Physical Random Number Generators

Eric Hofesmann
College of Charleston Department of Physics and Astronomy
(Dated: May 2, 2016)

Methods for creating true random numbers include the emission of radioactive particles, audio noise, and photocell noise. The most basic random number generator creates only ones and zeros. Radiation was collected with a Geiger counter and an even or odd value represented a zero or one. The audio and photocell noise were both attached to an amplifier and the voltage was read off of the oscilloscope data. The voltage being even or odd represented a zero or a one. The randomness of the numbers was tested by the percentage of time that a specific bit string appears in each sample. The data was also unbiased through the Von Neumann approach. The biased data showed that the photocell and 50 ms speaker noise are more random than the radiation and 5 ms speaker data. The unbiased data showed an improvement in all samples except radiation and the control.

I. INTRODUCTION

Random number generators are important for various computer applications like simulation and encryption [1]. Simulation must be based off of random number generators so that the results are representative of the physical system that is being simulated. The generator must be random because if is biased, then the simulation will also be biased. Other simulations like the lottery are crucial that they are random so that the numbers cannot be guessed. Random numbers are also used at the heart of encryption. When information is encrypted, random numbers are used to create keys to the information, and if the numbers are not truly random then hackers be able to replicate the key and obtain access to personal information [2].

Even though random number generators are widely used in software, computers have a difficult time actually creating random numbers. Computers are designed to be deterministic machines which implies that there is no aspect of randomness to them that can be exploited. There are two types of random number generators, PRNGS which are built around mathematical methods that take a non random number and turn it into a number that is mostly random [3]. TRNGS are true random number generators which are based off of random physical properties which are converted into numbers. One of the leading random number generators Random.org uses atmospheric noise to create random numbers. The numbers generated in this experiment are binary ones and zeros, but they can be converted into any length integer given enough bits.

A. Voltage Amplifier

Noise that is used for random number generators is often emitted at a low voltage. This is generally too low for most oscilloscopes to measure, thus it is necessary to amplify this voltage. The amplifier used during this experiment uses an op amp and two resistors [4]. Figure 1 shows a circuit diagram of the amplifier. There is a voltage divider between the output of the op amp, the inverting input, and ground. Thus the ratio of the resistors represents the gain of the amplifier. In this case there is a gain of 1000.

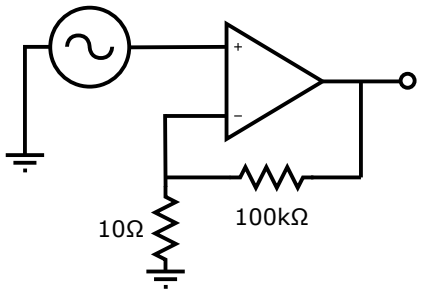

FIG. 1. A voltage amplifier can be built out of an op amp and two resistors. One resistor connects the output of the op amp to the inverting input of the op amp and is connected to a second resistor which is connected to ground. The resistance of the first resistor divided by the resistance of the second gives the gain of the amplifier. The input to the amplifier is a signal sent into the non-inverting input of the op amp and the amplified signal is taken from the output of the op amp.

II. RADIATION

Radiation is often one of the first examples of a true random number generator because it is inherently random due to the underlying quantum principles governing it [5]. The timing of the emission of a radioactive particle is non deterministic which will be the source of entropy for this random number generator.

A sample of strontium 90 was used as the radioactive source in this experiment. Strontium 90 undergoes β^- decay which releases an electron from it's nucleus and produces yttrium 90. A Geiger counter was used to count the number of particles it emitted every six seconds. The number of particles that were detected was between 70 to 100 every six seconds. These numbers are high enough to provide a random distribution of even and odd numbers. The probability of it being even should be the same as it

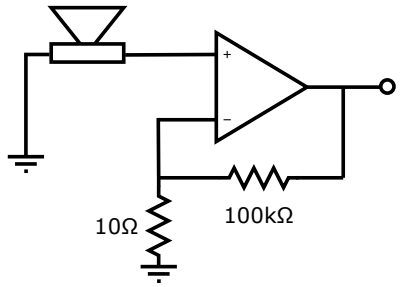

FIG. 2. A speaker generates a voltage which leads into the non-inverting input of an op amp. The op amp was used as a voltage amplifier with a gain of 1000.

being odd so a zero or one was recorded respectively. This method takes a long time, however so only 250 numbers were recorded.

