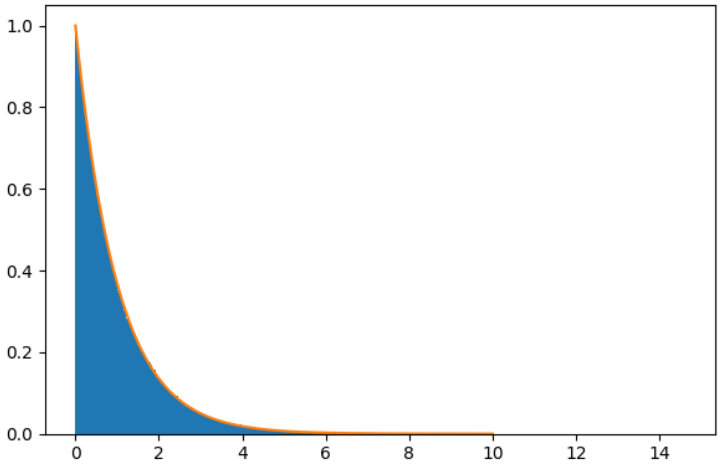
This does display some kind of normal distribution, as a chi^2 variable should display.

2

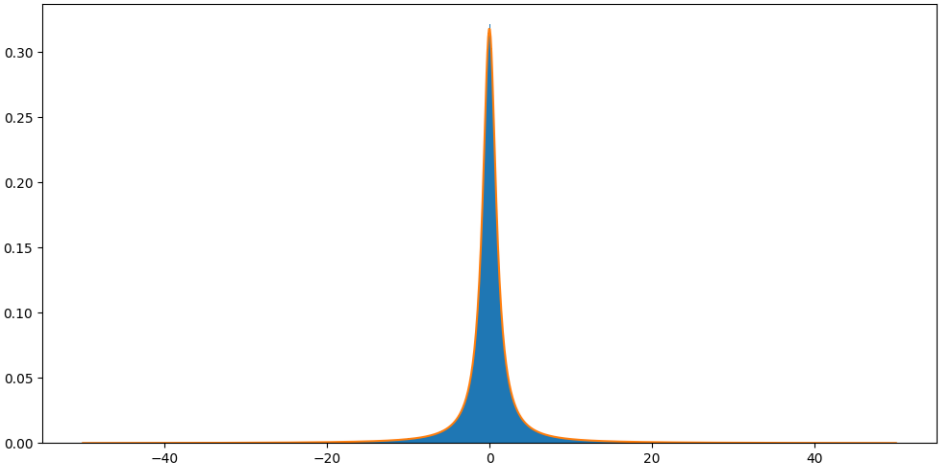
Code for both plots is at inverse.py
It shows the exp defaultly, uncommenting the Lorentz section to display its plot.

For ease of explanation, I include plots.

Exp with gamma=1
The code generates 10**6 numbers, then displays the histogram using plt.hist (in blue) with 1000 bins and density=True to “normalize” the histogram.
Then on top of that I plot the exp pdf function in orange:



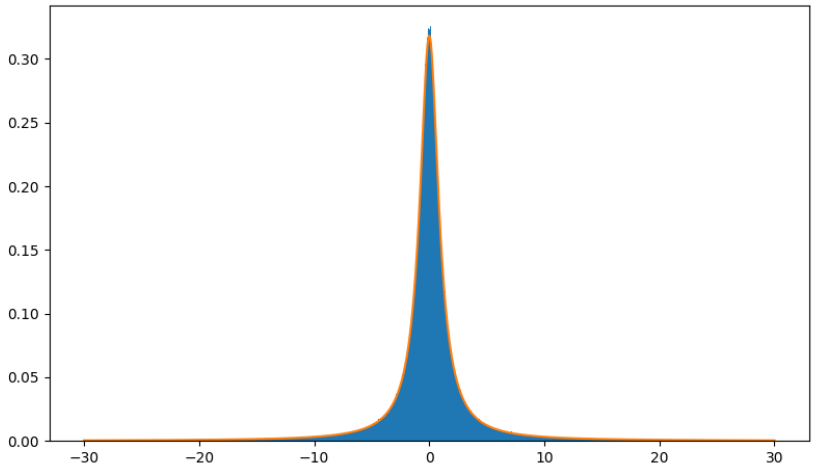
Now for the Lorentz distribution, I do the same as with the expo. The function does give some values which are very big (900 for example). I reject all that are not within (-50, 50)



