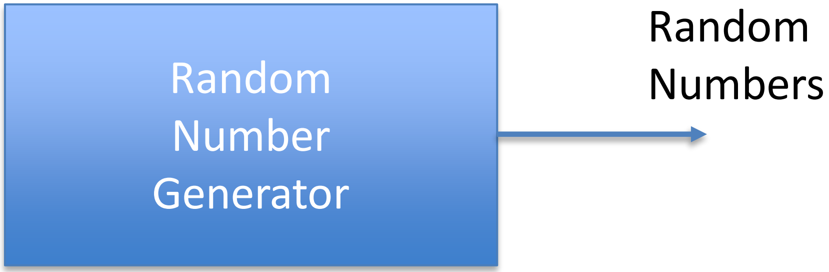
Statistical Tests for RNGs

Cryptography on Hardware Platform

Sujoy Sinha Roy
sujoy.sinharoy@iaik.tugraz.at

TU Graz



How could we verify that the numbers produced are indeed random?

Random bit sequence

NIST’s definition: A **random bit sequence** could be interpreted as the result of

- Flips of an unbiased ‘fair’ coin with sides labeled ‘0’ and ‘1’,
- With each flip having a probability of exactly 1/2 of producing a ‘0’ or ‘1’,
- And the flips are independent of each other.

Independent, identically distributed (IID) and unbiased.

Statistical Tests for Random Numbers

Goal: Check whether a given binary sequence is random or not

A statistical test is formulated to test null hypothesis

- Null Hypothesis (H0): the sequence being tested is random
- Alternate Hypothesis (Ha): the sequence is not random

The test accepts or rejects the null hypothesis, i.e., whether the sequence is (or is not) random.

NIST's random number generation tests

The NIST Test Suite is a package of 15 statistical hypothesis tests to test the randomness of arbitrary long binary sequences.

1. Frequency (monobit) test
2. Frequency test within a block
3. Runs test
4. Test for longest-run-of-ones in a block
5. Binary matrix rank test
6. Discrete Fourier transform (spectral) test
7. Non-overlapping template matching test
8. Overlapping template matching test
9. Maurer's 'Universal Statistical' test
10. Linear complexity test
11. Serial test
12. Approximate entropy test
13. Cumulative sums test
14. Random excursions test
15. Random excursions variant test

NIST's statistical tests: Their general framework

RNG under test

↓

Step1: Collect bits of sufficient length

↓

Step2: Run a statistical test and compute the test statistic.

↓

Step3: Compute the P_{value}

↓

Step4: Compare P_{value} with level of significance α (generally α =0.01)

- If $P_{value} > \alpha$, then H0 accepted → Sequence is random
- Else, H0 rejected → Sequence is non-random

For α =0.01 confidence is 99%

P-value is the probability that a 'perfect RNG' would have produced a sequence less random than the sequence that was tested.

NIST's statistical tests: Possible outcomes from a statistical test

A statistical hypothesis testing has two possible outcomes: accept or reject H_0.

TRUE SITUATION	CONCLUSION	
	Accept H_0	Accept H_a (reject H_0)
Data is random (H_0 is true)	No error	Type I error
Data is not random (H_a is true)	Type II error	No error

(Image source: [NIST])

Like any statistical testing, there can be Type-I and Type-II errors.

Type-I error: Test indicates that sequence is not-random when it really is random.
The probability of Type-I error is the 'level of significance' α.

Type-II error: Test indicates that sequence is random when it isn't.

NIST’s statistical tests

“A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications” by NIST. Date Published: April 2010.
https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final

