ATMOSPHERIC NOISE INDUCED RANDOM GENERATION

ANDREW E. WILLIAMSON

Abbreviated Title: True Random

Word Count*: 1,522

Figures and Tables: 27

Last Date of Revision: 08 May 2012

KEYWORDS

- Random
- True
- True Random
- Atmospheric
- Noise
- Induced
- Generation
- Philosophical
- Nature
- Analyze
- Cycles
- C
- C++
- Java
- Real Studio
- Python

* Word Count is of all pages, including Title Page, Abstract, References, Figures and Tables

ABSTRACT

To research the philosophical nature of random events and to create tools to further the research of individuals working in the same subject matter.

Analyse the natural formation or cycles of words formed during random processes. Included is interest in the frequency of each letter in the English alphabet, with a focus on creating computer source code to aid in the research.

Utilize computer languages C, C++, Java, Real Studio, and Python to produce source code for the project. Use Eureqa Formulize (Scientific data mining software package that searches for mathematical patterns found in sampled data), PSPP (PSPP is a program for statistical analysis of sampled data).

Testing results of the true random generator include popular test such as Frequency Test; Runs Test; Sums Test; Chi Test; and Craps Test, which use and error percentage between 3% to 5%. It is worth noting that the user has complete control over the error percentage.

INTRODUCTION

For many years, obtaining a good random set has been confined to the domain of pseudo-random (PRNG) methods. The inherent problem with such methods is that they are deterministic and so are the machines that run the code. Computers are actually very poor tools for creating random behavior due to their nature. Creating a program to generate random behavior on a machine is, actually, paradoxical. Computers rely on well-ordered instructions that can easily be back engineered. In fact, if one writes a program to produce random bits, it is very easy to track back to the function's initial state. There are always arbitrary starting states using a seed to initiate the sequence used to create prngs. This will always produce the same results if the seeds are equal.

So is there a better method? Instead of using PRNGs an individual could use and, in fact, will use true random number generators called TRNGs. TRNGs are seeded from physical processes and are usually found in hardware that generates low-level, statistically random "noise" signals.

The projects aims are to create a True Random Generator using computers, which will generate random numbers, and letters that fit to mathematical models of probabilistic and statistical inference expectation.

If a sample of random numbers or bits can pass the entire test then it will be considered true iff the sample was not derived from a pseudo method.

MATERIALS AND METHODS

See Supplementary Material at True Random for full details that complement the Methods described here, allowing for methodical interpretation and reproducibility of the study.

SUBJECTS

Theory

Calculating for any positive integer's n and x $\binom{n}{x} = \frac{n!}{x!(n-x)!}$ the probability for any number in any number of iterations is $\frac{\binom{n}{x}}{\sum_{i=0}^{n}\binom{n}{x}}$ as more and more iterations are computed; the graph of the probability of a given number becomes smoother and approaches a normal distribution as a limit. The normal distribution gives the probability as: $\frac{1}{\sqrt{2\pi}\sigma} e^{-\left(\frac{(x-\mu)^2}{2\sigma^2}\right)}$ where $\mu = \frac{n}{2}$ and σ is the standard deviation. Certainly, any set of random numbers may include consecutive numbers. So the probability of consecutive numbers is $\frac{(d-p+1)10^{(d-p)}}{10^d}$ where the number of possible numbers of *d* digits which contain one or more sequences of *p*. When measuring the significance to experimentally discrete results one could use the chi-square $x^2 = \sum_{1 \le s \le k} \frac{(Y_s - np_s)^2}{np_s}$ an experiment with *d* degrees of freedom (where *d=k-1*, one less the number of possible outcomes) is consistent with the null hypothesis and can be calculated as: $Q_{x^2,d} = \left[2^{\frac{d}{2}}\Gamma\left(\frac{d}{2}\right)\right]^{-1}\int_{x^2}^{\infty}(t)^{\left(\frac{d}{2}-1\right)}e^{-t}dt$
Where Γ is the generalization of the factorial function to real and complex arguments.

