Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Matematica

Testing a Random Number Generator: formal properties and automotive application

Tesi di Laurea Magistrale in Crittografia e Probabilità

Relatore:
Chiar.ma Prof.ssa
GIOVANNA
CITTI

Correlatore:
Chiar.ma Dott.ssa
ELISA
BRAGAGLIA

Presentata da:
FEDERICO
MATTIOLI

III Sessione

Anno Accademico 2017/2018

Introduzione

Lo scopo principale di questa tesi è lo studio di processi casuali in campo automotive. I veicoli moderni, infatti, sono gestiti da centraline (Electronic Control Units: ECUs) che contengono la maggior parte della struttura software del veicolo e sono responsabili della sicurezza dello stesso nei confronti di attacchi informatici. La maggior parte degli algoritmi crittografici è basata sulla generazione di numeri casuali: è quindi necessario che le stringhe generate siano effettivamente casuali. Il metodo di valutazione più riconosciuto e standardizzato, proposto dal National Institute of Standards and Technology (NIST), consiste nell'analizzare sequenze di numeri, costruite dal generatore, attraverso una serie di test statistici.

I capitoli 2 e 3 costituiscono il nucleo dell'elaborato: qui vengono date le definizioni di processi casuali da un punto di vista probabilistico e con riferimento alla nozione di entropia. Successivamente, il lavoro si concentra sullo studio della test suite per la validazione di Generatori di Numeri Casuali (RNG) proposta dal NIST. La suite consiste in 15 test, ognuno dei quali analizza le sequenze da una differente prospettiva. La solidità dei test cresce dai primi test (che sono caratterizzati da un bassissimo costo computazionale, permettendo di scartare gli RNG meno efficienti) agli ultimi, che studiano proprietà più sofisticate dei processi casuali.

I primi test sono basati su proprietà molto semplici delle sequenze casuali, come la probabilità di zero e uno all'interno della sequenza o all'interno di sotto-stringhe (*Frequency test* e *Frequency test within a block*), o la probabilità di successioni di bit uguali lungo la sequenza o all'interno di sotto-stringhe (*Runs test* e *Runs test within a block*).

Se una sequenza passa questi primi test, viene analizzata con altre classi di test, basati su strumenti matematici più sofisticati come ad esempio:

- trasformata di Fourier: il rispettivo test (*Spectral test*) è in grado di rilevare comportamenti periodici all'interno della sequenza;

- entropia: le sequenze casuali sono caratterizzate da alti valori entropici, strettamente connessi con la comprimibilità della sequenza (*Maurer's universal statistical*

test e *Approximate entropy test*);

- complessità algoritmica: le sequenza casuali sono sufficientemente complesse da non poter essere costruite da semplici algoritmi (*Linear complexity test* e *Serial test*);

- passeggiata aleatoria: la stringa binaria viene identificata come una passeggiata aleatoria di 1 e -1. In quest'ottica le somme parziali dei primi m valori della sequenza devono essere vicini allo zero (*Cumulative sums test*) e la sequenza deve visitare ogni stato con la stessa frequenza di una passeggiata aleatoria (*Random excursion test* and *Random excursion variant test*).

Dal momento che ognuno di questi analizza un diverso comportamento, un unico test non è sufficiente per fornire un adeguato responso riguardo la sequenza in oggetto inoltre, dal momento che lo scopo è quello di validare l'RNG, una sola sequenza non basta per affermare se le sequenze siano casuali o meno, ma sarà necessario testare un numero sufficientemente grande di sequenze attraverso un'adeguata quantità di test.

Con lo scopo di costruire un metodo per valutare, primi tra gli altri, gli RNG contenuti all'interno delle centraline, dopo aver analizzato e validato ognuno dei test proposti, cercheremo di implementare l'intero schema di validazione in CANoe, un software che permette di programmare ed interagire con le ECU, utilizzando, come step intermedi, MatLab e Simulink.

Questo lavoro non sarebbe stato possibile senza l'Azienda *Magneti Marelli*, che mi ha permesso di svolgere un tirocinio di un anno presso la sede di Bologna, supportato dal *Cyber Security Systems Architect* che mi ha insegnato, tra le altre cose, come è costruita la struttura software dei moderni autoveicoli, cyber security in primis, nonché l'utilizzo dei sistemi che verranno analizzati in questa tesi.

Introduction

The main scope of this dissertation is to study random processes in cyber security of automotive environment. Indeed modern vehicles are managed by the Electronic Control Units (ECUs), which contains most of the software structure of the vehicle and are responsible of its security against cyber attacks. Most cryptographical algorithms are based on generation of random numbers and it is necessary that the generated strings are truly random. The most standardized evaluation method consists in analyzing sequences of numbers, built by the generator, through a set of statistical tests, provided by the National Institute of Standards and Technology (NIST).

The core of the thesis are chapters 2 and 3, where we gave the definition of random process from a probabilistic and entropic point of view. Then we study the suite proposed by NIST to validate Random Number Generators. The suite consists of 15 tests, each one inspecting the sequence from a different perspective. The strength of tests increases from first tests (which have extremely low computational cost and allow to rapidly discard not efficient RNG) to the last ones, which study more sophisticated properties of the random process.

The first tests are based on very simple properties of random sequences as for example the probability of zeros and ones within the sequence or within any sub-block of the sequence (*Frequency test* and *Frequency test within a block*), or the probability of occurrence of k identical bits within the sequence or any sub-block of it (*Runs test* and *Runs test within a block*). If a sequence passes these first tests, it is submitted to other classes of tests, based on more sophisticated mathematical instruments:

- discrete Fourier transform: these tests are able to detect periodic features within the sequence which would indicate deviation from randomness (*Spectral test*);
- notion of entropy, which has high values in random sequences, and is strictly connected with non compressibility of the sequence (*Maurer's universal statistical test* and *Approximate entropy test*);
- algorithmic complexity: the high complexity of random sequences doesn't permit to build them by simple algorithms (*Linear complexity test* and *Serial test*);

- random walk properties: the binary string is identified with a random walk which attains values 1, -1. Hence the partial sum of the first m values of the sequence has to be near zero (*Cumulative sums test*) and the sequence has to visit each state with the same frequence of a random walk (*Random excursion test* and *Random excursion variant test*).

Since each of those analyzes a different behavior, a unique test is not adequate to give a response concerning the analyzed sequence and, given the fact that the final goal is to evaluate the RNG, a single sequence is not enough to say whether or not generated sequences are random, but it will be necessary to test a great number of sequences, by a commensurate number of tests.

With the purpose to build a method to evaluate, among all, RNGs embedded into ECUs, after studying and validating each proposed test, we will try to implement the whole validation scheme into CANoe, a software which permits to program and interact with ECUs, using, as intermediate steps, MatLab and Simulink.

This work wouldn't be possible without the *Magneti Marelli* Company, which allowed me to carry out a one year traineeship at its Bologna office, supported by the *Cyber Security Systems Architect* which taught me, among other things, how the software structure of modern vehicles is built, cyber security in the first place, as well as the use of the systems that will be analyzed along this thesis.

Contents

Chapter 1

Random Number Generators in automotive cyber security

This chapter is intended to give the reader a main idea of the importance of Random Number Generators for cryptographic purposes, and in particular for automotive cyber security: an overview of the design of the structure of modern vehicles is provided together with examples of usage of random numbers in cryptography. After that, an explanation of Random Number Generators types is proposed, in order to give the reader the framework of this work.

1.1 Automotive cyber security

The structure of modern vehicle allows designers to embed many functions through software, to make the vehicle more and more efficient and less subject to mechanical breakages. This allows to implement many features but, on the other hand, requires that the system is protected not only from mechanical threats, but also from hacking. This section is aimed to describe the design of vehicles communication system, together with possible point by which an attacker could interact with it.

1.1.1 The ECUs infrastructure

Modern automobiles are no longer totally mechanical systems, since the largest part of actions and communications, between the user and the car or within the car itself, are managed by electronic devices (e.g. if the driver push the brake pedal, the mechanical action is registered and electronically sent to the brakes, where the information is processed and executed by the physical system: that happens not only for brakes, but for the most part of the behavior of the car). To do that, every part of the vehicle which has to perform an action (such as brakes, lightening, steer and the entertaining system)

or to register values (e.g. emissions, consumptions and temperatures) is provided by an electronic system which is responsible to manage it, called Electronic Control Unit (ECU).

Main examples of ECUs[1] are:

- Engine Control Module (ECM): controls the engine by determining the amount of fuel to use, the time of ignition and many other parameters concerning the engine;
- Electronic Brake Control Module (EBCM): controls the brake system and the Antilock Brake System (ABS), formed by pumps and valves, which prevents the brakes to lock in situations of possible sliding;
- Transmission Control Module (TCM): responsible of the transmission and of the gear changing;
- Body Control Module (BCM): controls many functions of the cabin and often works as a firewall between messages from other ECUs;
- Heating Ventilation, Air Conditioning (HVAC): responsible of cabin environments, such as the temperature.

ECUs don't operate separately, but they are connected each another to share information, data and parameters which are necessary for the correct conduct of the vehicle. The connections and the internal communications network are usually complaint to the Controller Area Network standard, the CAN-bus, introduced in 1993 by ISO 11898[2]. The particularity of the CAN-bus is that it is a broadcast communication system: a single cable is needed to connect a network of ECUs, which are able to see each message other ECUs send. The communication via CAN allows to reduce cost, complexity and gain in efficiency, compared to the corresponding traditional wiring. Figure 1.1 provides an overview of the ECUs system and of the connections between them.

1.1.2 Vehicle entry points and hacking

The multiple connected structure of ECUs permits to make the system more efficient and simpler on one hand, but it also exposes the vehicle, the driver and the manufacturer to new threats that have to be taken into consideration and prevented. It become necessary, then, to embed processes to protect the system. To clarify this problem, we

[1] Most of ECUs are installed into each vehicle, however different manufacturers (OEMs) could decide to embed different types in their vehicles.

[2] Other protocols are the Local Interconnect Network (LIN) and the Ethernet, however they are less used in automotive due to the low speed of the first one and to the high cost of the second.

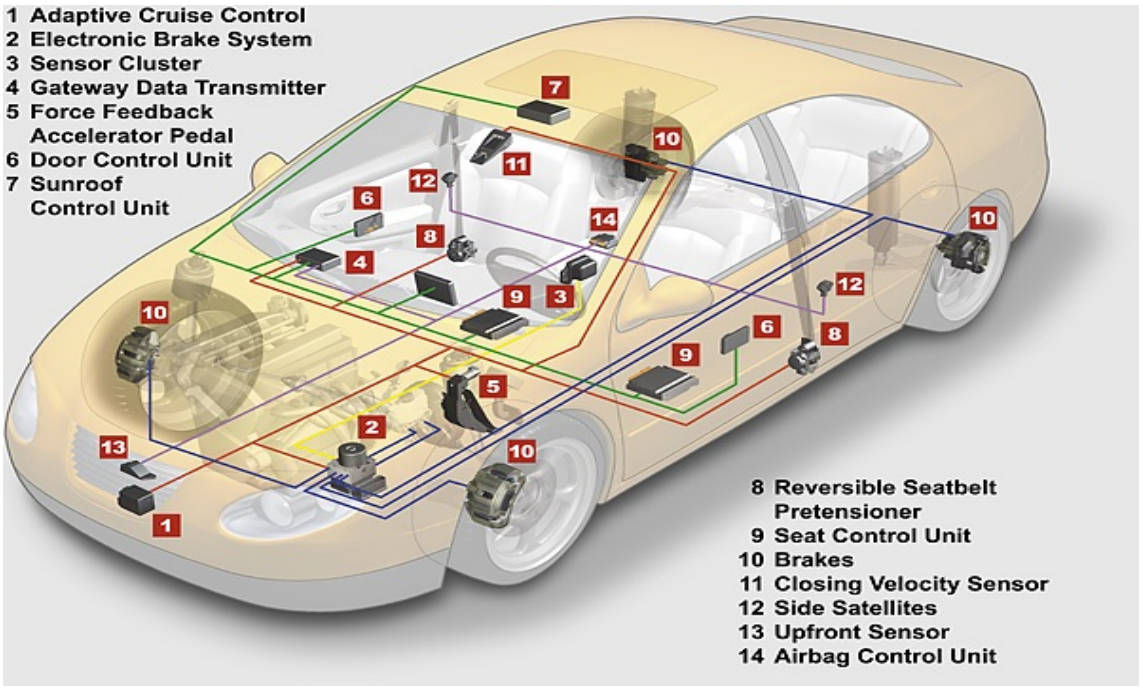

Figure 1.1: ECUs system and CAN-bus connections *Credits to* `www.silicon.it`.

shall make some examples of how a malicious attacker could attach the car system.