NIST's statistical tests: Two important functions

The tests use two functions for computing the P_{value}

1. The Gauss error function: $$erfc(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-u^2} du$$

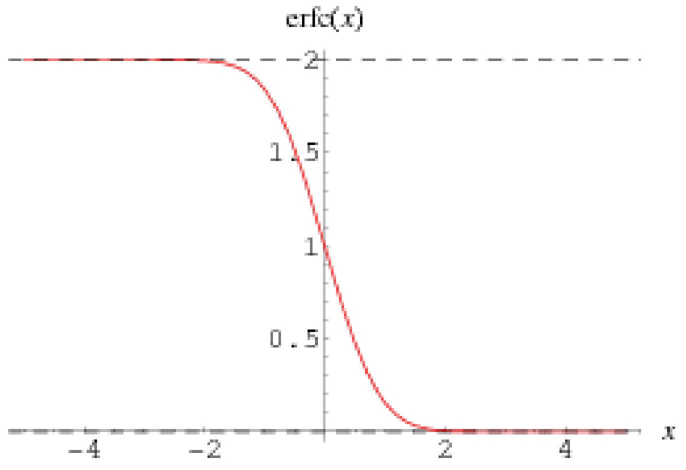


Image source: https://mathworld.wolfram.com/Erfc.html

NIST's statistical tests: Two important functions

The tests use two functions for computing the P_{value}

2. The incomplete gamma function: $$igamc(a,x) = \int_x^\infty u^{a-1}e^{-u}du$$

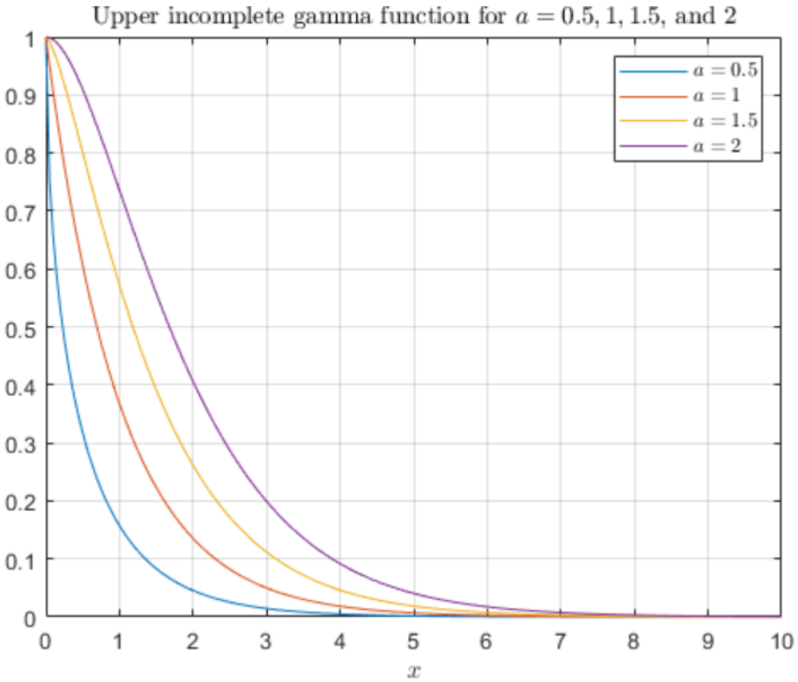


Image source:
https://nl.mathworks.com/help/matlab/ref/gammainc.html

PS: You get them as inbuilt functions in math calculators. E.g., GP/Pari has erfc(x) and incgamc(a,x). Online gp/pari calculator in https://pari.math.u-bordeaux.fr/gp.html

Frequency (monobit) test

Purpose: Determine whether the number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence.

Test description: Input is a bit sequence of length $n \geq 100$: $\varepsilon = \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n$.

1. Sum all the bits $S_n = \sum_{i=1}^{n}(2\varepsilon_i - 1)$
2. Compute the test statistic $s_{obs} = \frac{|S_n|}{\sqrt{n}}$
3. Compute the P_{value} = $\mathit{erfc}\left(\frac{s_{obs}}{\sqrt{2}}\right)$

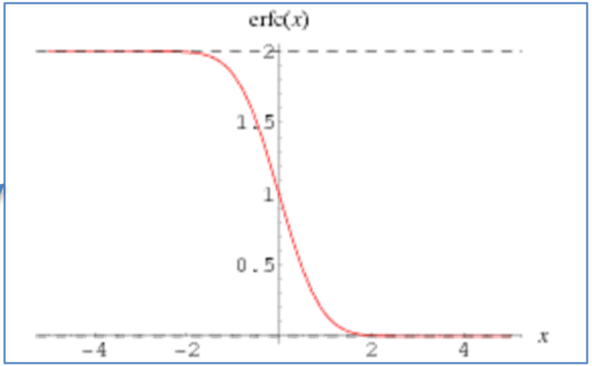
Decision rule: If $P_{value} > \alpha$, then the input sequence is considered as random. Otherwise it is considered as non-random.