By Andy Williamson

EXPERIMENTAL DESIGN

As with all programs accessing the Windows API we must trap errors and close all windows associated with the program so that a cleanup operation can be implemented. One of the first decisions that has to be made is what audio format should be used. For example, the sampling rate and number of channels must be decided. For a higher ideal frequency, we must choose a higher sampling rate. Unfortunately, there is always a trade-off as we increase the sampling rate to achieve higher audio frequency more coding becomes necessary. "Normally you should choose the lowest sampling rate suitable for your application, remembering that it needs to be at least double the highest audio frequency in which you are interested" (according to the Nyquist criterion).
First code must be written to set up our audio format while including wave input device as follows.

```
340 DIM Format{wFormatTag{l&,h&}, nChannels{l&,h&}, nSamplesPerSec%, \ 350 \
nAvgBytesPerSec%, nBlockAlign{l&,h&}, wBitsPerSample{l&,h&}, \ 360 \ cbSize{l&,h&}} 370
```

```
380 Format.wFormatTag.l& = 1 : REM WAVE_FORMAT_PCM 390 Format.nChannels.l& = 1 :
REM Monaural 400 Format.nSamplesPerSec% = 44100 410 Format.wBitsPerSample.l& = 16 420
Format.nBlockAlign.l& = Format.nChannels.l& * Format.wBitsPerSample.l& / 8 430
Format.nAvgBytesPerSec% = Format.nSamplesPerSec% * Format.nBlockAlign.l& 440 450
_WAVE_MAPPER = -1 460 SYS "waveInOpen", ^WaveIn%, _WAVE_MAPPER, Format{}, 0, 0,
0 TO ret% 470 IF ret% ERROR 100, "waveInOpen failed: "+STR$~ret%
```

For this project, I chose a sampling rate of 44100 Hz. This range is more adequate for the project. 44100 Hz was chosen because most audio input may be as low as 11025 HZ even when stating a rating of 44100 Hz. The next logical step in the creation of the code is to create and initialize a few buffers. In this project, I created three and all are reused cyclically. The first buffer is set up for inputting the sampled sound, the second buffer is actually being processed by the program, while the third buffer was used as a spare. In order to avoid loss of data, the program has to be set up to process the input of data quickly. To minimize latency, it is best to increase the amount of buffers this works best rather than making larger buffers while working out the fluctuations in this dichotomy. Here is the code used for buffering note there are three buffers with 1024 samples at 44100 Hz. Thus, we can expect a latency of at least 24 ms.

```
530 nBuffers% = 3 540 SamplesPerBuffer% = 1024 550 BytesPerBuffer% = SamplesPerBuffer% *
Format.nBlockAlign.l& 560 570 DIM _WAVEHDR{lpData%, dwBufferLength%,
dwBytesRecorded%, dwUser%, \ 580 \ dwFlags%, dwLoops%, lpNext%, Reserved%} 590 600 DIM
Headers{(nBuffers%-1)} = _WAVEHDR{} 610 620 FOR buff% = 0 TO nBuffers%-1 630 DIM
buffer% BytesPerBuffer% - 1 640 650

Headers{(buff%)}.lpData% = buffer% 660 Headers{(buff%)}.dwBufferLength% =
BytesPerBuffer% 670 680 SYS "waveInPrepareHeader", WaveIn%, Headers{(buff%)},
DIM(_WAVEHDR{}) TO ret% 690 IF ret% ERROR 100, "waveInPrepareHeader failed:
"+STR$~ret% 700 710 SYS "waveInAddBuffer", WaveIn%, Headers{(buff%)},
DIM(_WAVEHDR{}) TO ret% 720 IF ret% ERROR 100, "waveInAddBuffer failed: "+STR$~ret%
730 NEXT
```

Please note that in this code I have allocated the buffers using BASIC's heap, However I could have used the Windows API to allocate the memory. After preparing a way to input real-time audio capture the audio should be set up.

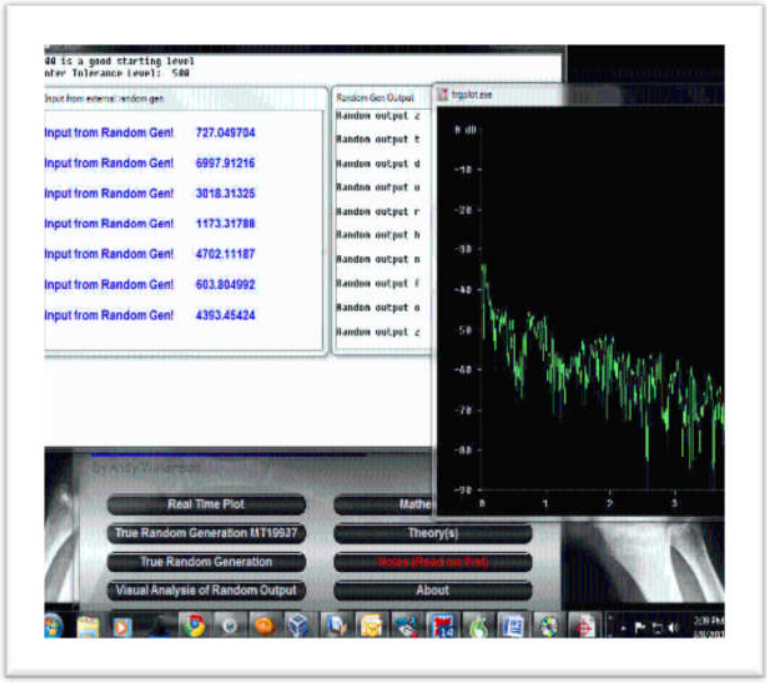
```
820 SYS "waveInStart", WaveIn% TO ret% 830 IF ret% ERROR 100, "waveInStart failed:
"+STR$~ret%
```