We can distinguish between two sets of entry points: physical entry points and wireless connection. Physical entry points are "doors" as for example:

- the OBD-II port: which allows to exchange information by and to the whole vehicle through the CAN-bus and to diagnose malfunctions without physically checking all components or to modify ECUs software. To do that a cable has to be inserted into the port and connected to a device, such as a computer with a dedicated tool[3]. OBD-II port is often situated under the steering wheel: figure 1.2 shows the appearance of this device. It is clear that an attacker who could access this port could seriously put at risk driver's health by injecting tampered traffic in the internal network.

- the infotainment system: the interface by which the user can use the radio, play music, control wireless connections or access the GPS. The infotainment system is coupled with an Operating System (e.g. Windows, Linux, Android..) that could be subject to wireless or wired modifications in order to affect vehicle behavior.

[3]We will present and use one of them in chapter 5: the CANoe software [70].

- CD player, USB, phone docking port: mainly used to interact with the infotainment system, could be a door to enter the system and inject data.

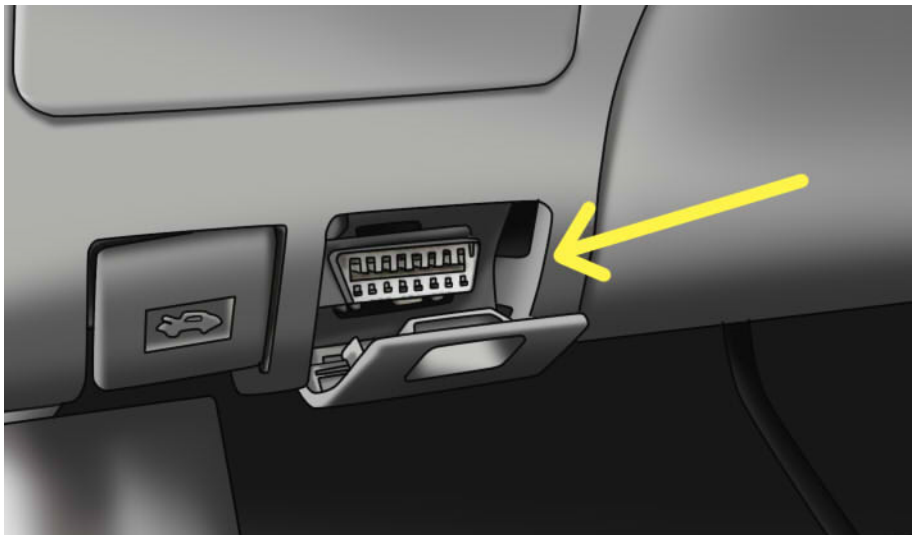

Figure 1.2: The OBD-II port. *Credits to* `www.yourmechanic.com`.

Modern vehicles are also provided with many wireless connection which could be used by an attacker to compromise the system. Examples of wireless entry points are:

- bluetooth: used to pair user's devices, such as a smartphone, to reproduce music or to use an eventual car speakerphone;
- Radio Frequency Identification (RFID) and Remote Keyless Entry (RKE): devices which, pairing radio signals, permit to remotely open and turn on the vehicle;
- Wi-Fi connection: used, mainly, to communicate between nearby vehicles which support it;
- Global Position System (GPS);
- Digital Audio Broadcasting (DAB);
- Mobile connection (3G,4G): used to connect to OEM's servers, e.g. to remotely update software or to transmit vehicle's data.

We can also classify these connection among short-range (bluetooth), mid-range (RFID and Wi-Fi) and long-range connections (GPS, Internet, DAB and Mobile connection).

All of these entry points could be used, in different ways, to hack the car systems with different techniques. The table below shows main types of attack, coupled with a brief description of them and possible consequences.

Hacking technique	Description	Consequence
Flooding	Requesting of some service(s) with a larger frequency than nominal condition	DoS[4] or activation of safe state
Starvation	Trying to repetitively self authenticate using spoofed (i.e. not working) credentials	DoS or activation of safe state
Memory Injection	Injecting malicious code to make a target ECU execute it and corrupt it	Alter target ECU functionality or take control of it
Spoofing	Pretending to be an authorized system	Take control of the target system

Example 1.1.1 (Spoofing attack)**.**
During an extended diagnostic session, gears can be changed using a CAN message (e.g. by the mechanic to check the transmission) if the speed of the vehicle is 0 km/h.
A malicious attacker who can access the CAN-bus can do the following to change gears:

1. overwriting speed message, read by gear ECU, with the value 0;
2. sending the tester presence message to the gear ECU;
3. sending the request to the gear ECU to open an extended diagnostic session;
4. authenticating itself to the gear ECU, calculating the key by reversing the seed and key algorithm, used for authentication;
5. sending the gear change request to the gear ECU.

That could be potentially done even while the car is moving, and consequences could be extremely serious.

The example shows one of possible hacking cases which can occur if there is no or low security. Consequences could be different (and could have different degrees of dangerousness): from the changing of the radio station to the shutdown of the engine, passing trough the disabling of the power steering. It becomes indispensable, then, to provide each part of the vehicle with some kind of security, depending on the vulnerability of it.

[4]Denial of Service: due to the huge amount of requests, the systems doesn't manage to fulfill its standard tasks.

In the order form the bigger to the smaller, it is necessary to take into account the security of the whole system, of the vehicle, of ECUs and of the Software each ECU carries and uses: in this dissertation we will focus on the particular aspects which concerns the use of random numbers.

1.2 The importance of randomness in cryptography

Modern cryptography is based on public algorithms and functions (called *cryptographic primitives*): its whole strength is based on the key and on the unpredictability of some values used for encryption, it' here that randomness plays a fundamental role.

The relevance of randomness became more clear if we think that most of modern cryptography algorithms and protocols are designed around the *Kerckhoff principle*, which states that a cryptographic system has to be secure even though everything about the system is known, except for the key[5]. This means that, if we assume that the cryptographic structure is well designed, we should measure the strength of the system by the number of key an attacker has to guess before he can break it, which also implies that the security of the algorithm dangerously decreases if an enemy is able to be aware of one or more bits of the key.

Random numbers are mainly required in the following cases:

- As cryptographic key in both symmetric and asymmetric algorithm: random numbers are used as values which define the transformation of the unencrypted information (*plaintext*) into the encrypted one (*ciphertext*), and vice versa. In modern cryptography, indeed, encryption and decryption algorithms are on public domain, as well as the ciphertexts sender and receiver exchange; the security of the whole process depends, then, on the unguessability of the key: that's why this must depend on random values. Most famous examples of algorithms where the key is generated by random numbers are AES (symmetric cipher) and RSA (asymmetric cipher).

- In symmetric ciphers (especially stream and block chipers), random numbers are used as initialization vector to make different encryptions, even maintaining the same key which, in this case, doesn't need to be changed each message. Anyway, maintaining the same key for a long time could produce a deterioration of security, so that this modality is used only in the case of a brief-time communication.

[5]This principle is also known as Shannon's maxim since C. Shannon has independently formulated it as "the enemy knows the system".

- Random numbers are used as *nonces* (number used once) and challenges, to ensure freshness of messages and authentication (which can be mutual or unilateral) such as ElGamal, Goldwasser-Micali and McEliece ciphers.
- Digital signature algorithms also need random numbers, to ensure authentication and non-repudiation, without exposing the message itself to the signer (this type of signatures are called *blind signature schemes*, shortly BSS). In this case the message is made unintelligible (blinded) through a random number, know only by the author of the message; the blinded message is sent to the entity which as to sign it with is private key, so that no private information are known to the signer. Typical examples are RSA BSS, Elliptic Curve BSS and ElGamal BSS.
- As cryptographic salts, such as as an extra argument used to generate new password, starting from a single master key which remains the same all the time. This is typically done in the case the starting key is particularly guessable or to create key and password that cannot be renegotiated each time.
- Padding is another use of random numbers: some ciphers (particularly block ciphers) require fixed-length messages. In the case the plaintext message doesn't reach the required length, blocks has to be filled with something in order to encrypt the text and, obviously, removed by the receiver. Padding can be done using fixed values (for example with zeros or by 0x80 followed by zeros), but that could result in a leak of information about the message (i.e. an attacker could know something about the message, by analyzing the ciphertext): that could be avoided by padding using random values. Some examples are Optimal Asymmetric Encryption Padding (OAEP) and Probabilistic Signature-Encryption Padding (PSEP).

It is clear that the use of not enough random values, or worse of deterministic ones, to perform these schemes could permit an attacker to break the encryption and find out what it hides (plaintext), to impersonate someone else or to access to other people's personal information: that could turn out in loss of privacy, theft of money or, in the worst case as in automotive's field, on put at risk the health of someone else.

We give the following, simple, examples to give the reader a basic idea of how random numbers are used.

Example 1.2.1 (The Diffie-Hellman protocol)**.**
The Diffie-Hellman key exchange is a cryptographic protocol by which two interlocutors (Alice and Bob in the following, as traditionally) can establish a secret shared key over a public (insecure) channel without the need to know one another or to meet. The shared key, produced by this protocol, could be used to encrypt successive communications throughout symmetric cryptography schemes.

Let g be a primitive root modulo p, where p is prime. The exchange between Alice (A) and Bob (B) proceeds as follows:

A: chooses a random number a, computes $A = g^a \mod p$ and sends (g, p, A) to Bob;

B: chooses a random number b, computes $B = g^b \mod p$ and sends it to Alice;

A: computes $K_A = B^a \mod p$;

B: computes $K_B = A^b \mod p$.

Alice and Bob, now, have the same key $K = K_A = K_B$, since $B^a \mod p = A^b \mod p$.

Let's suppose the attacker could intercept the whole speech: he knows A and B, but couldn't know the key K, since he doesn't know random numbers a and b. Theoretically he could compute, by the discrete logarithm, a and b but that is considered too computationally onerous to be dangerous. If the attacker could guess a or b, for example if a and b aren't randomly generated, he could impersonate Alice or Bob and see the following encrypted communication or substitute itself to one of the two.

Example 1.2.2 (One-Time Pad (OTP))**.**
Vernam Cipher (often called OTP) is the only cryptographic system which has been proven (by C. Shannon in [66]) to be secure, since the ciphertext gives absolutely no information about the plaintext (this property is said to be *perfect secrecy*): for this reason the cipher has been called "the perfect cipher".
To encrypt using OTP, one has to generate an infinite number of random sequences of each possible length and both the sender and the receiver (and no one else) has to own those keys.
Given the plaintext $x_1, \ldots, x_n$, a random sequence of the same length has to be chosen as key $k_1, \ldots, k_n$: the ciphertext is created by an element by element sum $y_i = x_i + k_i \ \forall i$.
The receiver has to know which is the random sequence the sender used and simply decrypt by element by element subtraction.
Each use, the used key (i.e. the random sequence) has to be destroyed and never used again, in order to maintain the perfect secrecy of the cipher.