III. AUDIO NOISE

Noise is a common method of producing random numbers. Generally, thermal noise is used for random number generators in computer systems [2] but audio noise is more straightforward to produce in a lab. A speaker was used to function as a microphone as it takes in sound and emits an electrical signal. A low quality speaker is desired since it would generate a noisy signal. The signal had a low amplitude on the oscilloscope so it had to be amplified using the circuit seen in Figure 2.

The digital oscilloscope used was able to take an image of the waveform and save the voltages in a spreadsheet. Each image of the voltage waveform was broken down into 4000 integer values. Similarly to the radiation, a zero is for an even value of the voltage and a one is for an odd value of the voltage. There were fifteen images taken totaling 60,000 random bits of one and zero.

Two timescales were tested, 50 ms and 5 ms. Initially 5 ms data was recorded since at this frequency, the image was refreshed quicker than at 50 ms and thus should result in a more random waveform. However, the 5 ms data had a smoother waveform than the 50 ms data which implies less variation between the integer values and thus less randomness. Both samples of 60,000 bits were taken to compare the results.

IV. PHOTOCELL NOISE

Light can also be used to generate noise through the variations in photon levels on a detector. The type of detector that varies it voltage with changing levels in light is called a photocell. Since the number of photons hitting the detector is never the same, there will be a random aspect in the signal. Figure 3 shows the circuit used in the photocell experiment. This circuit is was similar to the speaker except that instead of the microphone generating

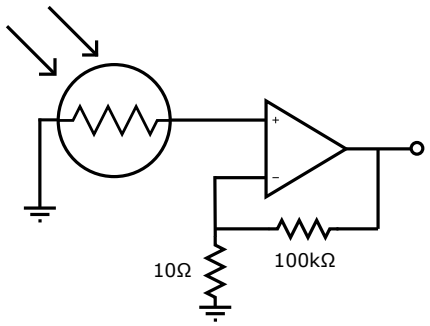

FIG. 3. This is the circuit through which the photocell noise was generated. The photocell creates a voltage which leads into the op amp. The op amp was used as a voltage amplifier with a gain of 1000.

a signal, a photocell would detect the incoming light and fluctuate the voltage. This signal amplitude was 100 mV and thus had an approximate 10 times lower amplitude than the speakers signal.

The random number data was obtained through the same method as the speaker noise. The digital oscilloscope was set to a 50 ms timescale and then the data was extracted directly from the waveforms seen on the screen. The even and odd voltages represented zeros and ones in the data set. This was repeated for fifteen waveforms and they were compiled into a set of 60,000 random bits.

V. RANDOMNESS TESTING

In order to determine the quality of the random number generators, the randomness of the numbers must be quantitatively tested in a way that can be compared between the various methods. A control group was used for these tests. This control group was a set of 60,000 ones and zeros generated by Random.org [3].

NIST has a test suite designed to determine the randomness of a random number generator [6]. There are 12 tests in this suite each increasing in complexity. The methods implemented in this experiment were two of the simpler ones since the random number generators must first pass the simple tests before being tested with the complex ones. The two methods that were used consisted of finding the frequency of having a one and a zero and determining the number of times that a sequence of bits repeats. This was generalized in the program that was written, see Appendix B, to compare all possible combinations of bit strings for a certain number of bits. If one bit is used, then this will determine the frequency of finding a one or a zero but it can be extended to any number of bits. The program would generate every possible combination of ones and zeros for a certain number of bits and then count the number of times that combination appeared in the samples data. This number of times that each combination appeared was divided by the to-