In order to make sure the program can keep up with the received audio implementation, the next code decides if the buffers need to process.

```
870 _WHDR_DONE = 1 880 REPEAT 890 FOR buff% = 0 TO nBuffers%-1 900 IF
Headers{(buff%)}.dwFlags% AND _WHDR_DONE THEN 910
PROCprocessbuffer(Headers{(buff%)}.lpData%, SamplesPerBuffer%) 920
Headers{(buff%)}.dwFlags% AND= NOT _WHDR_DONE 930 SYS "waveInAddBuffer",
WaveIn%, Headers{(buff%)}, DIM(_WAVEHDR{}) 940 ENDIF 950 NEXT 960 SYS "Sleep", 1
970 UNTIL FALSE
```

Processing the audio data is the most important aspect of the project. However, I prefer to keep this process proprietary. Each audio sample consists of a signed 16-bit value in the range

-32768 to +32767. Lastly, a standard cleanup routine should be implemented. The above code is in BBC Basic however, the final program was coded in Python.



First code using BBC Basic

STIMULI

Background

$SPL = 20\log_{10}\left(\frac{P}{P_{ref}}\right)$ Sound Pressure Level

$P = P_{ref} \times 10^{\frac{SPL}{20}}$ Sound Pressure or Reference Pressure

$IL = 10\log_{10}\left(\frac{I}{I_0}\right)$ Sound Intensity Level or Sound Intensity

$I = \frac{P_{AV}}{4\pi r^2}$ Intensity at a distance from point source

$frequency = \frac{velocity}{wavelength}$ Wave Velocity or Wave Length

Where

SPL = sound pressure level decibels (db)

P = sound wave pressure, newton's/meter2

Pref = reference pressure or hearing threshold, newton/meter2

IL = intensity level, decibel (db)

I = sound intensity, watt

I0= reference intensity or least audible sound level, watt

PAV = average power, watt

NPL = noise pollution level, decibel (db)

Notes:

- Usually, I0 is set to 10-12 watts

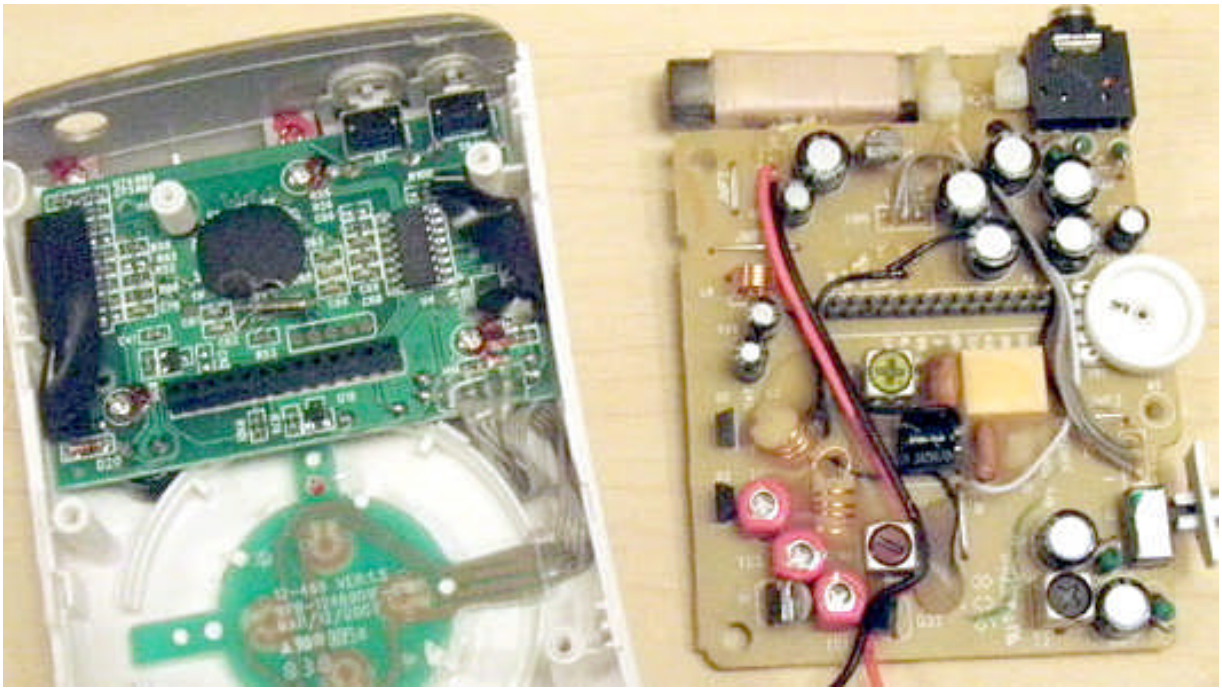
- Usually, Pref is set to 0.00002 newton's/meter2