Notwithstanding the theoretical perfect security of this cipher, the necessity to share a potentially non-finite number of key (which have to remain secret over time) and to know which is used each time, the effective practice usability of OTP is very low in modern cryptography, where the amount of information to encrypt is huge[6].

[6] Anyway, OTP has been used since the early 1900s and registered an exploit in the World War II, where many nations used OTP systems to secure their sensitive traffic. [34] provides an interesting reading on these facts, as long as the history of cryptography from its beginning to the time of writing (1967).

Example 1.2.3 (Digital Signature)**.**
The digital signature is a scheme by which a receiver could verify that the message had been created by a known sender and that it wasn't modified by someone else, then it provides *authentication* and *integrity* of the message. Moreover, signing a message, the sender couldn't deny to have sent the message anymore, so that digital signature also produces *non-repudiation*.

The most classical, and standard since 1994, way to provide a digital signature is the Digital Signature Algorithm (DSA), described here.
To perform a digital signature the triple (p, q, g) is on public domain where p and q are prime numbers, $g = h^z \mod p$ where z is such that $p = qz + 1$ and h is such that $h^z \mod p > 1$.
Moreover, the signer generates a random number x, which will be his private key, and computes $y = g^x \mod p$ to form the public key. Let H be an hash function (for the DSA algorithm a SHA function is used).

The signature process for the message m proceed as follows:

- a random value k is chosen, with the restriction that $1 < k < q$;
- $r = (g^k \mod p) \mod q$;
- $s = (k^{-1}(H(m) + x \cdot r)) \mod q$;
- the signature of the message m consists on the couple (r, s).

To verify the sign one has to perform the following steps:

- $w = s^{-1} \mod q$;
- $u_1 = H(m) \cdot w \mod q$;
- $u_2 = r \cdot w \mod q$;
- $v = (g^{u_1} y^{u_2} \mod p) \mod q$.

The signature is considered to be valid if $v = r$.

1.2.1 Employment of randomness in automotive

The examples above show some of common uses of randomness in cryptography. In this section we would make some more examples, related to the automotive case, in order to make the usage (and the importance) of randomness more clear, and specific to our study case.

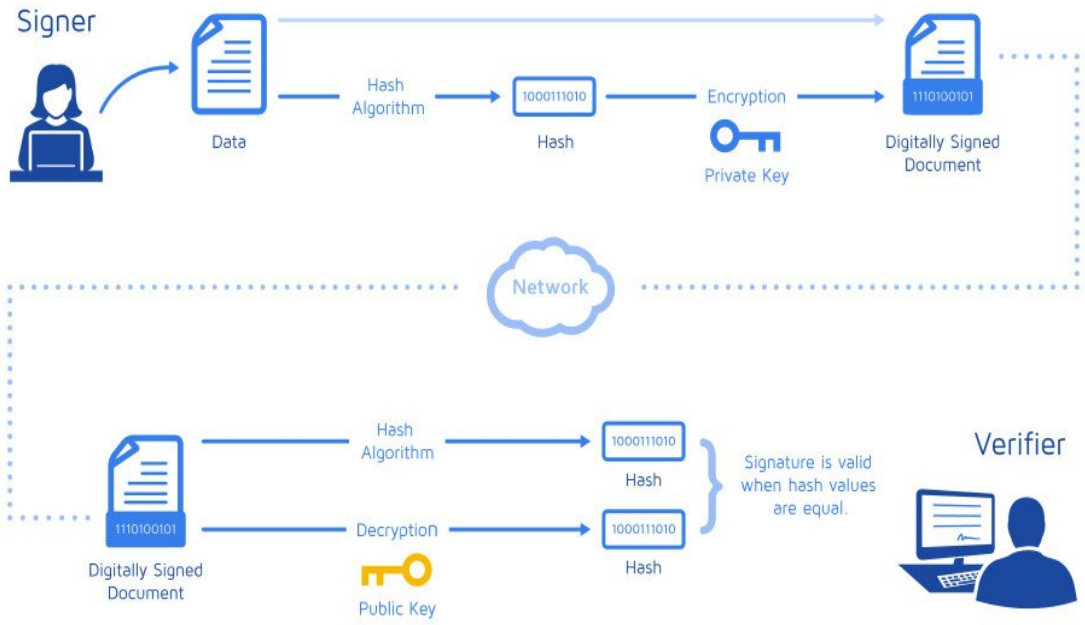

Figure 1.3: Digital Signature scheme. *Credits to* `www.docusign.com`.

In certain cases the car maker has to ensure that any request to access the CAN-bus is made by authorized entity and in particular that critical actions are performed by legitimate entities:

Example 1.2.4 (Diagnostic authentication (challenge and response))**.**
Apart from troubleshooting, diagnosis communication with vehicle and with ECUs is also used for more critical tasks, such as reprogramming (flashing) a specific control unit or updating software. It is crucial, then, to protect the communication between the tester and ECUs against unauthorized persons or equipment. To fulfill this need, an authorized tester has to be recognized and authenticated by the system, in order to perform critical actions: that is made by embedding a scheme of authenticated access, which is based on a *challenge and response* protocol and (possibly) on digital signature, described in example 1.2.3 A tester who has to make critical actions has to perform the following actions:

1. opening a request to the system (i.e. the ECU), which generates a random number to challenge the tester;

2. the tester encrypts the random number together with a pre-shared secret key, using a shared algorithm (or a one-way function) and sends the ciphertext back to the ECU;

3. the ECU performs the same encryption and checks if the two values match: if it's the case it permits the access.

This process could be even more secure by the use of certificates: in this case the tester has to ask the OEM (through its PKI[7]) for the access: the OEM signs a certificate using its private key; the tester sends the signed certificate to the ECU to request the access. In this case the authentication is performed directly between OEM and ECU, by means of tester's connection. The OEM signs a certificate reporting tester's identity and sends it to the ECU, though the tester, to request the access; the ECU sends a chance to the OEM, exploiting tester's connection; the OEM signs the challenge and sends it back to the ECU, that can validate it and prove the authentication.

Example 1.2.5 (Secure communication)**.**
During vehicle activity, ECUs exchange a lot of information between themselves to ensure the functionality of the whole system. This communications, via CAN-bus, should be checked to be secure. This is typically done by encrypting sensitive data, by asymmetric or symmetric encryption (e.g. by the use of the Diffie-Hellman protocol, described in example 1.2.1).

Each time the vehicle is started, an appointed ECU has to ensure that any other part wasn't modified or replaced (e.g. ECUs or software).

Example 1.2.6 (Component protection)**.**
To ensure integrity of the system (i.e. to check if an ECU has been substituted by an unauthorized one), a master ECU performs the following:

1. central ECU extracts a random number and sends it to a target ECU;
2. target ECU appends its unique ID[8] to the random number and encrypts it using a pre-shared key and sends it back;
3. central ECU decrypts the message, checks if the random number is the same it sent and if the ID appears among authorized ECU's IDs (stored into its memory).

Other employment of random numbers are, for example:

- symmetric key generation for ECUs internal usage, such as to encrypt parts of ECU's memory, authentication between ECUs and secure boot;
- asymmetric key generation for authentication;
- digital signature by the RSA-PSS algorithm, which works as in example 1.2.3 but using RSA to sign, instead of a hash function;

[7] Public Key Infrastructure by which a third-party could ask the manufacturer to certificate a key.
[8] Identity: its made by an alphanumeric string which is unique for each component.

- asymmetric encryption: when the encryption is based on RSA, the padding is often made by random values and it takes the name of *Optimal Asymmetric Encryption Padding* (OAEP) and the algorithm takes the name of RSA-OAEP.

To allow ECUs to perform cryptographic functionalities, as the above described, ECUs can be equipped with a specific hardware which fulfill all of these needs: an example is the Hardware Security Module (HSM). HSM guarantees a trusted execution environment where cryptography algorithms can be executed and sensitive data stored.

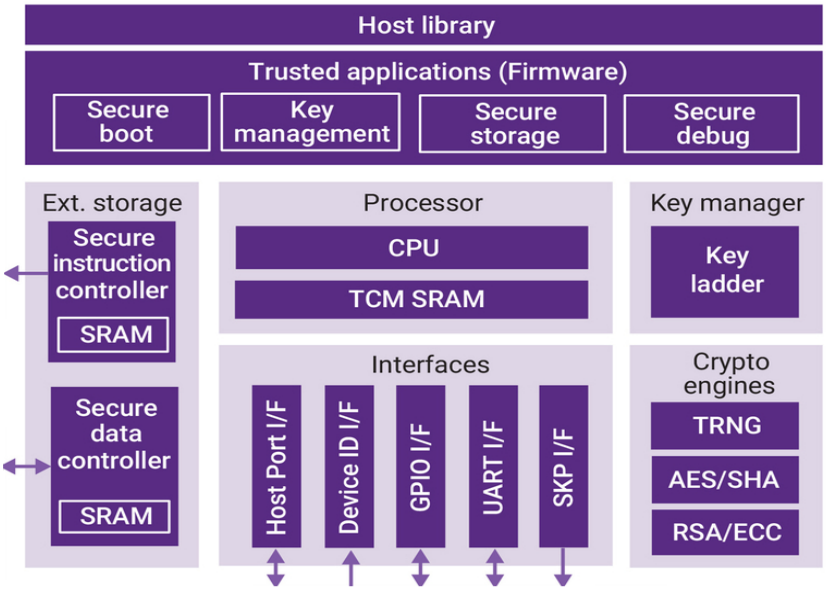

Figure 1.4: Hardware Security Module scheme. This HSM is provided with a True Random Number Generator (TRNG). *Credits to* `www.synopsys.com`.

1.3 Random Number Generators

It becomes fundamental to create keys, in the form of strings of digits (numeric or alphanumeric) which can't be uncovered by a possible attacker: that is done by the embedding of Random Number Generators (shortly RNG) in the cryptographic system (for example in the HSM, speaking about ECUs). RNGs are software or hardware (or, in some cases both software and hardware) systems, which generate random numbers as output. The output of an RNG is typically a binary sequence that could have any length: there exists generators which create decimal or alphanumeric strings, but they're less used and, in fact, any sequence could be converted in binary, as well as any binary sequence can be converted in any other encoding. For this reason Random Number

Generators are also, but rarely, called Random Bit Generators. In this thesis, unless differently specified, we will refer to RNGs and to random sequences/strings encoded in binary format. We could distinguish Random Number Generators between *Pseudo-Random Number Generators* (PRNGs) and *True Random Number Generators* (TRNGs) depending on the way they create their output.

PRNGs are algorithms which, starting from a short initial value, called *seed*, produce long sequences by manipulating that value through various mathematical functions in such a way that output appear random. Some trivial PRNGs were also made by pre-calculated tables from which a long sequence was extracted, depending from the seed. PRNGs can produce sequences which are indistinguishable from really random sequences in a suitable sense, even though they are, in fact, *deterministic*. When true randomness is not required, PRNGs are the most *efficient* choice, since they don't require any input but the seed and they can produce a huge amount of digits in a very short time. Note that, given a seed, a PRNG will always produce the same sequence as output: this property of *reproducibility*, is very useful if one has to simulate or model random phenomena. On the other hand, since the output can be predicted just knowing the seed, PRNGs shouldn't be used when the produced numbers have to be really unpredictable, such as lotteries, gambling, random sampling and security.

When effective unpredictability is required, it becomes necessary to use a TRNG: a Random Generator which takes values from some source of entropy, which is typically a random physical phenomena. Some typical source from which randomness is extracted are:

- radioactive source: a radioactive source typically decays at completely unpredictable times and that is often simple to register and use. This approach is the one implemented, for example, by the `HotBits` service of Fourmilab Switzerland (`www.fourmilab.ch`);

- shot noise: the noise produced by the fluctuation of electric current within electric circuits;

- movement of photons: typically observed while traveling through a semi-transparent mirror, detecting reflection and transmission;

- atmospheric noise: atmospheric noise could be picked up by any radio and its easy to be registered. The problem, in this case, is that it has to be detected in situation where there is no periodic phenomena. `www.random.org` follows this approach;

- thermal noise, typically taken from resistors.

