Robb Hill
Professor Gordon
CS4050
3 May 2020

Analysis of Pseudo-Random Number Generators

INTRODUCTION

Pseudo random number generators are just that, pseudo random. Unless a generator is based on true unaltered chaos, the numbers cannot be truly random. They are bound by some level of sensor input and mathematical equation.

Python Random

The Python library random has a number generator based on the Mersenne Twister method. This method uses the Mersenne prime, which is a prime number one less than a power of two. This is the most common random generator in python.

My pseudo random number generator

Image noise is a by-product of unwanted change in signal. This is most commonly observed in digital images, though noise in the form of grain can be observed in film images at the negative and, more so, at the print level. In film the grain becomes more apparent the further the image is enlarged. However, in digital images common types of noise are Gaussian, Salt-and-pepper and Uniform. The types of noise typically result in a relatively even spread of the actual noise and the noise is usually uniform in its value distribution. Because of this it is a relatively fair representation of random. All images have the possibility to be all noise, be partially noise, or be noise free. This means noise can be represented on a 0 to 1 scale of infinitely growing decimals by its calculation.

My goal will be to create a program that uses an existing random number generator library to generate an index to represent a word in a dictionary. That word will then be used as a key term in a google image search. The first result yielded from the random word will be downloaded. Once downloaded the image will be run through a function that calculates the noise in the image by calculating the average sigma of the distribution and variation. This will be represented as a 64bit-floatingpoint number. This number will then be used in part with a user provided range of numbers to generate a final random value and will be printed to screen.

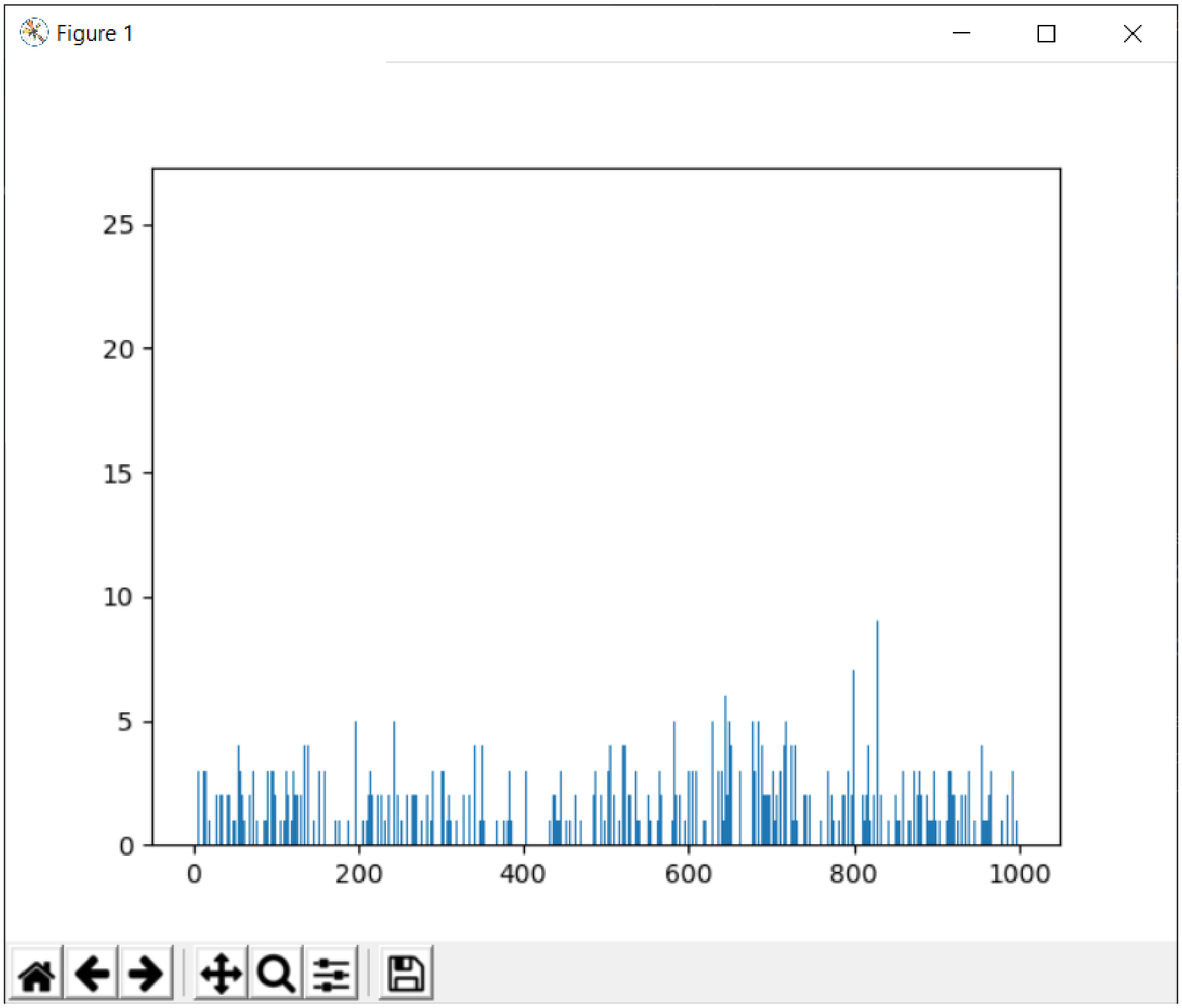
METHOD

Diving into this problem required first determining how to get the images required off google. After days of research and attempting this it was determined that an HTML scraper was needed. Again, more research was done into doing this and many open source projects were found, all of whom were unsuccessful in scraping images from google searches except one. I looked the source code, and everything seemed in order until I started checking the open issues on the repository. The most recent issue was that google purposefully changed their HTML output to thwart all efforts to scrape its images. After all this time spent, I abandoned the idea of using a google search to generate the image required as input.