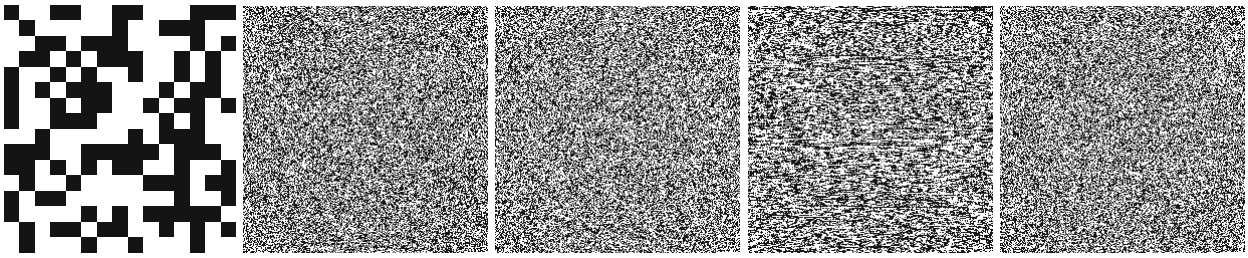

FIG. 4. a. Radiation, b. Random.org, c. Photocell, d. Speaker 5 ms, e. Speaker 50 ms

tal number of combinations to determine the probability that each bit string appeared. The Central Limit Theorem suggests that as the number of ones and zeros goes to infinity, the ideal probability of finding each combination should be equal [7].

While it is difficult to create a truly random source, mathematical methods exists to unbias a sample [8]. One such methods is called the Von Neumann strategy and it includes examining the transitions from one to zero and zero to one [5]. Even if the probability of obtaining a one is different than the probability of obtaining a zero, then the probability of a one following a zero and of a zero following a one should still be the same. Thus, the samples were unbiased by creating a new set which contains a zero anytime that a one followed a zero in the original data and it contains a one anytime a zero followed a one.

VI. RESULTS

The randomness testing program was applied to each set of data; the Random.org control, photocell data, 50 ms speaker data, 5 ms speaker data, and the radiation data.

Tables I and II show the statistical results of each sample. Table I shows the statistics for the percentage of the time that each possible combination of one bit, only zero or one, appeared in the samples. Table II shows the statistics for each five bit combination appearing in the samples. The Tables include the string that appeared the maximum percentage of the time and the minimum percentage of the time. They also include the standard deviation of the data points in each sample compared with the ideal percentage and the standard deviation of the mean. The ideal percentage for one bit is 50% for each combination and for five bits is 3.125% for each combination. The statistical tests were repeated up to 10 bits and the results continued the trends shown in Table II including Random.org having the lowest standard deviation and the speaker at 5 ms having the highest standard deviation.

The samples where then unbiased and the statistical tests were performed again. These results are shown in Tables III for the one bit test and IV for the five bit test. These show that there is a general improvement in the results apart from the control sample and the radiation sample. Unbiasing the radiation sample further limited the size of the sample and thus skewed the results.

Sample	Max (%)	Min (%)	s (%)	s_m (%)
Random.org	0: 50.40	1: 49.60	0.57	0.30
Speaker 50 ms	0: 52.10	1: 47.90	2.97	2.10
Photocell	0: 52.15	1: 47.85	3.04	2.15
Radiation	0: 52.94	1: 47.05	4.16	2.94
Speaker 5 ms	0: 52.49	1: 47.51	3.52	2.49

TABLE I. This table shows the statistical results of one bit for the biased data set. The ideal probability is 50% for each combination.

Sample	Max (%)	Min (%)	s (%)	s_m (%)
Random.org	00010: 3.33	01111: 2.96	0.09	0.02
Speaker 50 ms	00000: 3.75	11011: 2.57	0.31	0.05
Photocell	00000: 3.98	11111: 2.64	0.32	0.06
Radiation	01010: 5.98	01111: 0.85	1.35	0.24
Speaker 5 ms	00000: 13.28	10101: 1.10	2.55	0.45

TABLE II. This table shows the statistical results of five bits for the biased data set. The ideal probability is 3.125% for each combination.

