Random Number Generation

Who needs randomness?

- Cryptographic protocols need randomness:

- "generate a random session key"
- "generate a random prime"
- "generate a random 64-bit challenge"
- ...

- … the generation process is not always specified.

- Randomized algorithms, simulations, games, stochastic modeling use randomness

What can happen is random numbers are not random

Some examples:

- 2008: Random Number Bug in Debian Linux [1]
  - 'custom' OpenSSL implementation only produced 32,768 possible TLS keys
- 2012: repeated public keys, factorable public keys [2]
  - appear to come from embedded devices with weak RNGs

https://www.nytimes.com/2012/02/15/technology/researchers-find-flaw-in-an-online-encryption-method.html

[1] https://www.schneier.com/blog/archives/2008/05/random_number_b.html

[2] https://freedom-to-tinker.com/blog/nadiah/new-research-theres-no-need-panic-over-factorable-keys-just-mind-your-ps-and-qs/

Random number generator

- A box/function that generates outputs.

- Output is from an alphabet. (binary, q-ary, ..)

Questions:

- How to generate randomness?
- How to evaluate randomness?

Random Number Generators

- Pseudorandom number generators (PRNG)
  - Repeated use of an algorithm
    - Starting from a "seed".

- True random number generators (TRNG)
  - Use physical sources with inherent randomness
    - Thermal noise of a diode
    - Atmospheric noise
    - Computer hardware random properties
    - ..

PRNG

- Important properties:

1. Statistical uniformity
   - e.g., same number of 0 and 1

2. Unpredictability
   - e.g. Given n previous values, the next bit cannot be guessed with a probability better than 1/2

PRNG in C/C++

- **int rand(void)**
  - returns a PRN in range [0 RAND_MAX]
  - RAND_MAX**:** a constant that depends on implementation, at least 32767.

```
int main(void)
{
        for(int i = 0; i<5; i++)
        printf(" %d ", rand());
    return 0;
}
```

→ 45 1762 425 891 always

- **srand() — Set Seed for rand() Function**.

```
int main(void)
{
        srand(time(0));

    for(int i = 0; i<5; i++)
        printf(" %d ", rand());

    return 0;
}
```

PRNG in C/C++

POSIX.1-2001 gives the following example of an implementation of **rand**() and **srand**(), possibly useful when one needs the same sequence on two different machines.

```
static unsigned long next = 1;

/* RAND_MAX assumed to be 32767 */
int myrand(void) {
    next = next * 1103515245 + 12345;
    return((unsigned)(next/65536) % 32768);
}

void mysrand(unsigned seed) {
    next = seed;
}
```

PRNG

- Linear Congruential RNG

  $X_{n+1} = aX_n + b \mod m$

- Parameters are: m is modulus, a and b and constants

- Period is at most m
- Common values: 2^{32}, 2^{64}

- Statistical uniformity only
  - knowing a few consecutive values reveals the parameters → predictable after a few elements

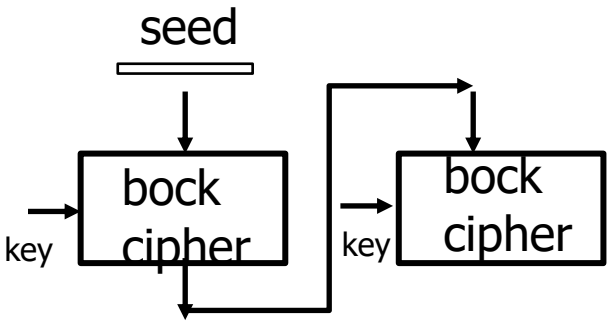
PRG: example

- PRNGs have a period:
  - Repeat after the period
- $x_{n+1} = 3\,x_n + 7 \bmod 19$
- Seed $= x_0 = 5$
- 3, 16, 17, 1, 10, 18, 4, 0, 7, 9, 15, 14, 11, 2, 13, 8, 12, 5, 3, 16,…
- No new entropy is introduced in each output.