Frequency (monobit) test

Purpose: Determine whether the number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence.

Test description: Input is a bit sequence of length $n \geq 100$: $\varepsilon = \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n$.

1. Sum all the bits $S_n = \sum_{i=1}^{n}(2\varepsilon_i - 1)$
2. Compute the test statistic $s_{obs} = \frac{|S_n|}{\sqrt{n}}$
3. Compute the P_{value} = $erfc\left(\frac{s_{obs}}{\sqrt{2}}\right)$



For large S_{obs} (i.e., S_n is large) P_{value} is small. Large S_n happens when number of 0s and 1s are significantly different.

Decision rule: If $P_{value} > \alpha$, then the input sequence is considered as random. Otherwise it is considered as non-random.

Next: Implementing NIST’s tests in HW

Challenges and Simplifications

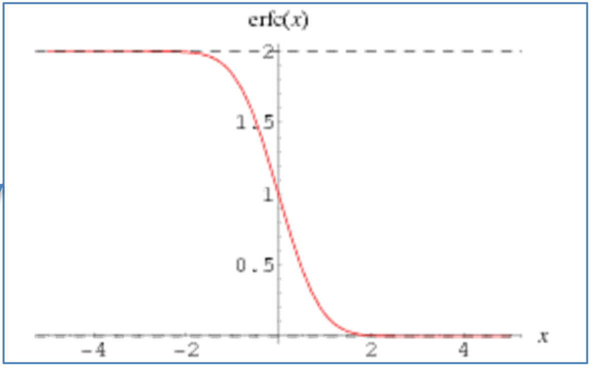
Let’s consider the ‘Frequency Test’ as a case study.

- It is the simplest of all.
- Yet, its HW implementation can be challenging

Recap of the Frequency (monobit) test

Purpose: Determine whether the number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence.

Test description: Input is a bit sequence of length $n \geq 100$: $\varepsilon = \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n$.



1. Sum all the bits $S_n = \sum_{i=1}^{n} (2\varepsilon_i - 1)$

2. Compute the test statistic $s_{obs} = \frac{|S_n|}{\sqrt{n}}$

3. Compute the P_{value} = $erfc\left(\frac{s_{obs}}{\sqrt{2}}\right)$

For large S_{obs} (i.e., S_n is large) P_{value} is small. Large S_n happens when number of 0s and 1s are significantly different.

Decision rule: If $P_{value} > \alpha$, then the input sequence is considered as random. Otherwise it is considered as non-random.

HW building blocks for frequency test (1)

1. Sum all the bits $S_n = \sum_{i=1}^{n} (2\varepsilon_i - 1)$

HW building blocks for frequency test (1)

1. Sum all the bits $S_n = \sum_{i=1}^{n} (2\varepsilon_i - 1)$

This is a simple step. Implemented as a counter.

Bits are input serially

Increment by 1 if Bit = 1
Decrement by 1 if Bit = 0

HW building blocks for frequency test (2)

2. Compute the test statistic $s_{obs} = \frac{|S_n|}{\sqrt{n}}$

Requires
1. A square-root() operation, and
2. A division() by a real number.

Both are expensive operations.
A floating-point arithmetic unit is needed.
Not easy to implement in HW.

HW building blocks for frequency test (3)

3. Compute $P_{value} = erfc\left(\frac{s_{obs}}{\sqrt{2}}\right)$

Requires the erfc() which computes integration

$$erfc(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-u^2} du$$

Much harder to implement in HW than the previous two operations!
Large area and memory requirements.

Can we simplify them so that we can implement in HW?

Note: We are interested in knowing whether

$\alpha < P_{value}$ is true of false.