Obviously, any other physical source could be used to extract digits from, but the randomness of the source have to be proved to call that a TRNG. Well designed TRNG could create high-quality random sequence, however typically TRNG are much slower than PRNG since they are related to the physical event.

To fulfill the request of a great amount of numbers in a short time, but maintaining the unpredictability of TRNGs, it is also possible to combine a hardware based RNG to a software based one (i.e. mixing a TRNG and a PRNG together): that type of RNG is called *Hybrid Random Number Generator* (HRNG). In this case hardware extract randomness from a physical source of entropy, like TRNGs do, creating a short, unpredictable, sequence; the sequence is then given as input to the software algorithm, which uses it as the seed of a PRNG and extends it to the desired length.
By this choice, the produced sequence maintains randomness properties and remains unpredictable, but can be generated much faster than a classical TRNG. However, the design of a HRNG is doubly critical since both the physical entropy source and the algorithm has to be well designed and has to work properly.
Many manufacturers are taking this path, due to the high demand of efficiency requested, while maintaining the unpredictability (in the form of unguessability) feature.

We said that PRNGs are not suitable to use in some application, with cryptography among all however, at least theoretically, there exist a small subset of Pseudo Random Number Generators which could be used in fields where a TRNG would otherwise be requested: these are called *Cryptographically Secure Pseudo-Random Number Generators* (CSPRNGs). A PRNG is said to be cryptographically secure if it fulfill two requirements:

i) next-bit test: if an attacker knows the first k bits of a produced random sequence, there is no polynomial-time algorithm which can predict the $(k+1)$th bit with more than 50% of probability;

ii) state compromise extensions: if an attacker guess a part or all of the states of the PRNG, there is no way he can reproduce the stream of random numbers which where produced in previous states (i.e. he can't go backward).

The design of a CSPRNG is hard and most PRNG fail to satisfy one of the two requests, however there are many PRNG which have been awarded by the cryptographically secure prize: for example the ChaCha20 algorithm (embedded in Mac OS) , the Fortuna algorithm (Linux) and the CryptGenRandom (Microsoft). Some other CSPRNGs have also been standardized (in [4] and in [2]), such as Hash_DRBG, HMAC_DRBG and CTR_DRBG.

A well designed RNG has to create sequences which are random by themselves and which have no relations with other created sequences. In the following we will present a

standardized method to evaluate if a RNG works properly analyzing the behavior of its outputs.

Chapter 2

The notion of randomness

This chapter will present an overview of the state of the art of randomness definitions.

2.1 Theoretical background

This section consists of a short introduction to the notions of probability which will be useful to test randomness.

2.1.1 Probability bases

We give, here, some basic definitions we'll use in the following.

Definition 2.1.1 (σ-algebra)**.**
Fixed Ω an arbitrary non-empty set, a σ-algebra $\mathcal{F}$ over Ω is a family of subsets of Ω s.t.

i) $\Omega \in \mathcal{F}$;

ii) *if* $A \in \mathcal{F} \Rightarrow A^c \in \mathcal{F}$;

iii) *if* $(A_n)_{n\in\mathbb{N}} \in \mathcal{F} \Rightarrow \bigcup_{n\in\mathbb{N}} A_n \in \mathcal{F}$.

Definition 2.1.2 (Probability measure)**.**
A probability measure on $(\Omega, \mathcal{F})$ is a function $P : \mathcal{F} \to [0, 1]$ such that

i) $P(\Omega) = 1$;

ii) *P is σ-additive: if $\{A_n\}_{n\in\mathbb{N}} \subseteq \mathcal{F}$ is a countable collection of pairwise disjoint sets, then*

$$P(\bigcup_{n\in\mathbb{N}} A_n) = \sum_{n=1}^{\infty} P(A_n).$$

Elements of $\mathcal{F}$ are called events and if $A \in \mathcal{F}$, the quantity $P(A)$ is said to be the probability of (the event) A.

Definition 2.1.3 (Probability space)**.**
A probability space is a triple $(\Omega, \mathcal{F}, P)$ where:

- *Ω is an arbitrary non-empty set;*
- *$\mathcal{F}$ is a σ-algebra over Ω;*
- *P is a probability measure.*

Definition 2.1.4 (σ-algebra generated by a subset)**.**
Let A be a set of subsets of Ω. Define

$$\sigma(A) = \{A \subseteq \Omega \text{ s.t. } A \in \mathcal{F} \text{ for all } \sigma\text{-algebras } \mathcal{F} \text{ containing } A\}$$

$\sigma(A)$ is a σ-algebra, which is called the σ-algebra generated by A. It is the smallest σ-algebra containing A.

Definition 2.1.5 (Borel σ-algebra)**.**
Given $d \in \mathbb{N}$. The σ-algebra $\mathcal{B} = \mathcal{B}(\mathbb{R}^d)$ generated by the Euclidean toplogy of $\mathbb{R}^d$ is called the Borel σ-algebra.

Definition 2.1.6 (Random variable)**.**
Let $(\Omega, \mathcal{F}, P)$ be a probability space and fix the couple $(\mathcal{V}, \mathcal{A})$, where $\mathcal{V}$ is a set and $\mathcal{A}$ is a σ-algebra. A function $X : \Omega \to \mathcal{V}$ is called a random variable if $X^{-1}(A) \in \mathcal{F}\ \forall A \in \mathcal{A}$. X is said to be discrete if $\mathcal{V}$ is countable. If $\mathcal{V} \subseteq \mathbb{R}$, X is said to be real.
The probability that X takes on a value in a measurable set $S \subseteq \mathcal{V}$ is written as

$$P(X \in S) = P(\{\omega \in \Omega | X(\omega) \in S\})$$

and is called the probability distribution of X.

Definition 2.1.7 (Conditional probability)**.**
Let $(\Omega, \mathcal{F}, P)$ be a probability space and $A, B \in \mathcal{F}$ be two events. If $P(B) > 0$, the conditional probability of A given B (i.e. the probability of A to occur, while B occurs) is defined as

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

where $P(A \cap B)$ is the probability that both A and B occur.
For two random variables X and Y, this definition becomes

$$P(\{X \in A | Y \in B\}) = \frac{P(\{X \in A\} \cap \{Y \in B\})}{P(\{Y \in B\})}.$$

If two events don't depend one another (we say that they are independent) we have that $P(A \cap B) = P(A)P(B)$ so that $P(A|B) = P(A)$.

Remark 2.1.1.
From the previous definition, it's easy to obtain the Bayes' theorem, which states that if $A, B \in \mathcal{F}$ are two events, than

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}.$$

Definition 2.1.8 (Independent random variables)**.**
Random variables $X_1, X_2, \ldots$ are said to be independent if for any $k \in \mathbb{N}$ and for any sequence $A_1, \ldots, A_k \in \mathcal{F}$ the equality

$$P(X_1 \in A_1, \ldots, X_k \in A_k) = P(X_1 \in A_1) \cdot \ldots \cdot P(X_k \in A_k)$$

holds.
For discrete random variables $X_1, X_2, \ldots$ this condition simplifies to

$$P(X_1 = x_1, \ldots, X_k = x_k) = P(X_1 = x_1) \cdot \ldots \cdot P(X_k = x_k)$$

for any sequence $x_1, \ldots, x_k \in \mathcal{V}$.

Definition 2.1.9 (Distribution function)**.**
A distribution function μ is a map

$$\mu : \mathcal{B}(\mathbb{R}^d) \to [0, 1]$$

defined by

$$\mu(S) = \sum_{x \in S \cap A} \gamma_d(x) + \int_S \gamma_c(x)dx, \qquad S \in \mathcal{B}$$

where

- *γ_c, called the probability density function of μ, is a $\mathcal{B}$-mesurable function*

  $$\gamma_c : \mathbb{R}^d \to [0, +\infty]$$

  such that

  $$\int_{\mathbb{R}^d} \gamma_c(x)dx =: I \in [0, 1]$$

- *γ_d, called the discrete distribution function of μ, is a function*

  $$\gamma_d : A \to [0, 1]$$

  where $A \subseteq \mathbb{R}^d$ is a countable set, and

  $$\sum_{x \in A} \gamma_d(x) = 1 - I$$

  with the convention that $\gamma_d(x) = 0$ if $x \in \mathbb{R}^d \setminus A$.

In the case that $\gamma_d \equiv 0$, we say that μ is an absolutely continue distribution function.

Definition 2.1.10 (Uniformly distributed random variable)**.**
A random variable X that assumes values in a finite set $\mathcal{V}$ is said to be uniformly distributed if it assumes all $v \in \mathcal{V}$ with the same probability.

Definition 2.1.11 (Stochastic process)**.**
A stochastic process is a parametrized collection of random variables $\{X_i\}_{i\in I}$ defined on a probability space $(\Omega.\mathcal{F}, P)$ and assuming values in $\mathbb{R}^n$.

Definition 2.1.12 (i.i.d. random variables)**.**
Let X and Y be two random variables on the same probability space. If X and Y are independent as in (2.1.8) and have the same probability distribution, than X and Y are said to be independent and identically distributed random variables: in the following we'll shortly say i.i.d..

A stochastic process $\{X_i\}_{i\in I}$ is said to be an i.i.d. process if random variables that forms it are.

Definition 2.1.13 (Stationary stochastic process)**.**
A stochastic process is said to be stationary if

$$P(X_1 \in A_1, \dots, X_k \in A_k) = P(X_{1+t} \in A_1, \dots, X_{k+t} \in A_k) \quad \forall k, t \in \mathbb{N}, \forall A_1, \dots, A_k \in \mathcal{F}$$

For discrete random variables, this condition simplifies to

$$P(X_1 = x_1, \dots, X_k = x_k) = P(X_{1+t}; \dots, X_{k+t} = x_k) \; \forall x_1, \dots, x_k \in \mathcal{V}.$$

Remark 2.1.2.
By definitions, an i.i.d. stochastic process is always stationary.

Definition 2.1.14 (Mean)**.**
The mean (or expected value) of a discrete real-valued random variable X is given by

$$E[X] := \sum_{x\in\mathcal{V}} xP(X = x).$$

If X is a continue random variable (i.e. $\int_\Omega |X(\omega)|dP(\omega) < \infty$), then its mean is

$$E[X] := \int_\Omega X(\omega)dP(\omega) = \int_{\mathbb{R}^n} xd\mu_X(x).$$

Definition 2.1.15 (Cumulative distribution function)**.**
Let X be a real-valued random variable on $(\Omega, \mathcal{F}, P)$.
The cumulative distribution function of X is the function

$$F_X : \mathbb{R} \mapsto [0, 1]$$

$$s.t. \; F_X(y) = P(X \leq y), \quad y \in \mathbb{R}.$$

Proposition 2.1.1.
If two random variables $X, Y : \Omega \to \mathbb{R}$ are independent then $X, Y \in L^1(\Omega, P)$ if and only if $XY \in L^1(\Omega, P)$. In that case we have

$$E[XY] = E[X]E[Y].$$

Definition 2.1.16 (Variance and standard deviation)**.**
Given a random variable X, we can define its variance and its standard deviation respectively as:

$$Var(X) := E[E[X] - X]^2$$
$$\sigma := \sqrt{Var(X)}$$

Definition 2.1.17 (Ergodic stochastic process)**.**
A stationary stochastic process is said to be ergodic if its statistical properties can be deduced from a single, sufficiently long realization of the stochastic process.