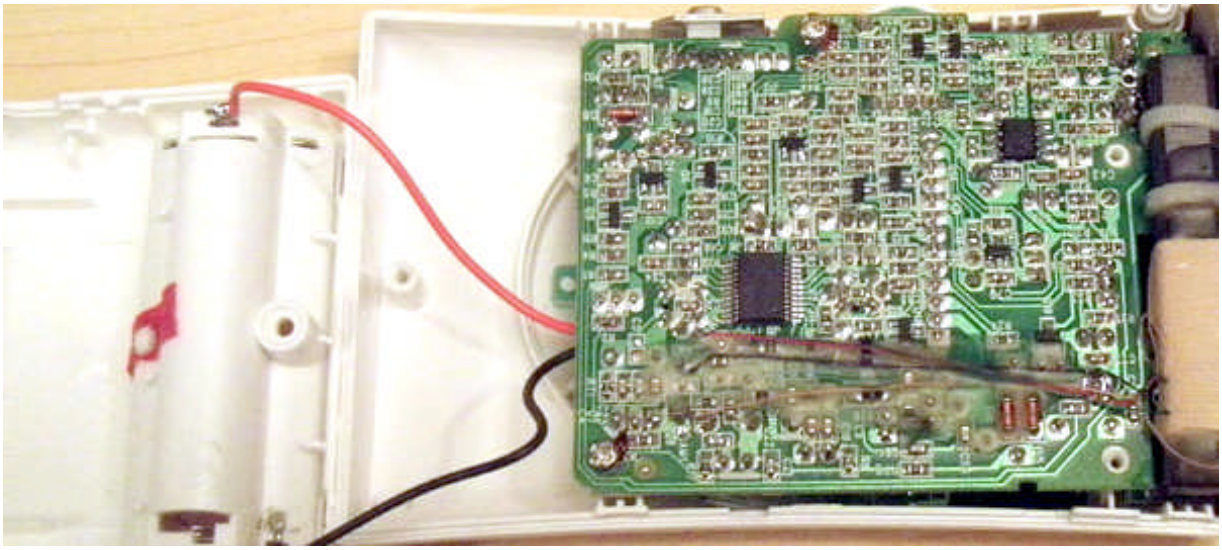
References - Books:

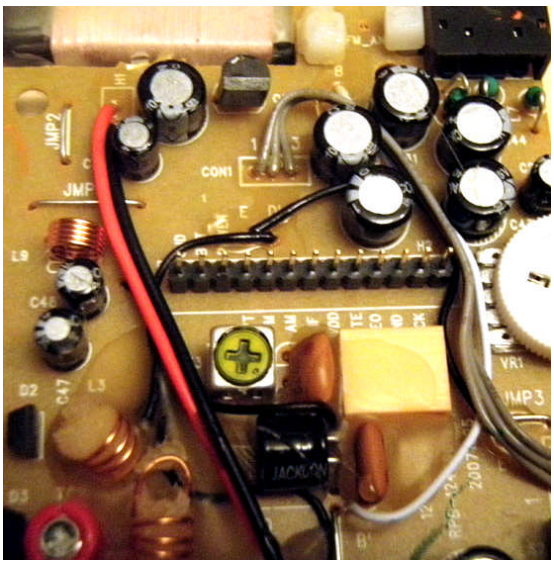
1) P. Aarne Vesilind, J. Jeffrey Peirce and Ruth F. Weiner. 1994. Environmental Engineering. Butterworth Heinemann. 3rd ed.