Unpredictability through crypto

- Cryptographically secure PRNG

1. Block-ciphers in OFB mode



2. Based on hardness assumption
   - Blum Blum Shub (1966)
   - $X_{n+1} = X^2_n \mod M$ where M=p.q
     - p and q are two large primes.
   - The output is "unpredictable" if the adversary does not have enough computational power

TRNG

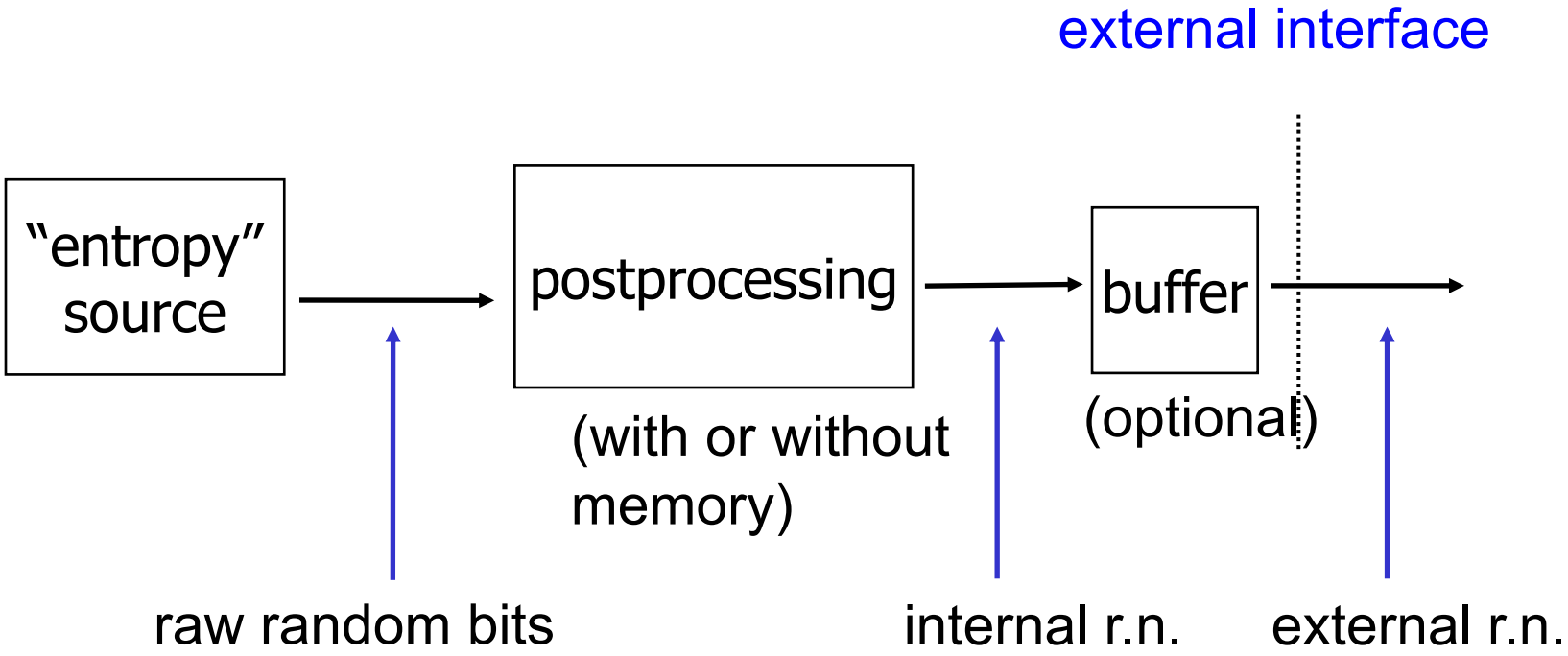
- TRNG is an algorithm that uses an entropy source to generate a random sequence.