Definition 2.1.18 (Dirac distribution)**.**
The Dirac distribution δ_{x_0} centered on $x_0 \in \mathbb{R}$ is the distribution with

$$\gamma_c \equiv 0 \quad \textit{and} \quad \gamma_d = 1 \textit{ on } A = \{x_0\}.$$

So

$$\delta_{x_0}(S) = \begin{cases} 1 & \textit{if } x_0 \in S \\ 0 & \textit{if } x_0 \notin S. \end{cases}$$

Definition 2.1.19 (Normal distribution)**.**
The real normal distribution $N(\mu, \sigma^2)$ of parameters $\mu \in \mathbb{R}$ and $\sigma \geq 0$ is the distribution which density is

$$\gamma_c(x) = \begin{cases} 0 & \textit{if } \sigma = 0 \\ \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} & \textit{if } \sigma > 0 \end{cases}$$

and discrete distribution function on $A = \{\mu\}$

$$\gamma_d = \begin{cases} 1 & \textit{if } \sigma = 0 \\ 0 & \textit{if } \sigma > 0. \end{cases}$$

So $N(\mu, 0) = \delta_\mu$ and, if $\sigma > 0$,

$$N(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \int_S e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \qquad S \in \mathcal{B}(\mathbb{R}).$$

A particular, and really important special case, is when $\mu = 0$ and $\sigma^2 = 1$: in this case the distribution is called Standard Normal distribution and its cumulative distribution function is

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} dt. \tag{2.1}$$

An extremely useful result for our analysis will be the following

Theorem 2.1.2 (Central limit theorem).
Let $\{X_n\}$ be a stochastic process of independent and identically distributed random variables, whose mean μ and variance $\sigma^2 < +\infty$. Then the random variable

$$Z_n := \frac{X_1 + \cdots + X_n - n\mu}{\sigma\sqrt{n}}$$

approaches the Standard Normal Distribution $N(0,1)$ as n approaches to infinity. In other words, for all $a < b$ we have

$$\lim_{n\to\infty} P\left(a \leq \frac{X_1 + \cdots + X_n - n\mu}{\sigma\sqrt{n}} \leq b\right) = \Phi(b) - \Phi(a)$$

where Φ is the cumulative distribution function of the Standard Normal Distribution, as above.

2.1.2 Entropy

We introduce, now, the concept of entropy, as defined by Shannon [65]. Thanks to this concept, we can measure the uncertainty of a stochastic source of data or, equivalently, the average amount of information content, regardless the object of study.

Let X be a discrete random variable and let P be its probability distribution defined on $\mathcal{V} = \{x_1, \ldots, x_M\}$ such that $P(X = x_i) = P(x_i) = p_i$. We want to define a set of functions $\mathrm{H}_M : \mathbb{R}^M \to \mathbb{R}^+$ that meets the following axiomes:

i) Monotony: $f(M) := \mathrm{H}_M(\frac{1}{M}, \ldots, \frac{1}{M})$ be a strictly increasing function;

ii) Extensiveness: $f(LM) = f(L) + f(M) \quad \forall L, M \geq 1$;

iii) Groupability: fixed $r < M$ we define $\vec{q} = (q_A, q_B)$ s.t. $q_A = \sum_{i=1}^{r} p_i$ $(\Rightarrow q_B = 1 - q_A = \sum_{i=r+1}^{M} p_i)$

$$\mathrm{H}_M(\vec{p}) = \mathrm{H}_2(\vec{q}) + q_A \mathrm{H}_r\left(\frac{p_1}{q_A}, \ldots, \frac{p_r}{q_A}\right) + q_B \mathrm{H}_{M-r}\left(\frac{p_{r+1}}{q_B}, \ldots, \frac{p_M}{q_B}\right)$$

iv) Continuity.

Theorem 2.1.3 (Shannon).
The only function (up to multiplicative constant) which satisfies the previous axioms is

$$\vec{p} \mapsto H(\vec{p}) = -c \sum_{i=1}^{M} p_i \log_b p_i$$

with the convention that $0 \log 0 = 0$ *(which is consistent by the fact that* $\lim_{p \to 0} p \log_b \frac{1}{p} = 0$*). The function H is said to be the entropy function (or simply entropy).*

Proof.
It's easy to prove that the function H meets the previous axioms, so we'll show the existence and uniqueness of that function.

1. $\forall M, k \geq 1$ we can write $f(M^k) = f(M \cdot M^{k-1}) = f(M) + f(M^{k-1})$ using the groupability property.
   Iterating the procedure we find $f(M^k) = f(M) + f(M^{k-1}) = \ldots = kf(M)$

2. Let's show that $\forall M \geq 1 \; \exists \, c > 0$ s.t. $f(M) = c \log M$:
   if M=1 we have that

   $$f(1) = f(1 \cdot 1) = f(1) + f(1) \Rightarrow f(1) = 0.$$

   If, otherwise, $M > 1$, one can prove that for all $r \geq 1$ exists $k \in \mathbb{R}$ such that

   $$M^k \leq 2^r \leq M^{k+1},$$

   but $f(M) = \mathrm{H}_M(\frac{1}{M}, \ldots, \frac{1}{M})$ is a strictly increasing function, so we obtain

   $$f(M^k) \leq f(2^r) \leq f(M^{k+1}).$$

   Using the fist step

   $$kf(M) \leq rf(2) \leq (k+1)f(M) \Rightarrow \frac{k}{r} \leq \frac{f(2)}{f(M)} \leq \frac{k+1}{r},$$

   and the logarithm is strictly increasing too, so

   $$\frac{k}{r} \leq \frac{\log 2}{\log M} \leq \frac{k+1}{r} \Rightarrow \left| \frac{f(2)}{f(M)} - \frac{\log 2}{\log M} \right| \leq \frac{1}{r}.$$

   r was arbitrary then, to the limit for r to infinity,

   $$f(M) = \frac{f(2)}{\log 2} \log M = c \log M$$

3. We took uniform p_i: let's extend this definition choosing $p_i = \frac{r_i}{M} \in \mathbb{Q}, \; i = 1, \ldots, N$;
   let's divide the element of $\vec{p}$ in groups of r_i: each r_i element will have a probability

of $\frac{1}{r_i} = \frac{1}{M}/p_i = \frac{1}{M}/\frac{r_i}{M}$.
Let's use now the groupability property, obtaining

$$c \log M = f(M) = f(M) = \mathrm{H}_M\left(\frac{1}{M}, \ldots, \frac{1}{M}\right)$$
$$= \mathrm{H}(p_1, \ldots, p_N) + \sum_{i=1}^{N} p_i \mathrm{H}_{r_i}\left(\frac{1}{r_i}, \ldots, \frac{1}{r_i}\right) = \mathrm{H}(p_1, \ldots, p_N) + \sum_{i=1}^{N} p_i c \log r_i$$

then

$$\mathrm{H}(p_1, \ldots, p_N) = c\left(\log M - \sum_{i=1}^{N} p_i \log r_i\right) = c\left(-\sum_{i=1}^{N} p_i \log \frac{r_i}{M}\right) = -c \sum_{i=1}^{N} p_i \log p_i$$

We can now extend it to all of $\mathbb{R}$ thanks to the continuity axiom.

□

Remark 2.1.3.
The definition is consistent for any base of the logarithm b. Common values of b are 2, the Euler's number e and 10, and the corresponding units of entropy are, respectively, bits, nats and bans.
Our purpose is to analyze binary strings: we'll choose $b = 2$, so that $c = 1$. In the following we will always use that base, than we'll simply write log, instead of $\log_2$.

Remark 2.1.4.
By definition we have that $\mathrm{H}(x) \geq 0$.

Making some examples, we want to to show what we mean for "measure the uncertainty" of a source of data

Example 2.1.1.
If $\vec{p} = (0, \ldots, 0, 1, 0, \ldots, 0)_M$ we obtain that $\mathrm{H}(\vec{p}) = 0$: in the case that the distribution of probability is some of this kind, we don't have any uncertainty.

Example 2.1.2.
Let's try to find which is the event $\vec{p} = (p_1, \ldots, p_M)$ which maximize the entropy H.
We consider the function

$$h : \vec{p} \mapsto \mathrm{H}(\vec{p}) = -\sum_{i=1}^{M} p_i \log p_i$$

with the constraint that $\varphi(\vec{p}) = \sum_{i=1}^{M} p_i - 1 = 0$. Let's use the Lagrange multipliers:

$$\partial_{p_{ij}}(\mathrm{H}(\vec{p}) + \lambda\varphi(\vec{p})) = 0$$

$$-\log p_j - p_j \frac{1}{p_j}\frac{1}{\log 2} + \lambda = 0 \Rightarrow -\log p_j = \lambda \log 2 = cost. \Rightarrow p_j = cost.$$

p_j also has to meet the constrait, so we obtain that $p_j = \frac{1}{M}$.
One can easily see that the uniform vector $\vec{p} = \left(\frac{1}{M}, \ldots, \frac{1}{M}\right)$ is a maximum for the entropy: in this case we have $\mathrm{H}(\vec{p}) = \log M$.

Example 2.1.3.
If $\vec{p} = \left(\frac{1}{M-1}, \ldots, \frac{1}{M-1}, 0\right)_M \Rightarrow \mathrm{H}(\vec{p}) = \log(M-1)$, so that it's the same as having one less choice.

2.2 Characterizations of randomness

Many disciplines tried to give a definition of randomness and found different ways to call something "random". We are interested in expressing what is a "binary random string".

2.2.1 Probabilistic approach

One can formulate the notion of random string in the following way:

Definition 2.2.1 ((Probabilistic) random string)**.**
Let $\{X_i\}_{i\in I}$ be a stochastic process to assume values in $\mathcal{V} = \{r_1, \dots, r_M\}$ with probability $P(r_i) = P(r_j)\ \forall i, j$.
If the process consists on independent and identically distributed (i.i.d.) random variables, then its realization $(r_1, \dots, r_N) \in \mathcal{V}^N$ is called a random string.

Remark 2.2.1.
In the case of a binary string $(r_1, \dots, r_M)$, $r_i \in \mathcal{V} = \{0, 1\}$ we can consider the sequence as a realization of the same random variable X. Obviously, in this case, the probabilities are $P(0) = P(1) = \frac{1}{2}$ as in a coin flip.

The distribution that models the outcomes of an i.i.d. process is known as the Bernoulli distribution.

Definition 2.2.2 (Bernoulli distribution)**.**
Given $p \in [0, 1]$, the Bernoulli distribution function is defined on $\bar{\mathbb{R}}$ and is defined in the following way:

$$\gamma(x) = \begin{cases} p & \text{if } x = 1, \\ 1 - p & \text{if } x = 0, \\ 0 & \text{otherwise.} \end{cases}$$

The respective distribution is

$$B_{(1,p)}(H) = \begin{cases} 0 & \text{if } 0, 1 \notin H, \\ 1 & \text{if } 0, 1 \in H, \\ p & \text{if } 1 \in H, 0 \notin H, \\ 1 - p & \text{if } 0 \in H, 1 \notin H. \end{cases}$$

If a random variable X has a Bernoulli distribution, then $P(X = 1) = p$ and $P(X = 0) = 1 - p$.

Practically speaking, a Bernoulli process is a finite or infinite sequence of independent random variables $X_1, X_2, \dots$ such that:

i) the value of X_i is either "0" or "1", $\forall\, i$;

ii) $P(X_i = 1) = p, \quad P(X_i = 0) = 1 - p.$

So we can rationally model an ideal random string as the realization of a Bernoulli process with $p = \frac{1}{2}$.

A closely connected, and very useful in the following, distribution is the

Definition 2.2.3 (Binomial distribution)**.**
Let $n \in \mathbb{N}$ and $p \in [0, 1]$. The binomial distribution of parameters n and p is defined by the distribution function

$$\gamma(k) = \begin{cases} \binom{n}{k} p^k (1-p)^{n-k} & \text{if } k = 0, 1, \ldots, n \\ 0 & \text{otherwise.} \end{cases}$$

So if a random variable X has a binomial distribution (by symbols $X \sim Bin(n, p)$) we have

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}, \qquad k = 0, 1, \ldots, n.$$

Example 2.2.1.
Typical examples of binomial random variables are:

- the random variable "number of successes in n independent repeted trials with probability p", which has $Bin(n, p)$ distribution;
- if we suppose to have r objects into n boxes, the random variable "number of objects into the first box" has a $Bin(n, \frac{1}{r})$ distribution.

