Monte Carlo - Exercise 2

1

Run script using python (3) at qcg.py

Using :
m = 1067089 (1033 * 1033)
a = 309900 (1033 * 4 * 5 * 15)
b = 3100 (b = 1 mod p_i)
c = 463 (no common factors with m)

The repeat interval it prints is: 1067089 which is equal to m.

To find the repeat I save all the random numbers, and then search for a sequence of at least 10 repeating numbers. To test it use the find_repeat function.

2

$w = 31 \quad p = 13 \quad q = 2 \qquad a_{const} = 6B5ECCF6$

$= \ldots 1100, 1111, 0110$
$\qquad\qquad\quad C \quad F \quad 6$

$a_0 = 1001 = 11, 1110, 1001$
$a_2 = 1021 = 11, 1111, 1101$

then by a twisted GSRF scheme we get

$a_0 = a_2 \oplus (a_0 >> 1) \oplus a_{const}$ the LSB of a_0 is 1

$$\begin{array}{l} a_0 = 1\,1\,1\,1\,1\,1\,1\,1\,0\,1 \oplus \\ \qquad\; 0\,1\,1\,1\,1\,1\,0\,1\,0\,0 \oplus \\ \ldots 1\,1\,0\,0\,1\,1\,1\,1\,0\,1\,1\,0 \oplus \end{array}$$

$$= 11, 10, 11, 11, 11, 11$$
$$\qquad E \quad\; F \quad\; F$$

then going back to Hexa, and our last XOR being with a 32 bit number a_{const}.

$a_0 = 6B5ECEFF$ after one iteration.

Hope I understood the algorithm correctly.

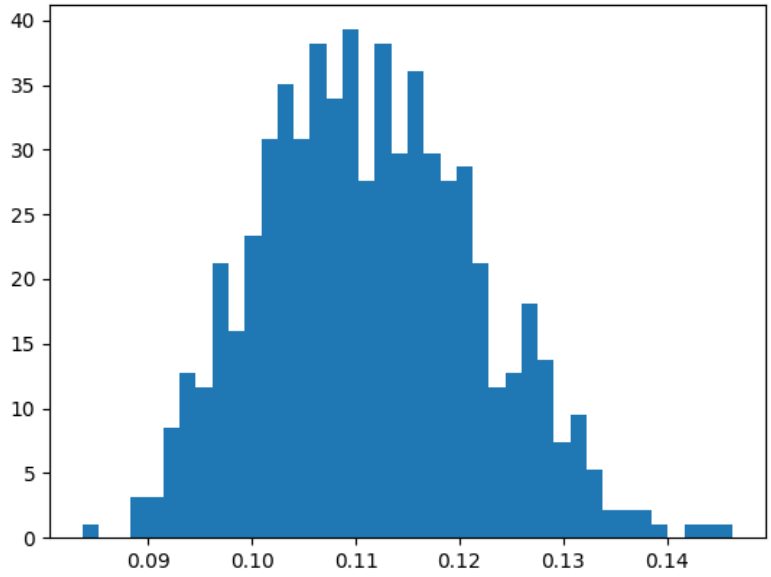
3

Code is at chi_test.py, and can run using python (3)
print_chi_mean_median_distribution() gets the mean, median and upper lower percentiles while also plotting the distribution of chi squared.
We calculate 600 Chi with 10^6 points each, and then plot them using an .hist() func with 40 bins.
Ideally we should get a normal distribution of the chis and they should fluctuate around the 100 - (⅔) point.

From the table at the following site: (https://www.medcalc.org/manual/chi-square-table.php)
For 100 bins, we should have 118.498, 124.342, 67.328 for the 0.10, 0.05, 0.95 percentiles respectively.