- In computers TRNG use,
  - Randomness in the environment
  - Randomness generated in the processor
    - Intel on-chip entropy source

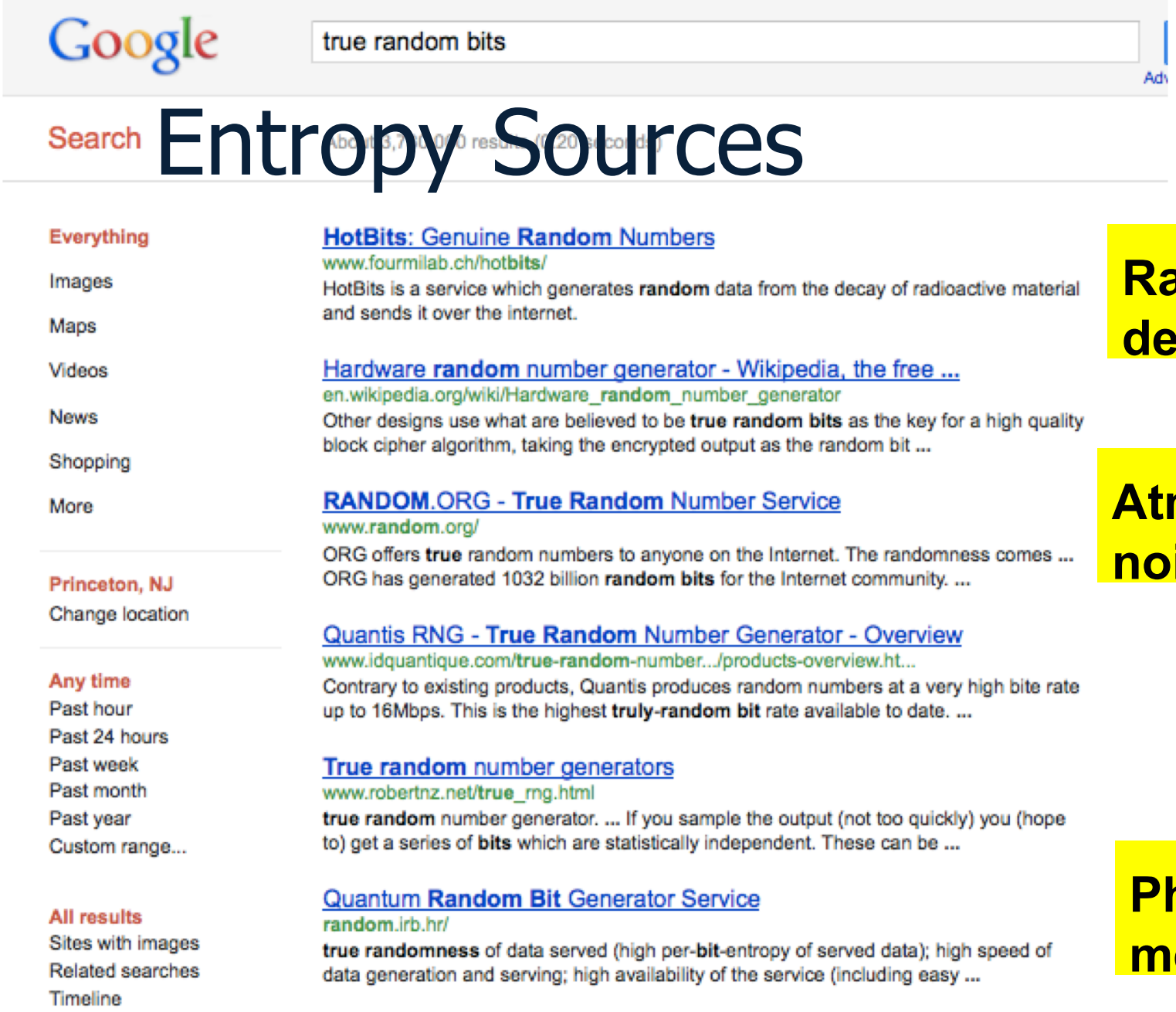
- Operating systems like Linux implement a TRNG
  - Special files /dev/random and /dev/urandom
  - /dev/random contains true random numbers
  - /dev/urandom contains pseudorandom numbers
  - They use environmental random events collected from device drivers that are further processed.