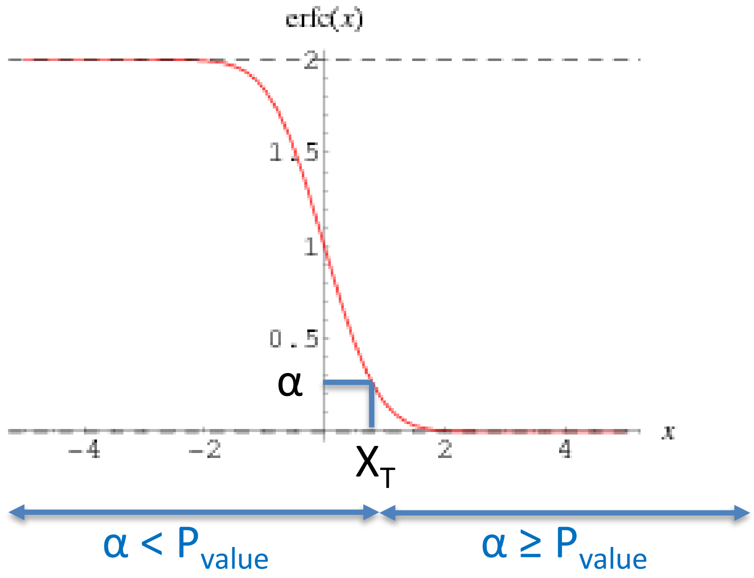
Note: We are interested in knowing whether

$\alpha < P_{value}$ is true of false.

That is:

$$\alpha < erfc\left(\frac{s_{obs}}{\sqrt{2}}\right)$$

When x increases, erfc(x) decreases monotonically.



For a given α there is a threshold point X_T s.t.
for all $x > X_T$ $\alpha \geq \mathrm{erfc}(x)$ (i.e., $\alpha \geq P_{value}$)

Simplification of frequency test (1)

3. Compute $P_{value} = erfc\left(\frac{s_{obs}}{\sqrt{2}}\right)$

No need to compute erfc()

Simplification of step 3:

1. For a given α (=0.01 in our case) precompute X_T
2. Check if $S_{obs} < \sqrt{2}\, X_T$ →

If true, then Pvalue > α and the sequence is random.
If false then the sequence is non-random.

Simplification of frequency test (2)

2. Compute the test statistic $s_{obs} = \frac{|S_n|}{\sqrt{n}}$

Step 2 requires
1. A square-root() operation, and
2. A division() by a real number.

We can avoid them too!

Further simplification:
1. In the previous slide, we were checking the comparison $S_{obs} < \sqrt{2}\, X_T$
2. The equivalent will be checking if $|S_n| < \sqrt{2n}\, X_T$

Simplification of frequency test (2)

2. Compute the test statistic $s_{obs} = \frac{|S_n|}{\sqrt{n}}$

Step 2 requires
1. A square-root() operation, and
2. A division() by a real number.

We can avoid them too!

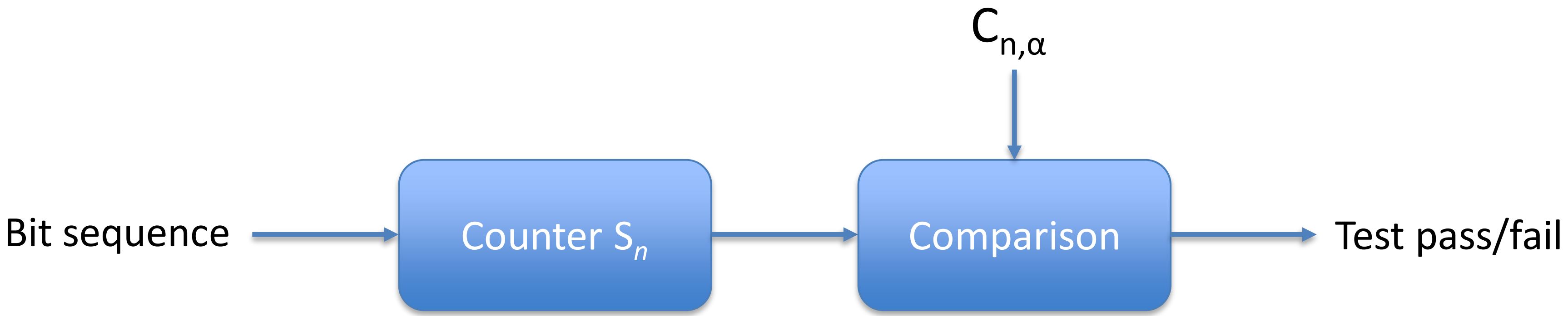
Further simplification:

1. In the previous slide, we were checking the comparison $S_{obs} < \sqrt{2}\ X_T$
2. The equivalent will be checking if $|S_n| < \boxed{\sqrt{2n}\ X_T}$

If *n* is kept constant, then this is a comparison with a constant.