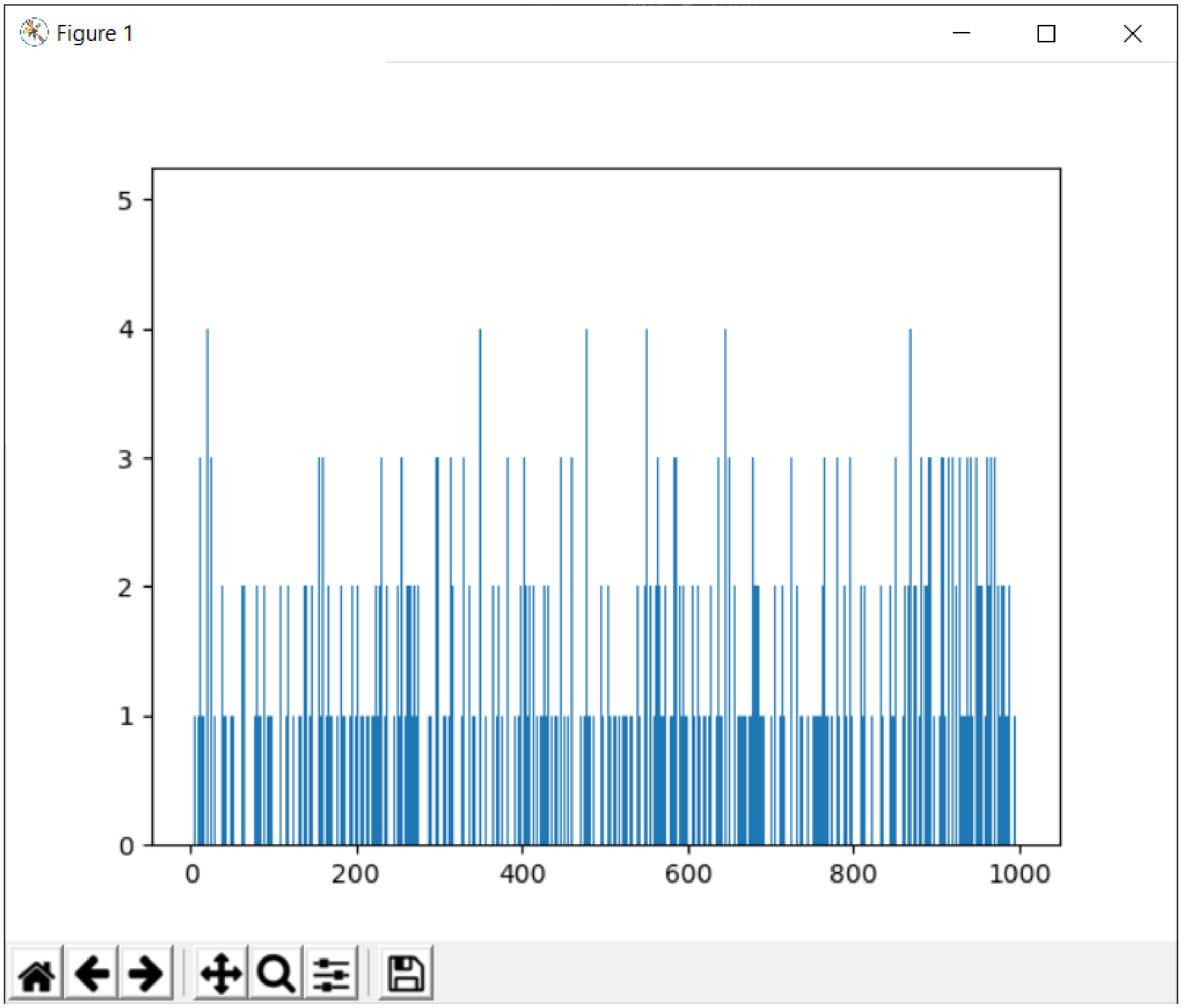
I moved my idea to now utilize a random image selector online. This proved simple as the site specifically showed how to pull its images to several places, including a simple download.

Using the OpenCV library to analyze the image I was able to determine the noise of an image in the hope of using this to generate a random number. This proved successful but I wished to take it one step further. Using the average value of the image’s pixel data, I generated a multiplier to be used in the calculation of a random number. This number then modulus with the desired range of the seed then returns a pseudo random number.

RESULTS



My Random Generator of 1000 #'s between 0 and 1000



Py random number generator 1000 #'s between 0 and 1000

DISCUSSION

As can be seen from the results, despite the graphics having different scaling, the randomly generated numbers visually appear very similar in density and spread. This surprised me as I know the photo generator has a specific number of photos in it and the likelihood of getting the same photo twice is as high as the number of tries to the number of photos. The spread is certainly more uniform with the python method and could be concluded is more random than my solution

If the original method could have been applied my number generator would have been quite unique and reasonably random. It still would not have likely been as good as pythons but it would be very close to reaching the goal of basing the generator on true chaos, as we all know the internet is.