Note that this point of view, nevertheless it provides ideas and characterization of random string, is in fact the main base of all of the following approaches and theories.

2.2.2 Information Theory approach

As we discussed in section (2.1.2), we could give a measure of the uncertainty of a source through the concept of entropy and this uncertainty of the source corresponds to the amount of information which the data contains.

We modeled a random binary string as a Bernoulli process, i.e. as a sequence of independent random variables $X_1, X_2, \ldots$ with $P(X_i) = \frac{1}{2}\ \forall i$, so that the information contained in it is maximum: this means that every single bit of the sequence contains relevant information. If each single bit of the sequence contains some information, that implies that we couldn't compress the string in a relevant way, without losing some of the information contained inside it. In the following we try to formalize this idea and to

show that this is strictly connected with the notion of entropy.

Let $\bar{X} = \{X_i\}_{i \in I}$, $I \subseteq \mathbb{N}$ be a stochastic process. To express how the entropy grows as soon as the process goes on (i.e. we add bits to the sequence) we can give the following

Definition 2.2.4 (Entropy-rate of a stochastic process)**.**
The entropy rate of a stochastic process is defined as

$$H(\bar{X}) := \lim_{n \to \infty} \frac{H(x_1, \ldots, x_n)}{n}$$

if this limit exists.

Using the fact that $H(x_1, \ldots, x_n) = \sum_{i=1}^{n} H(x_i | x_1, \ldots, x_{i-1})$ one can easily observe that the following holds:

Theorem 2.2.1.
If $\bar{X}$ is a stationary process, than $H(\bar{X})$ exists and

$$H(\bar{X}) := \lim_{n \to \infty} H(x_n | x_1, \ldots, x_{n-1}).$$

Example 2.2.2.
Let us consider a stochastic process formed by a sequence of i.i.d. random variables. In this case it holds that

$$H(X_1, \ldots, X_n) = H(X_1, \ldots, X_{n-1}) + H(X_n).$$

Iterating we find that

$$H(X_1, \ldots, X_n) = nH(X_1)$$

since $H(X_i) = H(X_j)\ \forall i, j$, so that the entropy rate of this process is exactly

$$H(\bar{X}) = \lim_{n \to \infty} \frac{H(X_1, \ldots, X_n)}{n} = \frac{nH(X_1)}{n} = H(X).$$

Since, as we have seen in previous section, a random sequence is produced by a Bernoulli process (which is an i.i.d. process) this is exactly the case of our interest, and the entropy rate is maximum, since the entropy is.

To compress a sequence we should define a way to transform elements of it or, in other words, we need a

Definition 2.2.5 (Code)**.**
A code C for a random variable X, which takes values on an alphabet χ, is a map

$$C : \chi \to \mathcal{D}^* = \bigcup_n \mathcal{D}^n$$

such that

$$x \mapsto C(x),$$

where $\mathcal{D}$ is an alphabet and $C(x)$ is said to be a code word. To each code word we can naturally associate it's length $l(x)$.

Each code can be extended to finite strings of the alphabet χ through

$$C^* : \chi^* \to \mathcal{D}^*$$

defined as

$$(x_1, \ldots, x_n) \mapsto C^*(x_1, \ldots, x_n) = C(x_1) \cdot \ldots \cdot C(x_n).$$

where $\cdot$ stands for the concatenation.

In the following we will consider the most useful codes which permits to compress/-code and decompress sequences without requiring any other information, i. e. codes with the following properties:

- Non-singular: if $x_i \neq x_j \Rightarrow C(x_i) \neq C(x_j)\ \forall i, j$;
- Prefix-free: no code word is prefix of other code words, i.e. if $c(x_i) = \alpha \Rightarrow \nexists j$ s.t. $C(x_j) = \alpha * **$.

Codes which satisfy these requirements are called *instantaneous* or *prefix code.*

To assess the existence of an instantaneous code we can use the following theorem, proved by Kraft in [40]:

Theorem 2.2.2 (Kraft's inequality)**.**
If C is an instantaneous code on $\mathcal{D}$, with $|\mathcal{D}| = D$, the set of length $\{l_1, \ldots, l_m\}$ is such that $\sum_{i=1}^{m} D^{-l_i} \leq 1$.
Vice versa, if the set of length $\{l_1, \ldots, l_m\}$ satisfy the inequality $\Rightarrow$ there exists an instantaneous code on $\mathcal{D}$ with those length.

Since we are interested in instantaneous codes, this theorem gives us constraints on the length of code words which permits to study them.
We are interested to compress a sequence, i.e. to minimize it's length. Let's consider

Definition 2.2.6 (Average length of the code)**.**
If $p(x)$ is the probability of the word x to occur and $l(x)$ is the length of the coded word, the average length of the code is defined as

$$L(C) = \sum_{x \in \chi} p(x) l(x).$$

A code that minimize the average length is said to be optimal.

We can find a lower bound for the average length, i.e. we can find the best performance for an instantaneous code, minimizing the average length function, with the constraints that the length of code words satisfy Kraft's inequality.

Proposition 2.2.3.
For each instantaneous code for a random variable X one has

$$L(C) \geq H_D(X),$$

where $H_D(X)$ is the entropy where the base of the logarithm is D.

Proof.
Let's use Lagrange multipliers to minimize the average length with the constraints of the Kraft inequality:

$$J(l_1, \ldots, l_m) = \sum_i p_i l_i + \lambda \left(\sum_i D^{-l_i} \right)$$

where p_i and l_i stands for the probability and length of the i-th symbol of the alphabet.

$$\frac{\partial J}{\partial l_k} = p_k + \lambda(-log_e D \cdot D^{-l_k}) = 0$$

from which

$$p_k = (\lambda log D) D^{-l_k}.$$

Now $\sum p_k = 1$ and if we suppose $\sum D^{-l_k} = 1$, we obtain

$$\lambda log D = 1 \Rightarrow \lambda = \frac{1}{log D}$$

so that

$$p_k = D^{l_k} \Leftrightarrow l_k = -log_D p_k.$$

Substituting we obtain that the minimum for the average length is $\tilde{L} = H_D(X)$. □

We can extend this definition to words of any chosen length by defining the

Definition 2.2.7 (Average length per-symbol)**.**
If $l(x_1, \ldots, x_n)$ is the length of the code word associate to the word $(x_1, \ldots, x_n)$,

$$L_n(C) = \frac{1}{n} \sum_{x_1, \ldots, x_n} p(x_1, \ldots, x_n) l(x_1, \ldots, x_n)$$

is said to be the average length per-symbol.

One can prove that, for an optimal code the following inequality holds:

$$\frac{H_D(x_1,\ldots,x_n)}{n} \leq L_n(C) \leq \frac{H_D(x_1,\ldots,x_n)+1}{n}$$

which to the limit $n \to \infty$ becomes

$$H(\bar{X}) \leq L_n(C) \leq H(\bar{X}).$$

As we have seen, for a random sequence the entropy rate is, in fact, the entropy of the process (which is maximum for random sequences), therefore studying the compression rate of optimal codes applied to sequences seems to be a good way to test for randomness: this has been done by most of test suites for randomness for long time, since equivalent but more efficient methods were found. The most used test suites, and especially NIST's, used the LZ78 code or its variants: a universal and optimal code, proposed by Lempel and Ziv in [71] and enhanced in [72]. Sequences were effectively compressed, using LZ78 and the compression rate was analyzed to decide whether or not the sequence was random: due to the high complexity of the compression algorithm and to some weakness of the implementation, such as constraints on the length of the sequence to test, this test was removed in 2004, since the Serial test (test 11) and the Approximate entropy test (test 12) produce analogous results.

2.2.3 Complexity Theory approach

In this section, we will present the approach of theory of complexity, introduced by Kolmogorov (see [9] for a review) and extended by Chaitin (see for example [12]), which defines randomness in terms of the length of the shortest code (called *Chaitin computer*) that could create the sequence itself. This approach also leads to the sharable idea that randomness implies an absence of regularity which is, obviously, strictly connected to the idea of incompressibility presented by the theory of information.

In this section we will use the concept of *partially computable function*, typical of the algorithmic information theory: a function $\varphi : X \to Y$ which is defined on a subset $Z \subseteq X$ is called a *partial function*; if $dom(\varphi) = X$, the function is said to be a *total function*. A partial function is said to be *partially computable function* if it can be completely described by an algorithm.

The Chaitin definition of randomness is based on the following

Definition 2.2.8 (Chaitin computer)**.**
Let $A = \{a_1,\ldots,a_Q\}$ be a sortable alphabet.
A computer is a partially computable function $C : A^* \times A^* \to A^*$.
A Chaitin computer is a computer C such that $\forall v \in A^*$ the domain of C_v is prefix-free,

with $C_v : A^* \to A^*$ *such that* $C_v(x) = C(x, v)\ \forall x \in A^*$.
A (Chaitin) computer C *is said to be universal if for each (Chaitin) computer* φ *there is a constant* c *(which depends on* ψ *and* φ*) such that if* $\varphi(x, v) < \infty \Rightarrow \exists x'$ *s.t.* $\psi(x', v) = \varphi(x, v)$ *and* $|x'| \leq |x| + c$.

As in [11] (Theorem 3.3) one can prove that a (Chaitin) universal computer effectively exists. In the sequel we will use ψ to indicate a universal computer and U for a Chaitin universal computer and we'll use them to define complexities of strings.

Definition 2.2.9 (Complexities)**.**
a) The ***Kolmogorov-Chaitin absolute complexity****, associated with a computer* φ *is the function* $K_\varphi : A^* \to \mathbb{N}$ *defined as*

$$K_\varphi(x) := min\{|u| \ s.t. \ u \in A^*, \ \varphi(u, \lambda) = x\}.$$

If $\varphi = \psi$ *we will write* $K(x)$ *instead of* $K_\psi(x)$.
b) The ***Chaitin absolute program-size complexity****, associated with the Chaitin computer* C *is the function* $\mathcal{H}_C : A^* \to \mathbb{N}$ *defined as*

$$\mathcal{H}_C(x) = min\{|u| \ s.t. \ u \in A^*, \ C(u, \lambda) = x\},$$

and we will write $\mathcal{H}(x) = \mathcal{H}_U(x)$ *if the associated computer* C *is the universal computer* U.

According to both definitions, the complexity of a string x is the the length of the shortest string y one has to give as input to a computer to obtain the string x: we can use this notion as the main quantity to define randomness. One can prove that the two notion of complexity are equivalent and that

$$K(x) \leq K_\varphi(x) + O(1), \quad \mathcal{H}(x) \leq \mathcal{H}_C(x) + O(1)$$

for each computer φ and for each Chaitin computer C.

Example 2.2.3 (Paradox of randomness)**.**
Consider the following binary strings:

$$x = 00000000000000000000000000000000,$$

$$y = 10011001100110011001100110011001,$$

$$z = 011010001001101oo101100100010110,$$

$$u = 00001001100000010100000010100010,$$

$$v = 01101000100110101101100110100101.$$

According to the classic probability theory all the previous strings have the same probability, which is 2^{-32} since they are made of 32 digit which can be 0 or 1, however, if we focus on regularity of those, they are extremely different: indeed we can express x simply by "32 zeros", y is "eight time 1001" and z is "0110100010011010 concatenated with it's mirror". This example also shows the strict relationship between regularity and compressibility, since the more a sequence is regular, the more we can compress it.
u and v are more cryptically: we can't apparently find any sort of regularity and we can't express them by a shorter formula. Simplistically thinking, this could be considered as the main idea behind the definition of randomness given by the theory of complexity.