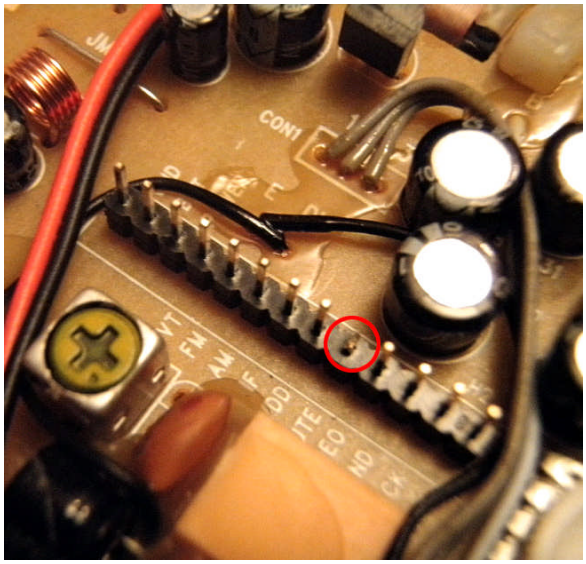
2) Tipler, Paul A.. 1995. Physics For Scientists and Engineers. Worth Publishers. 3rd ed.

MEASURES AND MEASUREMENT DEVICES









DATA ANALYSIS

At this time, processing the audio data will remain proprietary. However, a use of the quadratic mean, also called the Root-Mean-Square RMS, plays a minor role.

$$\{x | x \in \boldsymbol{R}, x_1, \ldots, x_n\}$$

$$x_{RMS} = \sqrt{\frac{x^2_1 + x^2_2 + \ldots + x^2_n}{n}}$$

$$= \sqrt{\frac{\sum_{i=1}^{n} x^2_i}{n}}$$

$$= \sqrt{(x^2)}$$

Where (x^2) denotes the mean of the values $(x^2{}_i)$ values of a discrete distribution. For a variant from a continuous distribution $p(x)$ $x_{RMS} = \sqrt{\frac{\int [p(x)]^2 \, dx}{\int p(x) \, dx}}$ where the integrals are taken over the domain of the distribution. In the application of this project the root-mean-square stands for the standard deviation and the square root of the mean squared deviation of a signal from a given baseline or fit.

Weisstein, Eric W. "Root-Mean-Square." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/Root-Mean-Square.html

RESULTS

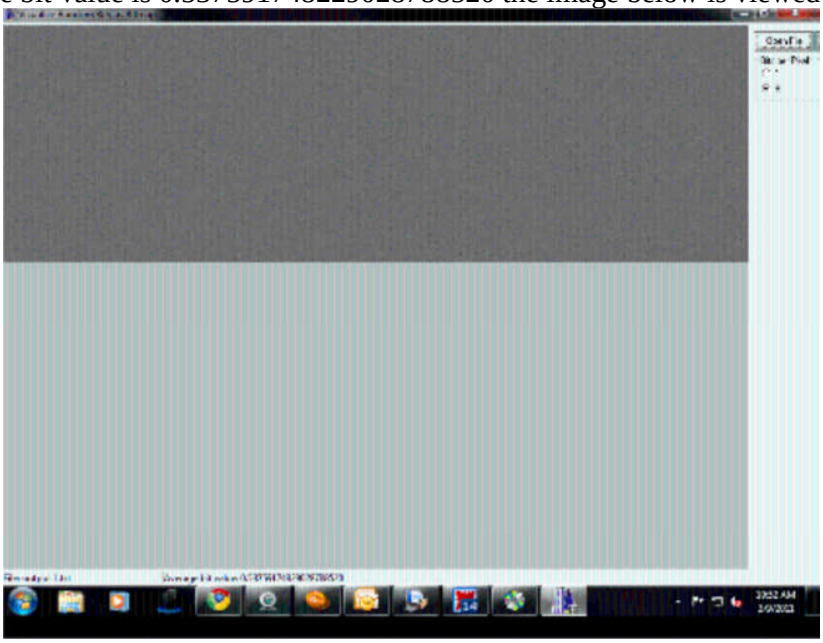
This experiment has achieved significant results, the following test have passed Frequency Test; Runs Test; Sums Test; Chi Test; and Craps Test.