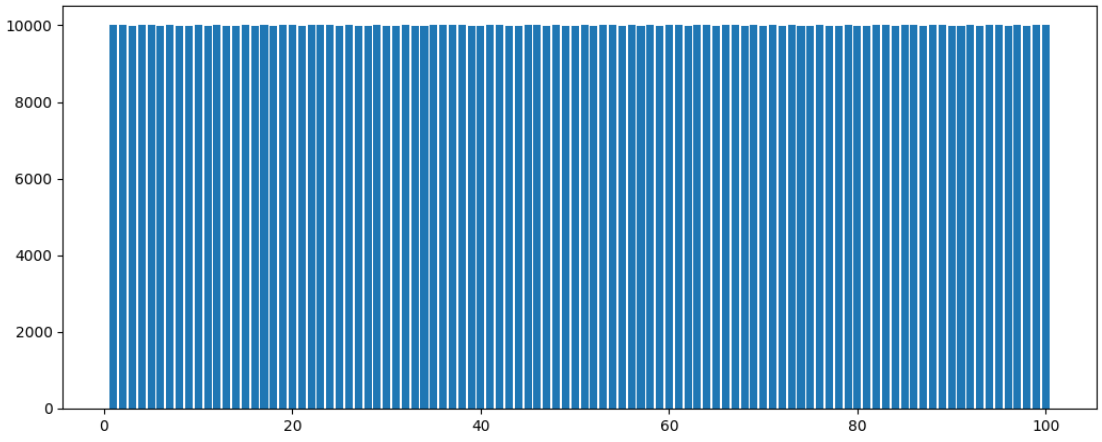
LCG:

Mean: 0.11124466666666667, Median: 0.11069999999999999
alpha=0.05 lower confidence: 0.09459000000000001, upper confidence: 0.12945
alpha=0.1, lower confidence: 0.09777999999999999, upper confidence: 0.1266,



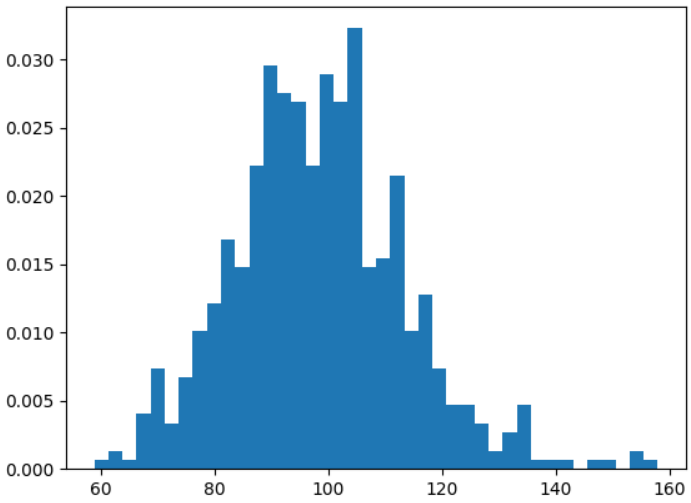
Seems that the chi^2 of the LCG RNG is very bad, I did fix it since the last exercise and got the needed repeat interval + pattern when we plot it.
It does have what seems like an almost normal distribution around the median point of 0.11 .
Looking at the distribution into bins of our random numbers when calculating one chi we get a

very flat one - which is not good.

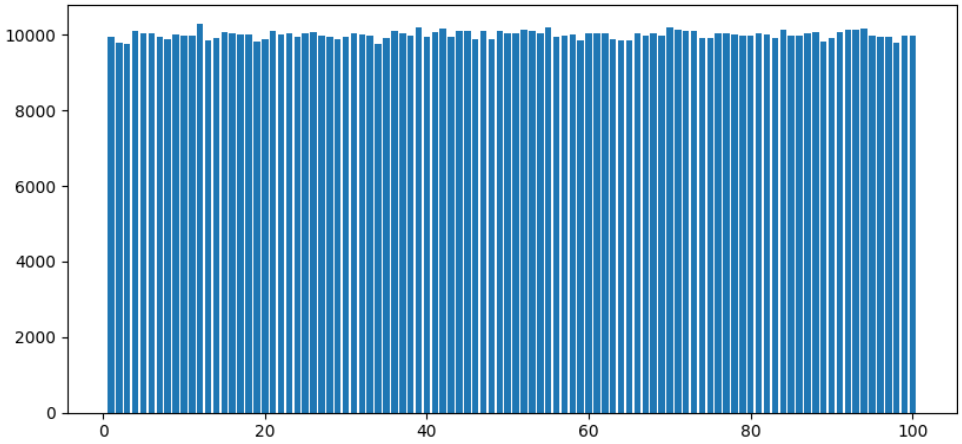


PM:

Mean: 98.48098333333333, Median: 98.04589999999999,
alpha=0.05 lower confidence: 74.73173999999999, upper confidence: 124.17316999999998
alpha=0.1, lower confidence: 80.31530000000001, upper confidence: 116.81353999999997,