(Note: X_T is also a constant if α is kept fixed)

Simplified frequency test: Summary



Where S_n is the sum of the bits,

and $C_{n,\alpha} = \sqrt{2n}\, X_T$ is a constant for a fixed n and α.

NIST's random number generation tests

The NIST Test Suite is a package of 15 statistical hypothesis tests to test the randomness of arbitrary long binary sequences.

1. Frequency (monobit) test
2. Frequency test within a block
3. Runs test
4. Test for longest-run-of-ones in a block
5. Binary matrix rank test
6. Discrete Fourier transform (spectral) test
7. Non-overlapping template matching test
8. Overlapping template matching test
9. Maurer's 'Universal Statistical' test
10. Linear complexity test
11. Serial test
12. Approximate entropy test
13. Cumulative sums test
14. Random excursions test
15. Random excursions variant test

Frequency test within a block

Purpose: Determine whether the frequency of ones in an *M*-bit block is approximately *M*/2, as would be expected for a truly random sequence.

Test description: Input is a bit sequence of length $n \geq 100$. Block size $M > 0.01n$.

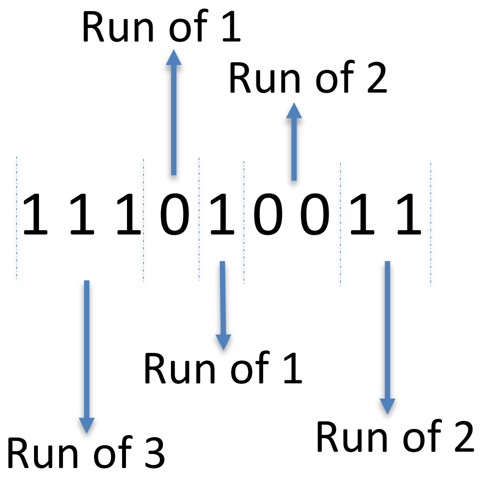
1. Split the input sequence into M-bit non-overlapping $N = \left\lfloor \frac{n}{M} \right\rfloor$ sub-sequences.
2. Determine the proportion π_i of ones in each *M*-bit block $\pi_i = \frac{\sum_{j=1}^{M} \varepsilon_{(i-1)M+j}}{M}$
3. Compute the χ^2 statistic: $\chi^2(obs) = 4M \sum_{i=1}^{N} (\pi_i - ½)^2.$
4. Compute the P_{value} = $\mathbf{igamc}\,(N/2, \chi^2(obs)/2)$

Decision rule: If $P_{value} > \alpha$, then the input sequence is considered as random. Otherwise it is considered as non-random.

Runs test

A ‘run’ is an uninterrupted sequence of identical bits.

E.g.,



Runs test

Purpose: Determine whether the number of runs of 0s and 1s of various lengths is as expected for a random sequence.

Test description: Input is a bit sequence of length $n \geq 100$: $\varepsilon = \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n$

Runs test is applicable only if the frequency test is passed.

1. Compute the proportion π of ones in the input sequence: $\pi = \dfrac{\sum_j \varepsilon_j}{n}$
2. Compute the test statistic: $V_n(obs) = \sum_{k=1}^{n-1} r(k) + 1$ where

   $r(k)=0$ If $\varepsilon_k = \varepsilon_{k+1}$, and $r(k)=1$ otherwise.

3. Compute the P_{value} $= erfc\left(\dfrac{|V_n(obs) - 2n\pi(1-\pi)|}{2\sqrt{2n}\pi(1-\pi)}\right)$

Decision rule: Same as the previous tests.