True RNG structure

- "Entropy" source is usually does not have uniform independently generated output.



- Post-processing extracts the randomness from collected entropy.



Entropy Sources

Radiactive decay

Atmospheric noise

Photons measurement

"Randomness" of a string

- Is 1000101111111110 "random"?
- Using randomness tests:
- Tests are used to evaluate "uniformity" and "unpredictability"
  - "Same" number of 0 and 1
  - Same number of 00, 11, 10 and 01
  - Same number of 000, 110, 100, 001, 111, 101, 010, 011
  - ..

NIST Statistical Test Suite

- A statistical framework:
- A set of 15 tests

1. The Frequency (Monobit) Test,
2. Frequency Test within a Block,
3. The Runs Test,
4. Tests for the Longest-Run-of-Ones in a Block,
5. The Binary Matrix Rank Test,
6. The Discrete Fourier Transform (Spectral) Test,
7. The Non-overlapping Template Matching Test,
8. The Overlapping Template Matching Test,
9. Maurer's "Universal Statistical" Test,
10. The Linear Complexity Test,
11. The Serial Test,
12. The Approximate Entropy Test,
13. The Cumulative Sums (Cusums) Test,
14. The Random Excursions Test, and
15. The Random Excursions Variant Test.

The first 5 pages, and Frequency test.

http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf

What do we mean by ”random sequence”?

- Informally,
- It is generated by a good TRNG

- Capturing this property through tests

Example

- 00000
- 10000
- 01000
- 00100
- 00010
- 00001
- 11000
- 10100
- 10010
- 10001
- 01100
- 01010
- 01001
- 00110
- 00101
- 00011
- 11100
- 11010
- 11001
- 01110
- 01101
- 01011
- 00111
- 10110
- 10101
- 11110
- 11101
- 11011
- 10111
- 01111
- 11111

-19 sequences have "almost" the same number of 0,1
(2 or 3 ones.)
$p_{\{2,3\}} = 0.59$

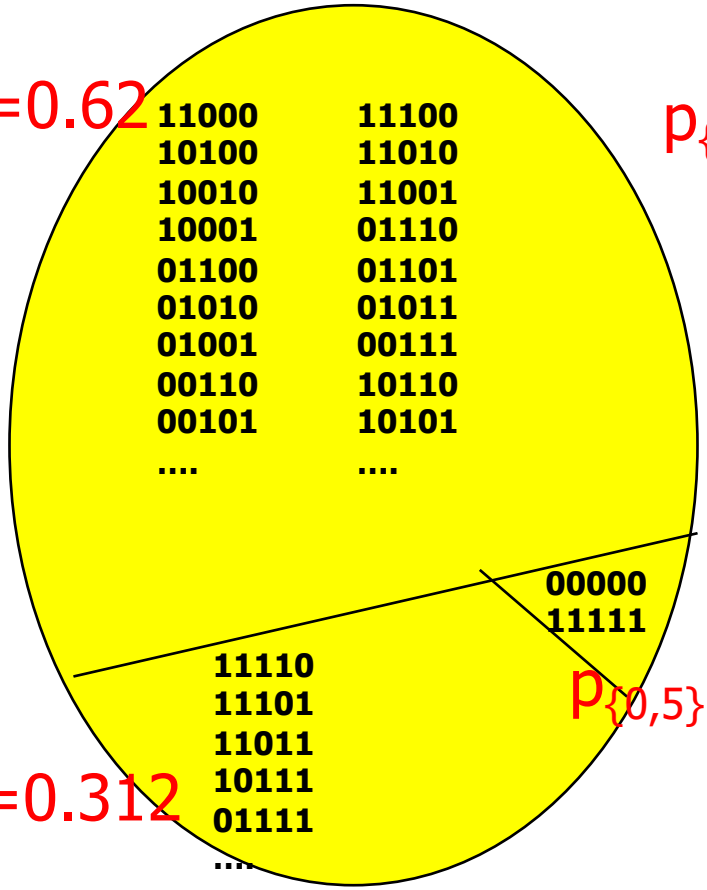
- 5 sequences with a four 1
5 sequences with a single 1
$p_{\{4,1\}} = 0.31$

