Can computers generate a truly random number?

Connie

If you ask another person to select a random number between 1 and 10, chances are that the number they pick will be 7[1] at 22.50% of the time (more than twice its expected frequency), and chances of them picking an odd number are higher than that of an even number, at 68.35%. Evidently, humans are not effective at generating truly random numbers, but how about computers?

Pseudo Random Number Generators

The idea of computers generating 'random' numbers can seem strange or even impossibly, as the fundamentals of computer science rely on computers using algorithms and sets of instructions to run programs. However it can be done using an algorithm known as pseudo-random number generators, or PRNGS. These algorithms rely on mathematical formulas to produce sequences of numbers that only appear random. The numbers generated are predictable if you know the algorithm and the initial 'seed' value used to start the process.

One of the oldest PRNGs is the **linear congruential generator**[2]. Which is defined as:

$$X_{n+1} = (aX_n + c) \bmod m$$

(where x is the seed/starting value, m is the modulus, c is the increment and a is the multiplier)

Each number in the sequence is generated by applying this formula repeatedly, with the values of *a*, *c*, and *m* carefully chosen to ensure good randomness properties.

Example parameters often used in basic LCGs.

- Multiplier a=1664525
- Increment c=1013904223
- Modulus m=2^{32} (which aligns with the word size of many computers).

Limits of PRNGS

Kolmogorov Complexity, introduced by Andrey Kolmogorov, measures the complexity or randomness of a string of data based on how compressible it is. A string is considered Kolmogorov random if the length of the program encoding it is longer than the string itself.

Here is an example of two strings.

randomrandomrandomrandomrandomrandomrandomrandomrandom

The string above would not be considered Kolmogorov random as it has a clear, simple pattern to it. Hereby showing it has a low Kolmogorov complexity.

qreuyiqireufhdnsowfbdhyprt

This string is a mash of ‘random’ keys, and would be considered Kolmogorov random as it has no clear pattern to it.

In relation to the PRNG we looked at above, the LCG, while the numbers it produces might appear random, the fact that they are generated by this short formula shows that the sequence has low Kolmogorov complexity. The algorithm and seed can produce the entire sequence, meaning it is easily compressible and therefore not **truly** random.

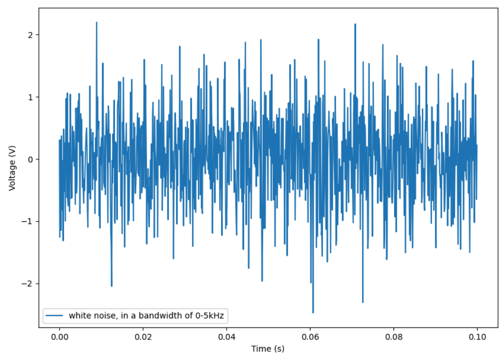
True Randomness

True Random Number Generators (TRNGs) are essential components in cryptography, ensuring the unpredictability of encryption keys and securing sensitive data. Unlike PRNGs, which rely on algorithms and can be theoretically predicted, TRNGs derive randomness from **physical processes**, providing genuinely unpredictable numbers. This hence makes them an example of Kolmogorov randomness.

Historically, physical devices have been used to predict random numbers for thousands of years. Some simple examples include dice (primarily used in gambling) or flipping a coin, the usage of which dates back to Ancient Rome.

In 2014, researchers **Mario Stipčević** and **Çetin Kaya Koç** categorised TRNGs into four key families[4]

- electrical noise
- free-running oscillators
- chaos
- quantum effects.



Electrical noise refers to tiny, random fluctuations in voltage or current that occur naturally in electronic components. This noise is chaotic and impossible to predict, making it a perfect source of randomness.

To the right is a photo example showing fluctuations in voltage, with no discernible pattern.

Free-running oscillators TRNGs work by utilising one or more free-running oscillators with slightly different frequencies. By comparing the timings of their oscillations, the TRNG detects when these signals line up. As the frequencies are not perfectly synced, these crossing points happen at random intervals, generating random bits (0s and 1s).

Chaos-based TRNGs rely on the unpredictable behaviour of chaotic systems to generate random numbers.

Chaos refers to systems that are highly sensitive to initial conditions. An example of this is the butterfly effect[5], in which a small change such as a butterfly flapping its wings could cause a hurricane or blizzard on the opposite side of the world. Another example of chaos theory is weather patterns - small changes lead to huge differences over time.

These TRNGs use physical systems that exhibit chaotic behaviour, like lasers, electronic circuits, or mechanical systems. These systems are set up to operate in chaotic modes, where their output is extremely sensitive to small changes. This unpredictability becomes the source of randomness.

Quantum based TRNGs generate random numbers by exploiting the unpredictability of quantum processes, such as photon emission and nuclear decay. Nuclear decay occurs when the nucleus of an atom is unstable and spontaneously emits energy in the form of radiation at unpredictable intervals. By measuring the precise timing between these decay events, these TRNGs can produce a stream of random numbers.

Why do random numbers matter?

By now we understand the key differences between PRNGs and TRNGs, and what defines a number as ‘random’, but what are the actual applications of this?

Random numbers have a variety of use cases, in cybersecurity and cryptography they can be used to generate truly secure encryption keys that cannot be predicted. Or, they can be used in gaming to allow developers to generate unpredictable events that therefore improve the games replayability, keeping it from getting monotonous and repetitive, and the overall user experience.

Conclusion

So to conclude, to answer our beginning question - can computers generate random numbers? Sort of. Computers by themself run on algorithms and deterministic methods, hence they have no way of producing something truly 'random' that does not follow a set formula and initial seed value. However, they can generate truly random numbers if they use external physical processes that we discussed previously.

Bibliography:

1.Saito, M. (1998) *Comparative cross-cultural color preference and its structure*. Available at: https://sites.socsci.uci.edu/~kjameson/ECST/Saito_ComparativeCrossCulturalColorPreferenceAndItsStructure.pdf

2. L'Ecuyer, P., & Simard, R. (1999). *Beware of Linear Congruential Generators with Modulus Power of Two*. Mathematics of Computation, 68(225), 249–264. Available at: https://doi.org/10.1090/S0025-5718-99-00996-5

3. Koucký, M. (2020). *Kolmogorov Complexity*. Charles University. Available at: https://iuuk.mff.cuni.cz/~koucky/vyuka/TI-LS2020/kolmcomp.pdf

4. Çetinkaya Koc, Ç. (2014). *Randomness Generators: A Comparison of Hardware and Software-Based Systems*. Oregon State University. Available at: https://cetinkayakoc.net/docs/b08.pdf.

5. Image Source - 'Beautiful butterfly blue abstract with zoom effect' by *Borchee* via Getty Images, available at Getty Images UK.