Looks a bit good, the distribution is normal like, around the 100 (M) point.
The median is almost as needed: (99 - ⅔)
Looking at the percentiles, it is close to what it should be: 118.498 and 124.342

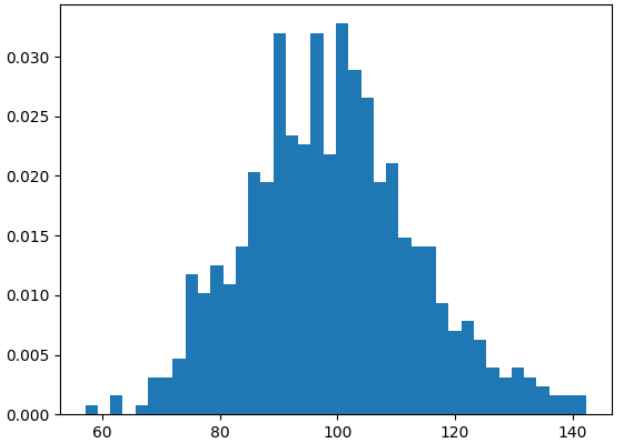


Looking at the bin distribution of one chi calculation, it looks nicer than the LCG, it fluctuates more.

Twister (random.random())

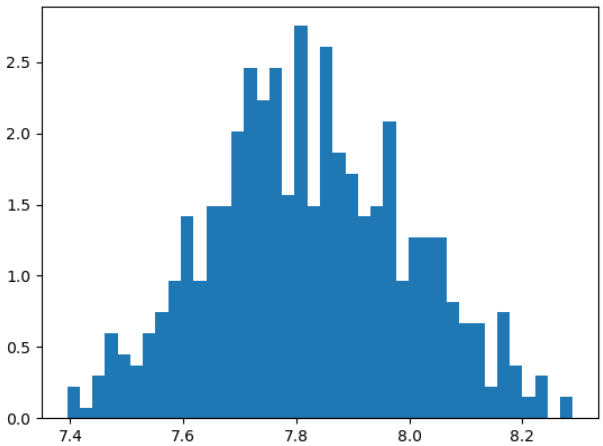
Mean: 99.05515166666667, Median: 98.5275
alpha=0.05 lower confidence: 75.64328000000002, upper confidence: 124.03106999999994,
alpha=0.1, lower confidence: 80.27634, upper confidence: 117.69074

The twister has very good values, the median is close to (99 - ⅔) and the percentiles are good. The percentiles are very close to the values given at the table, besides the lower 0.05.



QCG

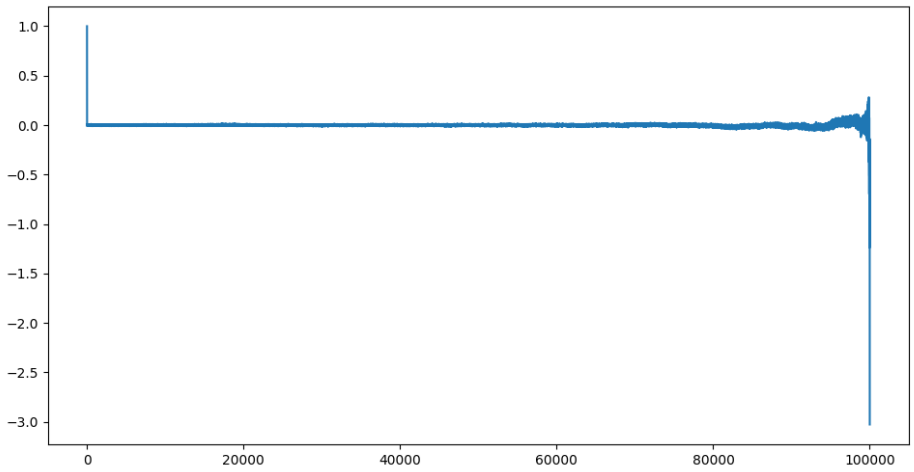
Mean: 7.822667, Median: 7.8108,
alpha=0.05 lower confidence: 7.54256, upper confidence: 8.124430000000002
alpha=0.1, lower confidence: 7.598279999999999, upper confidence: 8.0619



Bad results again, it is not close to being around the 100 mark, yet it is a bit better than the LCG. It does have what looks to be a normal distribution around the median.