NIST’s random number generation tests

1. Frequency (monobit) test
2. Frequency test within a block
3. Runs test
4. Test for longest-run-of-ones in a block
5. Binary matrix rank test
6. Discrete Fourier transform (spectral) test
7. Non-overlapping template matching test
8. Overlapping template matching test
9. Maurer’s ‘Universal Statistical’ test
10. Linear complexity test
11. Serial test
12. Approximate entropy test
13. Cumulative sums test
14. Random excursions test
15. Random excursions variant test

Non-overlapping template matching test

Purpose: Detect if there are too many occurrences of a given non-periodic pattern in the input binary sequence.

$$\varepsilon = \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n.$$

1. Input string is split into blocks of size *M*-bits. Thus, there are *N* = *n*/*M* blocks.

Example: Let $\varepsilon = 10100100101110010110$, of length *n* = 20. Let *M*=10, and *N*=2.

Non-overlapping template matching test

Purpose: Detect if there are too many occurrences of a given non-periodic pattern in the input binary sequence.

$$\varepsilon = \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n.$$

1. Input string is split into blocks of size *M*-bits. Thus, there are *N* = *n*/*M* blocks.

Example: Let $\varepsilon = \underbrace{1010010010}_{\text{Block 1}}\underbrace{1110010110}_{\text{Block 2}}$, of length *n* = 20. Let *M*=10, and *N*=2.

2. For a given target pattern *B*, count the number of appearances of *B* in each block.

Example: Let *B* = *001*.

(See next slide)

Non-overlapping template matching test (2)

The first block = *1 0 1 0 0 1 0 0 1 0*

Specified string B = *0 0 1*

Initialize counter for the number of matches W1 = 0

Non-overlapping template matching test (2)

The first block = $\underbrace{1\ 0\ 1}\ 0\ 0\ 1\ 0\ 0\ 1\ 0$

Specified string B = $0\ 0\ 1$

No match. Hence slide window by one bit.

Counter for the number of matches W1 = 0

Non-overlapping template matching test (2)

The first block = $\;1\;\underbrace{0\;1\;0}\;0\;1\;0\;0\;1\;0$

Specified string B = *0 0 1*

No match. Hence slide window by one bit.

Counter for the number of matches W1 = 0

Non-overlapping template matching test (2)

The first block = *1 0 1 0 0 1 0 0 1 0*

Specified string B = *0 0 1*

No match. Hence slide window by one bit.

Counter for the number of matches W1 = 0

Non-overlapping template matching test (2)

The first block = *1 0 1 0 0 1 0 0 1 0*

Specified string B = *0 0 1*

Match! Slide window by the length of B, i.e., 3 bits.

Increment counter for the number of matches W1 = 0 + 1

Non-overlapping template matching test (2)

The first block = $1\ 0\ 1\ 0\ 0\ 1\ \underbrace{0\ 0\ 1}\ 0$

Specified string B = $0\ 0\ 1$

Another match! Stop sliding as there are insufficient leftover bits.

Increment counter for the number of matches W1 = W1 + 1 = 2

Next, repeat this for all the M-bit blocks and compute W2, W3, …

Non-overlapping template matching test (3)

Test description:

1. Using the previous method, compute W_1, W_2, …, W_N for all the N blocks

2. Compute the theoretical mean μ and variance σ^2 as

$$\mu = (M-m+1)/2^m \qquad \sigma^2 = M\left(\frac{1}{2^m} - \frac{2m-1}{2^{2m}}\right)$$

   where M is the size of each block, and m is the size of the specified pattern B. (In the previous example M = 10 and m = 3)

3. Compute the test statistic $\chi^2(obs) = \sum_{j=1}^{N} \frac{(W_j - \mu)^2}{\sigma^2}$

4. Compute the $P_{value} = \mathbf{igamc}\left(\frac{N}{2}, \frac{\chi^2(obs)}{2}\right)$

Decision rule: Same as the previous tests.