To give a definition of randomness based on complexity, one should prove that, each time we choose a length n for the sequence, there is a maximum for the complexity of strings of n digits. We will present here the main results which lead to the definition of randomness given by Chaitin (see for example [11] for proofs).

Theorem 2.2.4 (Chaitin)**.**
Let $f : \mathbb{N} \to A^*$ be an injective, computable function.

- *The following inequality holds*

$$\sum_{n\geq 0} Q^{-\mathcal{H}(f(n))} \leq 1.$$

- *Let $g : \mathbb{N}^+ \to \mathbb{N}^+$ be a computable function, then*

  i) If $\sum_{n\geq 1} Q^{-g(n)} = \infty$, then $\mathcal{H}(f(n)) > g(n)$ for infinitely many $n \in \mathbb{N}^+$.
  ii) If $\sum_{n\geq 1} Q^{-g(n)} < \infty$, then $\mathcal{H}(f(n)) \leq g(n) + O(1)$.

Using this theorem and choosing $g(n) = \lfloor log_Q n \rfloor$ and $g(n) = 2\lfloor log_Q n \rfloor$ one can find the following bound for the Chaitin complexity of a string of length n on an alphabet of cardinality Q:

$$\lfloor log_Q n \rfloor < \mathcal{H}(string(n)) \leq 2\lfloor log_Q n \rfloor + O(1).$$

Thanks to this bound we have

Theorem 2.2.5.
For every $n \in \mathbb{N}$

$$\max_{x\in A^n} \mathcal{H}(x) = n + \mathcal{H}(string(n)) + O(1).$$

Since the maximum always exists, we can define a function $\Sigma : \mathbb{N} \to \mathbb{N}$ such that

$$\Sigma(n) = \max_{x\in A^n} \mathcal{H}(x) = n + \mathcal{H}(string(n)) + O(1).$$

A random string of length n could be defined as the string which have the maximal complexity among all strings of length n, i.e. the strings $x \in A^n$ such that $\mathcal{H}(x) \approx \Sigma(n)$ or, in other words, we can give the following

Definition 2.2.10 (Chaitin random string)**.**
A string $x \in A^*$ is said to be ***Chaitin*** *m-**random**, with $m \in \mathbb{N}$, if $\mathcal{H}(x) \geq \Sigma(|x|) - m$. x is* ***Chaitin random*** *if it is* 0*-random.*

Moreover, by combinatorial arguments, one can also prove the following result on the numerosity of Chaitin random strings:

Theorem 2.2.6.
For all $n \in \mathbb{N}$ there exists a constant $c > 0$ such that

$$\gamma(n) = \#\{x \in A^n \ s.t. \ \mathcal{H}(x) = \Sigma(|x|)\} > Q^{n-c}.$$

This approach is the main focus for the Binary matrix rank test (test 5), the Discrete Fourier transform test (test 6) and for the two test which analyze templates (test 7 and 8).

2.2.4 Algorithmic Information Theory approach

The goal of Martin-Löf's theory was to prove that the definitions of randomness given by Kolmogorov and Chaitin through the theory of complexity are consistent with the classical probability theory: the strength of his theory is that it includes all known and unknown properties of random sequences.

Let's suppose to have an element x of a general sample space and to test if this element is typical, i.e. if it belongs to some majority or if it is a special case among that sample space. Considering all these possible "majorities", an element would be "random" if it lies on the intersection of all majorities or, conversely, it would be "non-random" if there is some case where it can be considered "particular": this is, basically, the idea if the classical hypothesis testing we'll explain and use in the following chapter.

Definition 2.2.11 (Martin-Löf test)**.**
A set $V \subset A^* \times \mathbb{N}^+$ is called a ***Martin-Löf test*** *if the following holds:*

i) $V_{m+1} \subset V_m \ \forall m \geq 1$;

ii) $\#(A^n \cap V_m) < Q^{n-m}/(Q-1) \ \forall n \geq m \geq 1$.

$V_m = \{x \in A^* \ s.t. \ (x, m) \in V\}$ is called a m-section of V. For each m, the set V_m is called the critical region at level $Q^{-m}/(Q-1)$ and we say that a string is "random" at level m, with respect to the test V if $x \notin V_m$ and $|x| > m$.

Let make, here, two examples to show that tests we'll discuss in the following could be analyzed according to this theory.

Example 2.2.4.
Consider the set

$$V = \left\{ (x,m) \in A^* \times \mathbb{N}^+ \text{ s.t. } \left| \frac{N_j(x)}{|x|} - \frac{1}{Q} \right| > Q^m \frac{1}{\sqrt{|x|}} \right\},$$

where $N_j(x)$ is the number of occurrences of the j-th letter in the string x, which is a possible formulation of the frequency test: the most basic test that, in our (binary) case, analyzes if the number of zeros and ones is approximately the same, as we expect from a random sequence. This is a Martin-Löf test. Indeed the first condition is banally satisfied.
For the second condition, since

$$\# \left\{ x \in A^n \text{ s.t. } \left| \frac{N_j(x)}{|x|} - \frac{1}{Q} \right| > \epsilon \right\} \leq \frac{Q^{n-2}(Q-1)}{n\epsilon^2},$$

we obtain that

$$\begin{aligned} \#(A^n \cap V_m) &= \# \left\{ x \in A^n \text{ s.t. } \left| \frac{N_j(x)}{|x|} - \frac{1}{Q} \right| > Q^m \frac{1}{\sqrt{|x|}} \right\} \\ &= \frac{Q^{n-2}(Q-1)}{Q^{2m}} \\ &= Q^{n-2-2m}(Q-1) \\ &\leq \frac{Q^{n-m}}{Q-1} \end{aligned}$$

which is the second requirement of the definition.

Example 2.2.5.
Consider $\varphi : A^* \to A^*$ to be a computer. One can prove that the set

$$V(\varphi) = \{(x,m) \in A^* \times \mathbb{N}^+ \text{ s.t } K_\varphi(x) < |x| - m\},$$

is a Martin-Löf test. Note that this is a formulation for the test which rejects the hypothesis of randomness if regularities are found, as discussed in the previous section.

By this definition we can say what is "random" with respect to the test V, indeed we can give the following

Definition 2.2.12 (Martin-Löf random string)**.**
To a Martin-Löf test V we can associate a function $m_V : A^* \to \mathbb{N}$, defined as

$$m_V(x) = \begin{cases} max\{m \geq 1 \text{ s.t. } (x,m) \in V\} & \text{if } (x,1) \in V \\ 0 & \text{otherwise.} \end{cases}$$

which is called the critical level of the string x with respect to the test V.
We can say that x is q-random, according to the Martin-Löf test V, if $x \notin V_q$ and $q < |x|$.

Chaitin random strings and Martin-Löf's tests are strictly connected, indeed, as in [11], one can prove the following

Theorem 2.2.7.
Fixed $k \in \mathbb{N}$, almost all Chaitin k-random strings will be declared random by every Martin-Löf test.

Chapter 3

Statistical tests for randomness

As we have seen, the purpose of a Random Number Generator is to produce sequences of random digits, if it's a true RNG, or at least that behave as if they are random, in the case of a PRNG: how could we decide whether a sequence is random? The idea behind statistical tests is to compare the sequence, output of the RNG, to a sequence which is the output of an ideal random number generator. Each test inspects the sequence to assess the presence of some specific pattern or behavior which could indicate that the sequence is random. For an ideal random string the property is known a-priori, can be described in probabilistic terms and defines statistic T of each test. Each test gives us more and more confidence of the randomness of the sequence. The purpose is to propose a collection of tests that could cover a sufficiently large set of defections.

3.1 General procedure

The idea behind the tests is the hypothesis testing: let's fix a parameter space Θ, which indexes all possible probability distributions of the observed data. We define the null hypothesis H_0 as a subset $\theta_0 \in \Theta$ which corresponds to special parametric values of interest. The so called alternative hypothesis H_1 i.e. $\theta \in \Theta_1$ can also be considered: typically the alternative hypothesis corresponds to the complementary one $\Theta_1 = \Theta \setminus \Theta_0$. Once the null hypothesis is defined, one tests data to decide if he should maintain or reject his hypothesis, i.e. if the inspected objects belong to the subset Θ_0 or to its complementary.

In our case of testing randomness, the hypothesis is that the sequence is random and the alternative one is that it is not, we can give the following

Definition 3.1.1 (Null hypothesis (for randomness))**.**
Let Θ be the set of all possible distributions of a binary sequence, Θ_0 the set of distributions of ideal random binary sequence and $\Theta_1 = \Theta \setminus \Theta_0$.

The null hypothesis $H_0 : \theta \in \Theta_0$ is that bits of the observed sequence represent independent Bernoulli random variables with the probability of success $p(1) = p(0) = \frac{1}{2}$

As we said, each statistical test is based on a statistic T: to decide if we should maintain or reject the null hypothesis, one shall see if the observed statistic T is sufficiently close to the expected one T_0, which is the same statistic under H_0. According to that, one should define a cut-off value to make this decision. This approach leads to

Definition 3.1.2 (Significance level)**.**
The significance level α is the probability that the null hypothesis is rejected.

This significance level represents the probability that a "good" generator produced a non-random sequence. α is a-priori defined and empirical results in cryptography showed that it can be chosen between 0.001 and 0.01. One could also define the probability that a "bad generator" produced a random sequence; this is the

Definition 3.1.3 (Power of the test)**.**
The power of the test β is the probability that the alternative hypothesis H_1 is rejected, although is true (i.e. H_0 is accepted).

So that, the observer may commit two types of error: judging the generator as a "bad" generator while in fact it effectively generate random numbers, with probability α, (we call that *type I error*) or concluding that the sequence is random (i.e. produced by a "good" random generator) while it's not, with probability β (this will be called *type II error*). We summarize possible conclusions in the following table:

	CONCLUSION	
TRUE SITUATION	**Accept H_0**	**Reject H_0**
Data is random (H_0 is true)	No error	Type I error
Data is non-random (H_0 is false)	Type II error	No error

The probability β of a *type II* error is always difficult to express, however α and β are related to each other: a small α corresponds to a high β and vice versa. In most of hypothesis testings, the significance level α is chosen to be on the order of 0.05, however in cryptography α is commonly set to smaller values (i.e. $\alpha \leq 0.01$). Note that setting, for example, $\alpha = 0.01$ means that, *under randomness hypothesis* (i.e. our data is effectively random), we expect to reject the null hypothesis in less than 1% of cases. The existence of these errors shows that evaluating if the observed (empirical) values are sufficiently close to the expected one it's not enough, because we can fall in one of the previous errors: it's also useful to know how often we obtain the value of the statistic T(obs) we obtained from our sequence. Assume that the test leads to rejection of the null hypothesis for large values of a test statistic T (i.e. when $T > T_0$): the main characteristic to find is the

Definition 3.1.4 (Empirical significance level (*P-value*))**.**
The empirical significance level, called the P-value, is the probability that the random variable T exceeds its observed value T(obs), evaluated under the null hypothesis H_0. In formula we can write P-value $= P(T > T(obs)|H_0)$.

The advance of this approach is that the null hypothesis is rejected when the *P-value* is smaller than α. So, if for each test we compute the *P-value* we obtain two information: we know if the sequence is random and we get the probability this string is an output of a truly random generator. By this approach, a *P-value* which equals to 1 assess that the tested sequence has a perfect randomness, while a *P-value* which equals to zero says that our sequence is completely non-random. Chosen a significance level α, we proceed as follows

- if *P-value* $> \alpha \Rightarrow$ we accept the null hypothesis or, more precisely, we don't reject it;
- if *P-value* $< \alpha \Rightarrow$ we reject the null hypothesis and we conclude that the sequence is non-random;
- if *P-value* $\approx \alpha \Rightarrow$ we don't reject the null hypothesis, but we apply another test.