-1 sequence with zero 1
1 sequence with five 1
$p_{\{5,0\}} = 0.06$

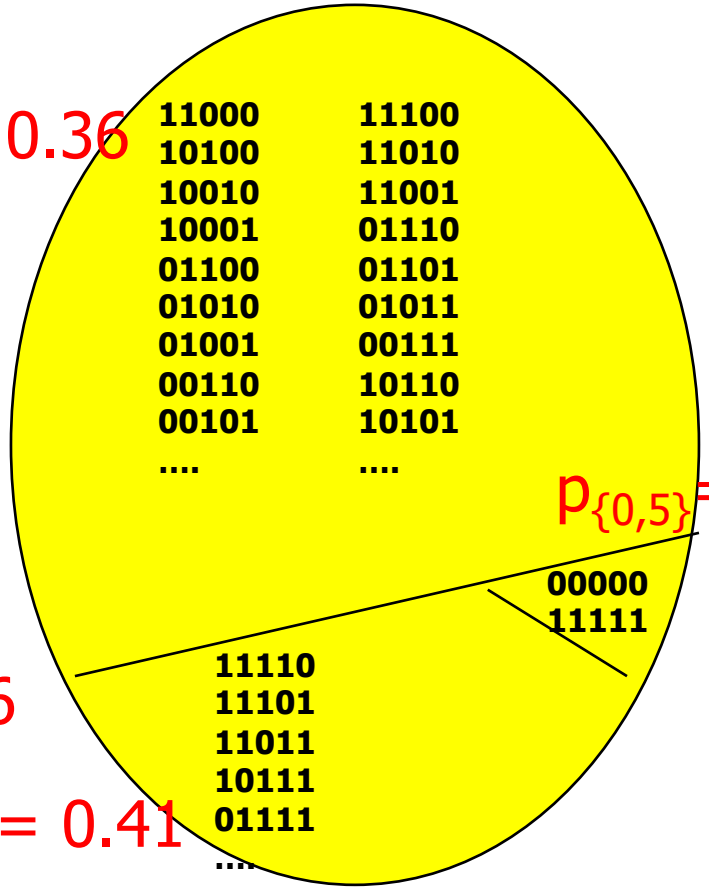
Deciding a sequence is "random"

(generated by a fair coin)

Fair coin



Biased coin: p(0)=3/4



Hypothesis testing &Statistical tests

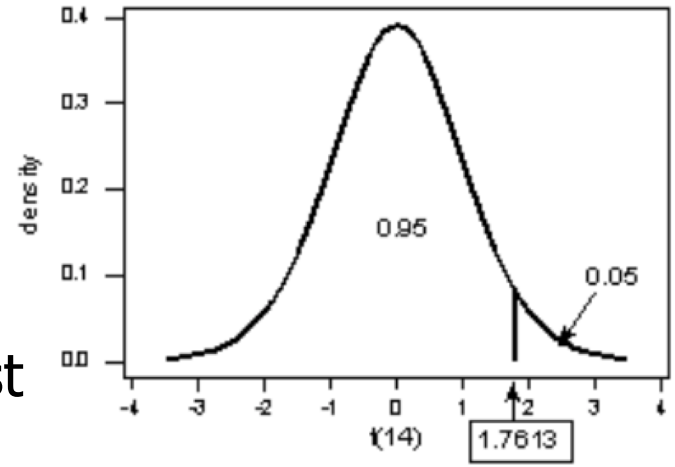
- Given the sequence 00100, which of the two hypotheses hold:

- Null Hypothesis H_0: "the sequence is generated by a fair coin (the generator is a good RNG)"
- Alternative Hypothesis: H_a is "the sequence is not generated by a good generator"

- Use a statistical test to decide:
- Statistical tests rely on test statistics:
  - A "test statistics" is a value that can be calculated for a sequence and has a known distribution under H_0

Statistical tests

- A "test statistics" that,
  - can be calculated for a sequence
  - has a known distribution under H_0
- Choose a significance level α
  - Probability of Type I error
- Using the known distribution of the test statistic, calculate the P-value
- If *P-value* <α, then the null hypothesis is rejected
  - the sequence appears to be random.
- If *P-value* ≥α, the null hypothesis is accepted; i.e.,



NIST Frequency (Monobit) Test

- Calculate test statistics and P-value
- Choose a significance level
- Decide by comparing the p-value with the significance level

For example, if $\varepsilon = 1011010101$, then n=10 and $S_n = 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) + 1 + (-1) + 1 = 2$.

(2) Compute the test statistic $s_{obs} = \dfrac{|S_n|}{\sqrt{n}}$

For the example in this section, $s_{obs} = \dfrac{|2|}{\sqrt{10}} = .632455532.$

(3) Compute $P\text{-}value = \textbf{\textit{erfc}}\left(\dfrac{s_{obs}}{\sqrt{2}}\right)$, where ***erfc*** is the complementary error function as defined in Section 5.5.3.

Complementary Error Function

$$erfc(z) = \frac{2}{\sqrt{\pi}}\int_z^\infty e^{-u^2}\,du$$

For the example in this section, $P\text{-}value = \textbf{\textit{erfc}}\left(\dfrac{.632455532}{\sqrt{2}}\right) = 0.527089.$

2.1.5 Decision Rule (at the 1% Level)

If the computed *P-value* is < 0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

2.1.6 Conclusion and Interpretation of Results

Since the *P-value* obtained in step 3 of Section 2.1.4 is ≥ 0.01 (i.e., *P-value* = 0.527089), the conclusion is that the sequence is random.

For example, if $\varepsilon = 10$
$(-1) + 1 = 2$.

(2) Compute the test statis

For the example in this

(3) Compute *P-value* = *er*

Section 5.5.3.

For the example in this

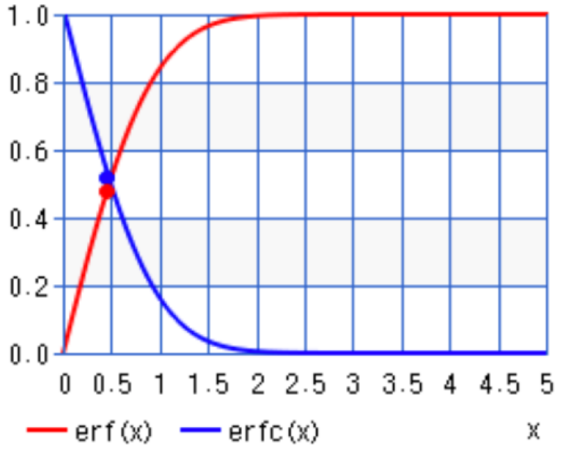
2.1.5 Decision Rule (at th

If the computed *P-value* is < 0.
that the sequence is random.

Error function Calculator

Home / Special Function / Error function

Calculates the error function erf(x) and complementary error function erfc(x).



x 0.632/1.4 real number

Execute Clear Store/Read Print 22digit

Error function	result
● erf(x)	0.476797347254205418 1176
● erfc(x)	0.523202652745794581 8824

2.1.6 Conclusion and Interpretation of Results

Since the *P-value* obtained in step 3 of Section 2.1.4 is ≥ 0.01 (i.e., *P-value* = 0.527089), the conclusion is
that the sequence is random.