NIST's random number generation tests

1. Frequency (monobit) test
2. Frequency test within a block
3. Runs test
4. Test for longest-run-of-ones in a block
5. Binary matrix rank test
6. Discrete Fourier transform (spectral) test
7. Non-overlapping template matching test
8. Overlapping template matching test
9. Maurer's 'Universal Statistical' test
10. Linear complexity test
11. Serial test
12. Approximate entropy test
13. Cumulative sums test
14. Random excursions test
15. Random excursions variant test

Overlapping template matching test

Somewhat similar to the previous non-overlapping template matching test.

$$\varepsilon = \varepsilon_1,\ \varepsilon_2,\ \ldots,\ \varepsilon_n$$

1. Input string is split into blocks of size *M*-bits. Thus, there are *N* = *n/M* blocks.

E.g., ε = 1011101111 0010110100 0111001011 1011111000 0101101001

$$\underbrace{1011101111}_{\text{Block 1}}\ \underbrace{0010110100}_{\text{Block 2}}\ \underbrace{0111001011}_{\text{Block 3}}\ \underbrace{1011111000}_{\text{Block 4}}\ \underbrace{0101101001}_{\text{Block 5}}$$

Where sequence length n = 50, block length M = 10, number of blocks N = n/M = 5

Overlapping template matching test (1)

2. An array of 6 counters is initialized to all 0s.

$v_0 = 0$	$v_1 = 0$	$v_2 = 0$	$v_3 = 0$	$v_4 = 0$	$v_5 = 0$

This counters will be incremented during the template matching operation using the following rule.

a. V_0 is incremented if the M-bit block contains 0 occurrence of B
b. V_1 is incremented if the M-bit block contains only 1 occurrence of B
c. V_2 is incremented if the M-bit block contains only 2 occurrences of B
d. V_3 is incremented if the M-bit block contains only 3 occurrences of B
e. V_4 is incremented if the M-bit block contains only 4 occurrences of B
f. V_5 is incremented if the M-bit block contains ≥ 5 occurrences of B

Overlapping template matching test (1)

Example of counter update.

Let's consider the 1st block = 1 0 1 1 1 0 1 1 1 1

And let the specified pattern be B = '11'.

Number of matches within the block = 0.

$v_0 = 0$	$v_1 = 0$	$v_2 = 0$	$v_3 = 0$	$v_4 = 0$	$v_5 = 0$

Counter before template matching starts in the block.

Overlapping template matching test (1)

Example of counter update.

Let's consider the 1st block = $\underbrace{1\ 0}\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 1$

No match with B = '11'.
Always slide by 1 bit.

Number of matches within the block = 0.

$v_0 = 0$	$v_1 = 0$	$v_2 = 0$	$v_3 = 0$	$v_4 = 0$	$v_5 = 0$

V[ ] counter doesn't change during the process.

Overlapping template matching test (1)

Example of counter update.

Let's consider the 1st block = 1 0 1 1 1 0 1 1 1 1

No match with B = '11'.

Always slide by 1 bit.

Number of matches within the block = 0.

$v_0 = 0$	$v_1 = 0$	$v_2 = 0$	$v_3 = 0$	$v_4 = 0$	$v_5 = 0$

V[ ] counter doesn't change during the process.

Overlapping template matching test (1)

Example of counter update.

Let's consider the 1st block = 1 0 1 1 1 0 1 1 1 1

Match with B = '11'.
Always slide by 1 bit. (This was different in non-overlap. Test)

Number of matches within the block = 1.

This counter increments

$v_0 = 0$	$v_1 = 0$	$v_2 = 0$	$v_3 = 0$	$v_4 = 0$	$v_5 = 0$

V[ ] counter doesn't change during the process.

Overlapping template matching test (1)

Example of counter update.

Let's consider the 1st block = 1 0 1 $\underbrace{1\ 1}$ 0 1 1 1 1

Another match with B = '11'.
Always slide by 1 bit.

$\boxed{\text{Number of matches within the block = 2.}}$

This counter increments

$v_0 = 0$	$v_1 = 0$	$v_2 = 0$	$v_3 = 0$	$v_4 = 0$	$v_5 = 0$

V[ ] counter doesn't change during the process.

Overlapping template matching test (1)

Example of counter update.

Let's consider the 1st block = 1 0 1 1 1 0 1 1 1 1

No match with B = '11'.
Always slide by 1 bit.

Number of matches within the block = 2.

$v_0 = 0$	$v_1 = 0$	$v_2 = 0$	$v_3 = 0$	$v_4 = 0$	$v_5 = 0$

V[ ] counter doesn't change during the process.

Overlapping template matching test (1)

Example of counter update.