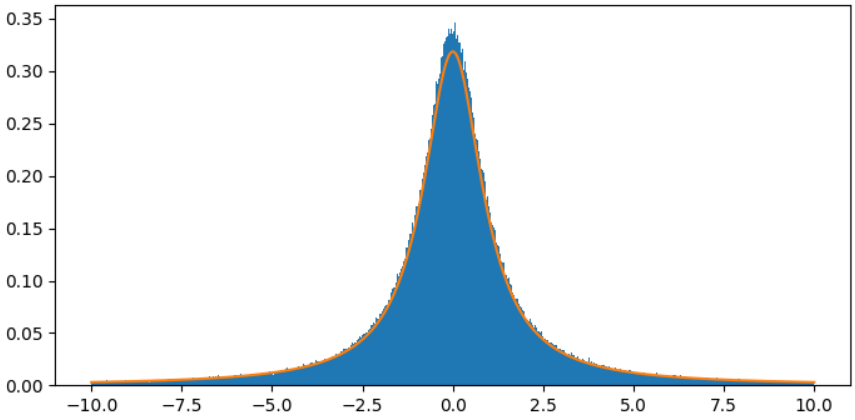
3

Code at quotient.py , running it displays a plot of the distribution and pdf on top.

Using the Box-Muller method I generate 10^6 numbers. Plot a distribution into 1000 bins and then a Lorentz function on top in orange.
The distribution is normalized by including the density=True flag when using the hist() method.



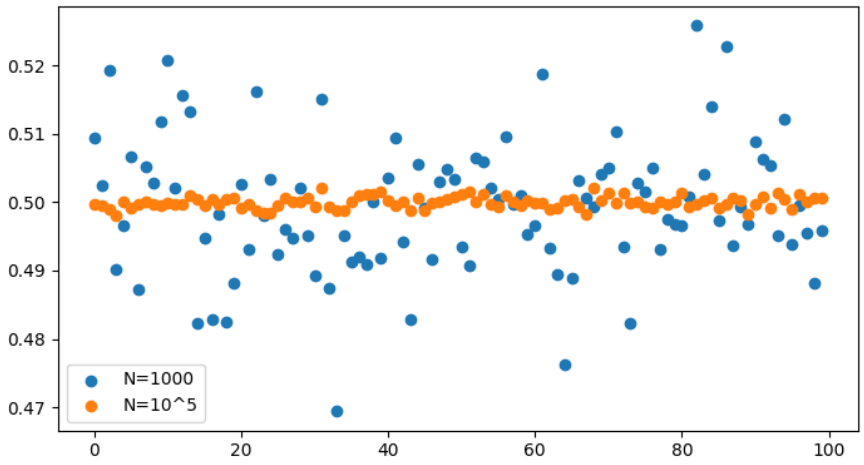
Showing that the result is correct, now I include the same but limiting to [-10, 10] as the question requires. This makes the histogram to not-fit a bit the expected function because of the density=True normalization



4

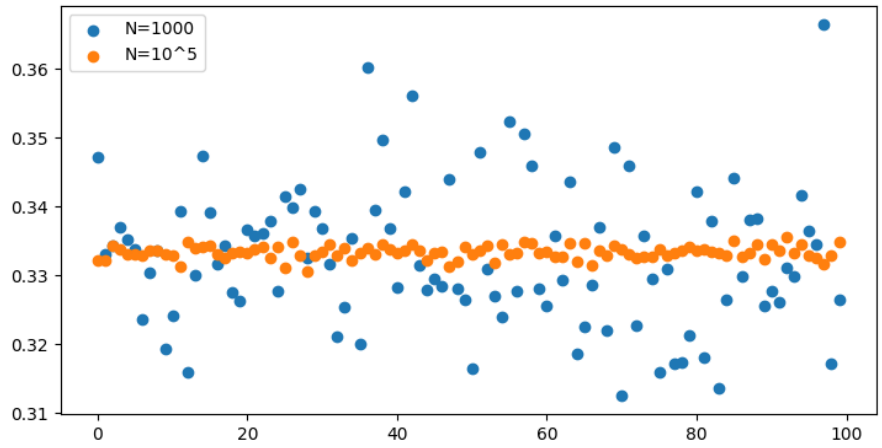
Code is at arithmetic_mean.py

Running a uniform [0, 1] rng, I plot the scatter for the arithmetic mean for N=1000 and N=10^5



Indeed we can see that the mean of the bigger sample size is closed to the expected value 1 / k + 1 = 1 2 for k = 1.
While for N=1000 we get a more scattered pattern.