Figure 4 shows images created by going line by line through a blank image and coloring the pixel black if the data is a one and white if the pixel is a zero. The radiation data had the lowest number of data points and this is why the resolution is much lower than the other images. The photocell, Random.org, and speaker at 50 ms images appear similar by eye. However, the speaker at 5 ms image has horizontal lines that can be seen repeating throughout the image. This is an indication that there are repeating bits which cause repetitive patterns to appear.

A. Error Analysis

The similarity between the 50 ms speaker data and the photocell data may be a byproduct of the method used to generate the numbers. It is possible that any noisy 50 ms data taken from the digital oscilloscope will result in similar output. This would be due to the way that the oscilloscope determines the value of the voltage at each point and how it determines if it is even or odd. However,

Sample	Max (%)	Min (%)	s (%)	s_m (%)
Random.org	0: 50.31	1: 49.68	0.45	0.32
Speaker 50 ms	0: 50.33	1: 49.67	0.47	0.33
Photocell	1: 50.03	0: 49.97	0.04	0.03
Radiation	1: 52.94	0: 47.06	4.16	2.94
Speaker 5 ms	0: 50.80	1: 49.20	1.13	0.80

TABLE III. This table shows the statistical results of one bit for the unbiased data set. The ideal probability is 50% for each combination.

Sample	Max (%)	Min (%)	s (%)	s_m (%)
Random.org	00110: 3.31	10110: 2.90	0.10	0.02
Speaker 50 ms	10010: 3.49	11101: 2.81	0.18	0.03
Photocell	01010: 3.48	11000: 2.73	0.18	0.03
Radiation	11001: 9.37	01010: 0.00	2.21	0.39
Speaker 5 ms	10100: 3.60	11111: 2.51	0.28	0.05

TABLE IV. This table shows the statistical results of five bits for the unbiased data set. The ideal probability is 3.125% for each combination.

the wave functions would have to be similarly noisy since the digital oscilloscope was also used to generate the 5 ms speaker data which was statistically different from the 50 ms data.

The radiation data was hypothesized to be one of the most accurate random number sources but the statistical tests showed it to be the second worst in terms of the samples that were tested. In order to compare it with the other samples, a random 240 bits were taken from each sample to compare. The program was used to determine the statistical properties of each sample. The results are shown in Table V for a one bit test and Table VI for a five bit test. The tests seem to break down for small samples since Table V shows the photocell and 50 ms speaker having a 0 standard deviation even though it is shown that they do not in Table I.

VII. CONCLUSION

Random number generators can be made from purely physical properties, however, in order to maximize entropy, the random number generator should be able to produce large amounts of data in a short amount of time.

Sample	Max (%)	Min (%)	s (%)	s_m (%)
Random.org	0: 52.92	1: 47.08	4.12	2.92
Speaker 50 ms	0 1: 50.0	0 1: 50.0	0.0	0.0
Photocell	0 1: 50.0	0 1: 50.0	0.0	0.0
Radiation	0: 52.94	1: 47.05	4.16	2.94
Speaker 5 ms	1: 52.50	0: 47.50	3.54	2.50

TABLE V. This table shows the statistical results of one bit for the biased data set where each sample is limited to a size of 240 bits. The ideal probability is 50% for each combination.

Sample	Max (%)	Min (%)	s (%)	s_m (%)
Random.org	01010: 7.20	00000: 0.42	1.57	0.28
Speaker 50 ms	00111: 5.51	10111: 1.27	0.80	0.14
Photocell	00100: 4.66	01011: 1.69	0.85	0.15
Radiation	01010: 5.98	01111: 0.85	1.35	0.24
Speaker 5 ms	11111: 13.56	11001: 0.42	2.53	0.45

TABLE VI. This table shows the statistical results of five bits for the biased data set where each sample is limited to a size of 240 bits. The ideal probability is 3.125% for each combination.