Let's consider the 1st block = 1 0 1 1 1 0 1 1 1 1

No match with B = '11'.
Always slide by 1 bit.

Number of matches within the block = 2.

$v_0 = 0$	$v_1 = 0$	$v_2 = 0$	$v_3 = 0$	$v_4 = 0$	$v_5 = 0$

V[ ] counter doesn't change during the process.

Overlapping template matching test (1)

Example of counter update.

Let's consider the 1st block = 1 0 1 1 1 0 1 1 1 1

Match with B = '11'.
Always slide by 1 bit.

Number of matches within the block = 3.

This counter increments

$v_0 = 0$	$v_1 = 0$	$v_2 = 0$	$v_3 = 0$	$v_4 = 0$	$v_5 = 0$

V[ ] counter doesn't change during the process.

Overlapping template matching test (1)

Example of counter update.

Let's consider the 1st block = $\quad 1\ 0\ 1\ 1\ 1\ 0\ 1\ \underbrace{1\ 1}\ 1$

Another match with B = '11'.
Always slide by 1 bit.

$\boxed{\text{Number of matches within the block = 4.}}$

This counter increments

$v_0 = 0$	$v_1 = 0$	$v_2 = 0$	$v_3 = 0$	$v_4 = 0$	$v_5 = 0$

V[ ] counter doesn't change during the process.

Overlapping template matching test (1)

Example of counter update.

Let's consider the 1st block = 1 0 1 1 1 0 1 1 $\underbrace{1\ 1}$

Another match with B = '11'.
Always slide by 1 bit.

Number of matches within the block = 5.

This counter increments

$v_0 = 0$	$v_1 = 0$	$v_2 = 0$	$v_3 = 0$	$v_4 = 0$	$v_5 = 0$

V[ ] counter doesn't change during the process.

Overlapping template matching test (1)

Example of counter update.

Let's consider the 1st block = 1 0 1 1 1 0 1 1 1 1

Template matching within this block has finished.

As the number of matches within the block is ≥ 5, increment V_5 by 1.

Number of matches within the block = 5.

$v_0 = 0$	$v_1 = 0$	$v_2 = 0$	$v_3 = 0$	$v_4 = 0$	$\mathbf{v_5 = 1}$

Overlapping template matching test (2)

For the 2nd block = 0 0 1 0 1 1 0 1 0 0

Number of matches within the 2nd block = 1.

Hence increment V_1 by 1.

$v_0 = 0$	$\mathbf{v_1 = 1}$	$v_2 = 0$	$v_3 = 0$	$v_4 = 0$	$v_5 = 1$

Continue in the same manner for all the remaining blocks.

Overlapping template matching test (3)

3. Compute $$\chi^2(obs) = \sum_{i=0}^{5} \frac{(v_i - N\pi_i)^2}{N\pi_i}$$

   where $\pi_0, \pi_1, \ldots, \pi_5$ are constants specified in Section 3.8 of [NIST]. (They dependent on the block size M and template size m).

4. Compute $P_{value} = \mathbf{igamc}\left(\frac{5}{2}, \frac{\chi^2(obs)}{2}\right)$

Decision rule: Same as the previous tests, i.e., if $P_{value} > \alpha$, then the input sequence is considered as random. Otherwise it is considered as non-random.

NIST’s random number generation tests

1. Frequency (monobit) test
2. Frequency test within a block
3. Runs test
4. Test for longest-run-of-ones in a block
5. Binary matrix rank test
6. Discrete Fourier transform (spectral) test
7. Non-overlapping template matching test
8. Overlapping template matching test
9. Maurer’s ‘Universal Statistical’ test
10. Linear complexity test
11. Serial test
12. Approximate entropy test
13. Cumulative sums test
14. Random excursions test
15. Random excursions variant test

The remaining statistical tests will not be covered in the lecture.

The specification document from NIST describes all the 15 tests in great detail and with examples.

"A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications" by NIST. Date Published: April 2010.
https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final

In most cases, we will **use these tests in 'black box' manner** to draw inference on the quality of generated randomness.

Demo: Using NIST’s test suit

Homework thoughts

How could you simplify the other tests so that they are lightweight and easy to implement on HW platforms?