Now for x^2 we get the following scatter plot:

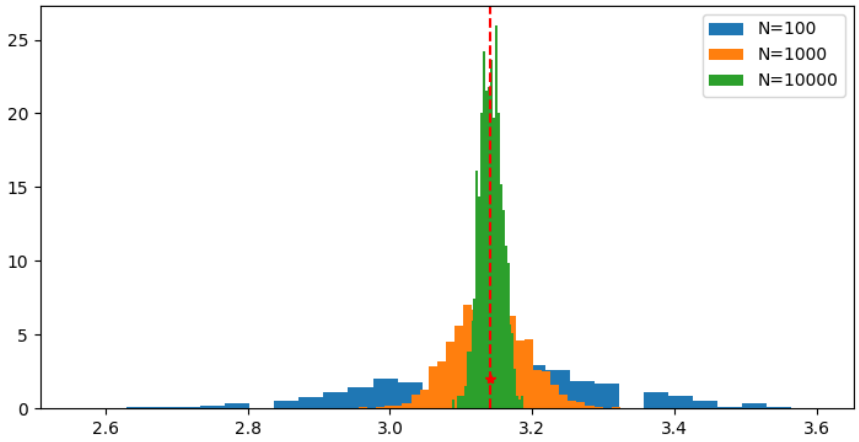


Which again fits 1 / k + 1 = 1/3 for k = 2

5

Code is at hitmiss.py

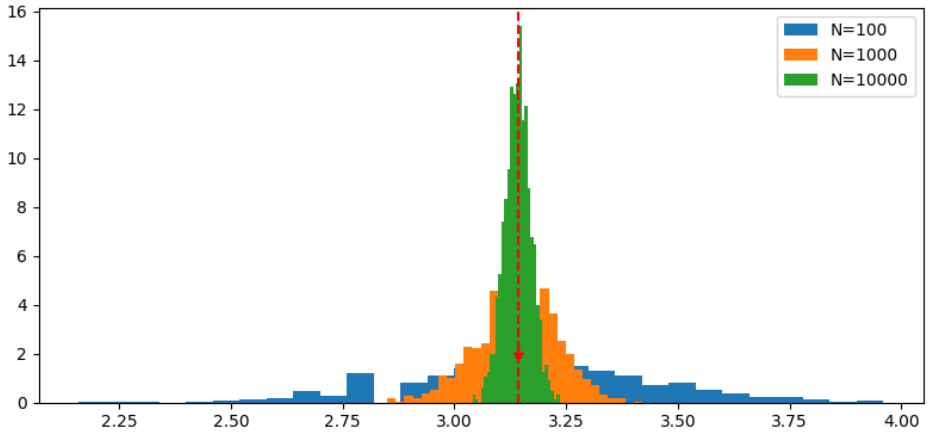
mypi(N) returns the value of pi based on the hit and miss method and pi_hist() plots a histogram of N=10^2, 10^3, 10^4.
The red vertical line shows the real value of pi.



The mean of those is 3.1528, 3.145036, 3.1416276

We see that N=100 is not enough to get reliable results (blue) while already and N=1000 we get dense enough samples to determine pi.

Now doing the same for a 3-sphere with the *mypi_sphere(N)* function we get the following plot:



The mean of those accordingly is: 3.15144, 3.14433, 3.1415628

Again the results don’t vary much between the 2 and 3 dimensions version, maybe the N=100 is more scattered in the 3 dimension version but still the mean is similar.
Now for higher dimensions,The volume grows much faster than the radius and the calculations become much heavier and less accurate.

Actually at the Monte Carlo course we did a comparison of those higher dimensions for a n-sphere volume calculation. For which at the 7th dimension it showed that the std-deviation of the volume is already 0.3, giving an unreliable estimate of the volume.