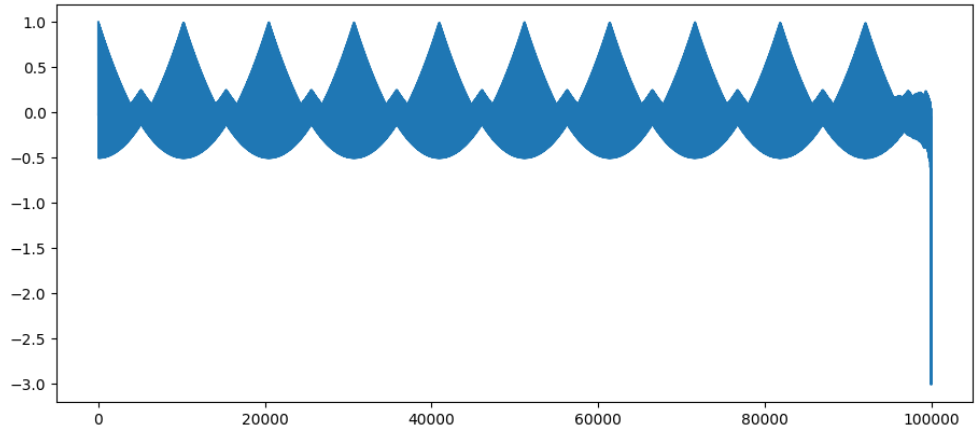
Autocorrelation is tested at the autocorrelation.py file, and can run using python (3).
Using 10^5 points, we plot the result of the LCG and PM random number generators.

PM



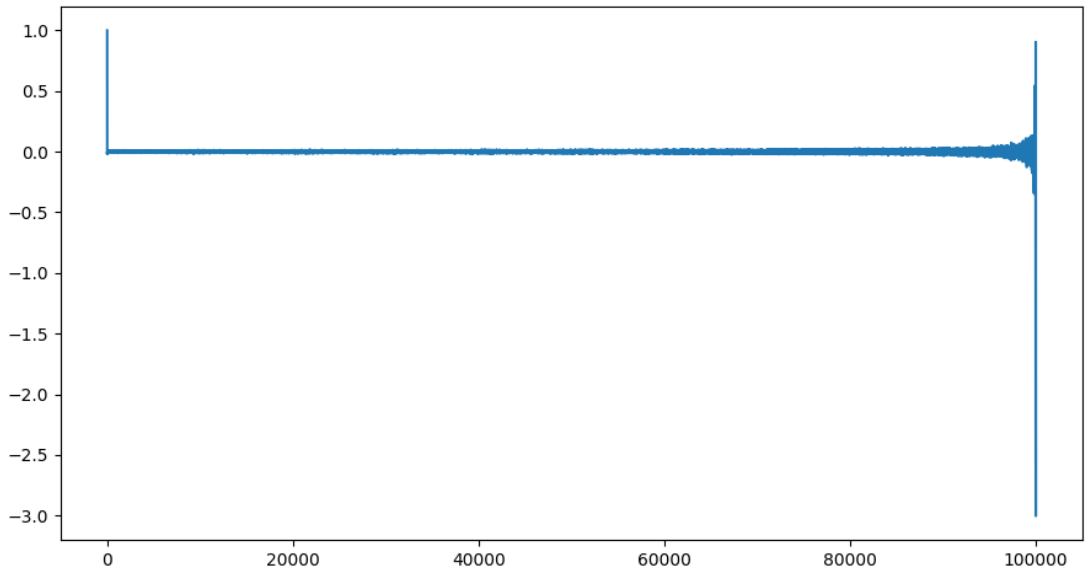
The autocorrelation of the PM scheme looks nice, it fluctuates around the 0 point and diverges only closer to the end where it has less points to compare to.

LCG



The autocorrelation of the LCG indeed looks bad, it has a definitive pattern to it which means it fails the test.

Trying to improve the LCG, I implemented a shuffle at lcg_shuffle.py where I create an array of 32 values, getting one randomly and replacing it by a new one.
Calculating the autocorrelation of 10^5 points we get



We get a much better result than the regular LCG, now it has no visible patterns and it even seems to fluctuate nicer than the PM.