[1] Erica Klarreich, *Take a Chance, Scientists put randomness to work*, 2004, https://www.sciencenews.org/node/21114

[2] Ian Goldberg and David Wagner, *How secure is the World Wide Web?*, Berkeley, 1996, http://www.cs.berkeley.edu/~daw/papers/ddj-netscape.html

[3] Mads Haahr, *Introduction to Randomness and Random Numbers*, https://www.random.org/randomness/

[4] Lewis A. Riley, 2004, *Operational Amplifier Circuits*, http://webpages.ursinus.edu/lriley/ref/circuits/node5.html

[5] Paul Crowley, *Generating random binary data from Geiger counters*, http://www.ciphergoth.org/crypto/unbiasing/

[6] A. Rukhin,J. Soto,J. Nechvatal,M. Smid,E. Barker,S. Leigh,M. Levenson,M. Vangel,D. Banks,A. Heckert,J. Dray,S. Vo , *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications* 2010, http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf

[7] John Walker, *Introduction to Probability and Statistics*, http://www.fourmilab.ch/rpkp/experiments/statistics.html

[8] Michael Mitzenmacher, *Tossing a Biased Coin*, Digital Equipment Corporation, Systems Research Center, Palo Alto, CA, http://www.eecs.harvard.edu/~michaelm/coinflipext.pdf

VIII. APPENDIX

A. Statistical Methods

The Von Neumann approach states that for an event of probability p and an event of a different probability q, $pq = qp$. Thus if a even if a 1 and 0 appear at different frequencies, 01 should appear as often as 10.

The standard deviation was calculated using this equation.

$$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{N-1}} \tag{1}$$

The standard deviation of the mean was calculated using this equation.

$$s_m = \frac{s}{\sqrt{n}} \tag{2}$$

B. Computational Methods

The following code is the program used to calculate the statistical results.

```
def getBlockList(blocksize, size):
    last_it = 0
    bitstr = 0
    string = ''
    blocks = []
    for j in range(0,size-blocksize+1):
        for k in range(j, j+blocksize):
            string += str(random[k])
        blocks.append(string)
        string = ''
    return blocks

def dec_to_bin(x):
    return int(bin(x)[2:])

def getBitStrings(numbits):
    decimal = 2**numbits
    zeros = numbits
    string = ''
    bitstrings = []
    for i in range(0,decimal):
        zeros = numbits - len(str(
            ↪ dec_to_bin(i)))
        for j in range(0,zeros):
            string += '0'
        string += str(dec_to_bin(i))
        bitstrings.append(string)
        string = ''
    return bitstrings

def getSTD(n, totalsum):
    return (totalsum/(n-1))**(0.5)

numbits = 10
total = 0
random = #list containing random bit
    ↪ string
count = 0
totalsum = 0
maxp = 0
minp = 1
maxstr = ''
minstr = ''
for i in range(1,numbits+1):
    blocks = getBlockList(i,total)
    bitstrings = getBitStrings(i)
    for j in range(0,len(bitstrings)):
        for k in range(0,len(blocks)):
            if(blocks[k] == bitstrings[j
                ↪ ]):
                count +=1
        if(maxp < count/len(blocks)):
            maxp = (count/len(blocks))
            maxstr = str(bitstrings[j])
        elif(maxp == count/len(blocks)):
            maxstr = maxstr + " and "+
                ↪ str(bitstrings[j])
        if(minp > count/len(blocks)):
            minp = (count/len(blocks))
            minstr = str(bitstrings[j])
        elif(minp == count/len(blocks)):
            minstr = minstr + " and "+
                ↪ str(bitstrings[j])
        totalsum += ((1/len(bitstrings))
            ↪ - (count/len(blocks)))**2
        count = 0
    std = getSTD(len(bitstrings),
        ↪ totalsum)
    stdmean = std/((len(bitstrings))
        ↪ **(0.5))
    maxp = 0
    minp = 1
    totalsum = 0
```