See Supplementary Material at True Random Generator for full details of all the results including raw data obtained from this study.

Graphs of Processed Data

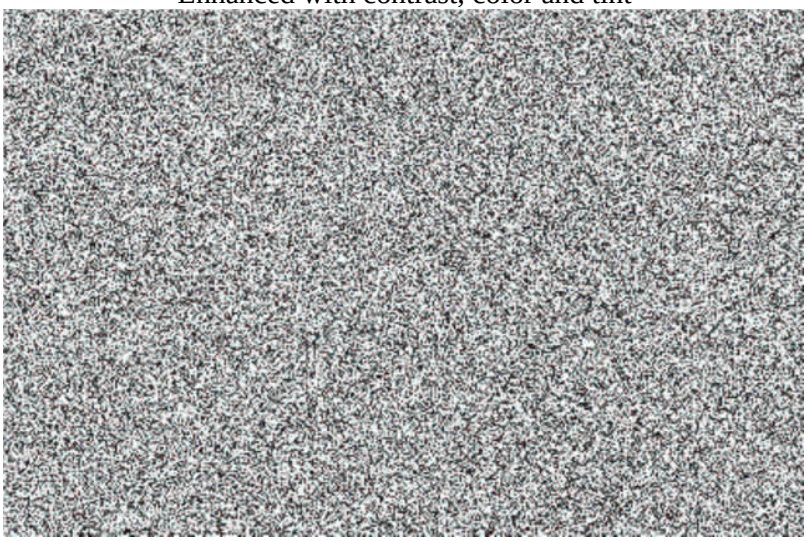
Visual distribution of bits.

Average bit value is 0.537591748229028788520 the image below is viewed in 8bit.

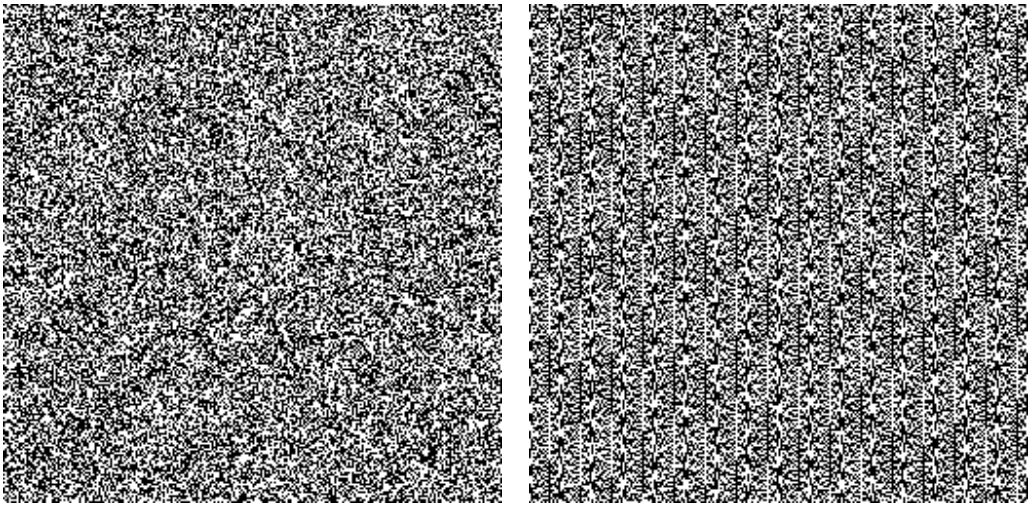


A magnification of bit distribution at 625% in 8bits no enhancement.

Enhanced with contrast, color and tint



A True Random Number Generator (TRNG), and the bitmap on the right with the rand() function from PHP on Microsoft Windows, which is a Pseudo-Random Number Generator (PRNG).



TRNG **PHP rand() on Microsoft Windows**

DISCUSSION

Summary of Main Findings

Interpretation of Own Data

Converging and Conflicting Evidence

General Limitations and Assumptions

Further Directions and Experiments

Implications / Significance of the Study

Conclusions and Key Points

ACKNOWLEDGEMENTS

Funding Bodies: Andy Williamson

Collaborators that Are Not Co-Authors: David Allen Rainey, Dustin Gibson

Advice and Sharing of Expertise: David Allen Rainey, Dustin Gibson

REFERENCES

Weisstein, Eric W. "Root-Mean-Square." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/Root-Mean-Square.html