As consequence, the *P-value* summarizes the strength of the evidence against the null hypothesis and gives us a confidence level: we can better explain this fact using the following examples

Example 3.1.1.
An $\alpha = 0.01$ indicates that we expect 1 sequence in 100 sequences to be rejected by the test, if the sequence is in fact random. So, if *P-value* ≥ 0.01 we consider the string as random with a confidence of 99%; if *P-value* < 0.01 we consider the sequence as non-random with a confidence of 99%.

Example 3.1.2.
Similarly, if α is chosen to be equal to 0.001, if *P-value* ≥ 0.0001 we consider the string as random with a confidence of 99.9% and, if *P-value* < 0.001 we consider the sequence as non random with a confidence of 99.9%. Note that $\alpha = 0.001$ would indicate that we expect 1 sequence in 1000 to be rejected by the test even if the sequence is in fact random.

As we have seen the significance level we a-priori chose is crucial for our evaluation: a too high value could permit "not-so-good" random number generators to pass tests, even if it doesn't generate random numbers but, on the other hand, a too low α could be too compelling and reject even some random sequences.

Since it is not possible to judge a RNG by testing only a sequence, it's important to make a number of tests that ensures us a good level of confidence. As we will see

in section 4.1, the key to reduce the probability of falling on a *type I* or *type II* error, and to have a confidence level which depends as much as possible on the chosen level of significance α is to increase the number of tested sequence to perform a second level test. NIST recommends to perform each test on $n \sim 1000$ sequences to obtain a level of significance quite close to 1%; as we'll see in the following, since we'll inspect asymptotic properties,these sequences would have a length on the order of 10^6 digits. Figure 3.1 sows an example on the relation that occurs between α, β and the number of tested sequence n: the plot is made for smaller sample size, since it produces a more clear representation of the phenomenon.

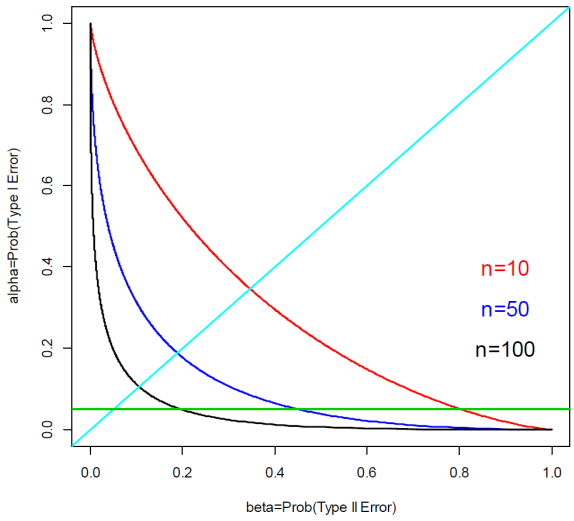

Figure 3.1: Connection between power of the test β, level of significance α and sample size n. The horizontal line corresponds to $\alpha = 0.01$ and the 45-degree line corresponds to the minimum of $\alpha + \beta$. *Credits to [35].*

3.1.1 χ^2 goodness-of-fit test and incomplete gamma function

As we said, each test inspect frequencies of observed objects with the expected statistic in random sequences. One of the most common test is the so called chi-squared test (χ^2). Once we defined the statistic to inspect, let's suppose that T leads every observation to fall into one of k categories. We take N independent observations: let π_i be the expected probability that each observation falls into i-th category, and let Y_i be the observed statistic of observations that actually fall into that i-th category, for $i = 1, \ldots, k$. We can define, and evaluate, the following statistic:

$$T = \sum_{i=1}^{k} \frac{(Y_i - N\pi_i)^2}{N\pi_i}. \tag{3.1}$$

Remark 3.1.1.
By construction we have

$$Y_1 + Y_2 + \cdots + Y_{k-1} + Y_k = N$$
$$\pi_1 + \pi_2 + \cdots + \pi_{k-1} + \pi_k = 1$$

Note that $Y_1, Y_2, \ldots, Y_k$ are not completely independent, indeed we can compute Y_k using the previous $k-1$ values. In this case we say that our statistic has $k-1$ *degrees of freedom*, i.e. the number of degrees of freedom is one less the number of chosen categories. To evaluate this statistic, we partition the original sequence $\epsilon = \epsilon_1 \ldots \epsilon_n$ of length n into N sub-strings, each of length M (if the length of the sequence can't be exactly divided by chosen M, we can discard the last part since we'll inspect very long strings), obtaining:

$$\epsilon = \epsilon_{11} \ldots \epsilon_{1M} | \epsilon_{21} \ldots \epsilon_{2M} | \ldots | \epsilon_{N1} \ldots \epsilon_{NM}$$

For each of these sub-strings we evaluate the frequencies $Y_1, \ldots, Y_k$ of the corresponding statistic, as described above.

We can now evaluate the statistic in (3.1) which, under the randomness hypothesis, has an approximate χ^2-distribution with $k-1$ degrees of freedom since it's a sum of squares of independent random variables.

We recall, now, the definition of the Euler's gamma function and of the incomplete gamma functions in order to provide a property of the P-value:

Definition 3.1.5 (Euler's gamma function)**.**

$$\Gamma(s) = \int_0^\infty e^{s-1} e^{-t} dt$$

Definition 3.1.6 (Incomplete gamma functions)**.**
The upper incomplete gamma function is defined as

$$\Gamma(s, x) = \int_x^\infty t^{s-1} e^{-t} dt$$

and the lower incomplete gamma function is

$$\gamma(s, x) = \int_0^x t^{s-1} e^{-t} dt.$$

Proposition 3.1.1.
The complementary of the P-value can be expressed in terms of the lower incomplete gamma function and of the Euler's gamma function as follows:

$$\lim_{n\to\infty} P(T \leq v) = \gamma\left(\frac{k-1}{2}, \frac{v}{2}\right) \Bigg/ \Gamma\left(\frac{k-1}{2}\right) \tag{3.2}$$

where $k-1$ is the number of degrees of freedom, as above.

Proof.
First of all we show that the probability that $Y_1 = y_1, \ldots, Y_k = y_k$ can be seen as

$$\frac{n!}{y_1! \ldots y_k!} p_1^{y_1} \ldots p_k^{y_k} \tag{3.3}$$

where $p_1, \ldots, p_k$ are the theoretical probabilities.
Indeed, since each Y_s counts the number of an event, we can assume that the Y_s has the value y_s with the Poisson probability

$$P(Y_s = y_s) = \frac{e^{-np_s}(np_s)^{y_s}}{y_s!}.$$

The Y's are independent, so

$$P((Y_1, \ldots, Y_k) = (y_1, \ldots, y_k)) = \prod_{s=1}^{k} \frac{e^{-np_s}(np_s)^{y_s}}{y_s!},$$

and the probability that $Y_1, \ldots, Y_k = n$ will be

$$\sum_{\substack{y_1+\cdots+y_k=n \\ y_1,\ldots,y_k \geq 0}} \prod_{s=1}^{k} \frac{e^{-np_s}(np_s)^{y_s}}{y_s!} = \frac{e^{-n}n^n}{n!}.$$

We can assume, now, that $Y_1, \ldots, Y_k$ are independent except for the condition that $Y_1 + \cdots + Y_k = n$, then the probability that $(Y_1, \ldots, Y_k) = (y_1, \ldots, y_k)$ is

$$\left(\prod_{s=1}^{k} \frac{e^{-np_s}(np_s)^{y_s}}{y_s!} \right) \Bigg/ \left(\frac{e^{-n}n^n}{n!} \right)$$

which equals (3.2). We can therefore consider $Y_1, \ldots, Y_k$ as independent random variables with Poisson distribution, except for the condition that their sum is fixed, and equals n. We make the change of variables

$$Z_s = \frac{Y_s - np_s}{\sqrt{np_s}},$$

so that $V = Z_1^2 + \cdots + Z_k^2$, and the condition $Y_1 + \cdots + Y_k = n$ becomes

$$\sqrt{p_1} Z_1 + \cdots + \sqrt{p_k} Z_k = 0. \tag{3.4}$$

Let's consider the $(k-1)$-dimensional space S of all vectors $(Z_1, \ldots, Z_k)$ such that (3.4) holds. Using the Central Limit theorem, each Z_s has approximately a normal distribution for large value of n; therefore points in a differential volume $dz_2, \ldots, dz_k$ of S occur with

probability approximately[1] proportional to $exp(-(z_1^2 + \cdots + z_k^2)/2)$.
The probability that $V \leq v$ becomes

$$\frac{\int_{(z_1,\ldots z_k)\in S \wedge z_1^2+\cdots+z_k^2 \leq v} exp(-(z_1^2 + \cdots + z_k^2)/2) dz_2 \ldots dz_k}{\int_{(z_1,\ldots,z_k)\in S} exp(-(z_1^2 + \cdots + z_k^2)/2) dz_2 \ldots dz_k}. \tag{3.5}$$

The hyperplane (3.4) passes through the origin of the k-dimensional space, so the numerator in (3.5) is an integration over the interior of a $(k-1)$-dimensional hypersphere centered at the origin. Applying a generalized change of coordinates, using polar coordinates with radius χ and angles $\omega_1, \ldots, \omega_{k-2}$, we can transform (3.5) into

$$\frac{\int_{\chi^2 \leq v} e^{-\chi^2/2} \chi^{k-2} f(\omega_1, \ldots, \omega_{k-2}) d\chi d\omega_1 \ldots d\omega_{k-2}}{\int e^{-\chi^2/2} \chi^{k-2} f(\omega_1, \ldots, \omega_{k-2} d\chi d\omega_1 \ldots d\omega_{k-2}}$$

for some function f. Now, the integration over the angles $\omega_1, \ldots, \omega_{k-2}$ gives a constant factor from numerator and denominator that cancels together. Finally, we obtain the formula

$$\frac{\int_0^{\sqrt{v}} e^{-\chi^2/2} \chi^{k-2} d\chi}{\int_0^{\infty} e^{-\chi^2/2} \chi^{k-2} d\chi} \tag{3.6}$$

which approximates the probability that $T \leq v$.
Substituting $t = \chi^2/2$ the integral can be expressed in terms of the incomplete gamma function, obtaining:

$$\lim_{n\to\infty} P(V \leq v) = \gamma\left(\frac{k-1}{2}, \frac{v}{2}\right) \Big/ \Gamma\left(\frac{k-1}{2}\right). \tag{3.7}$$

□

Remark 3.1.2.
In the following we'll use the symbol χ^2 to express the statistic T in (3.1).

Since we defined the empirical significance level as the probability of T to exceed the observed value, to compute the *P-value*, we need the complementary result, which is obtained by substituting the lower incomplete gamma function by the upper one. We can now define the

Definition 3.1.7 (Complementary incomplete gamma function)**.**
The complementary incomplete gamma function igamc is defined as

$$igamc(s, x) = 1 - \frac{\gamma(s, x)}{\Gamma(s)} = \frac{\Gamma(s, x)}{\Gamma(s)}.$$

[1] This step makes the result not an exact value, but only an approximation for large value of n.

Remark 3.1.3.
Using the definition above, we can finally write a simple formula for the *P-value*, in the case of χ-squared statistics, as

$$P\text{-}value = igamc\left(\frac{k-1}{2}, \frac{v}{2}\right) = \frac{1}{\Gamma(v/2)} \int_{v/2}^{+\infty} t^{(k-1)/2-1} e^{-t} dt$$

where $k-1$ is the number of degrees of freedom of the χ^2 statistic (i.e. k is the number of classes we are inspecting) and v is its empirical value, obtained from our observation of the sequence through the statistic itself.

We summarize in figure 3.2 the connection between the number of degrees of freedom, the distribution, the chosen level of significance α and the rejection region.

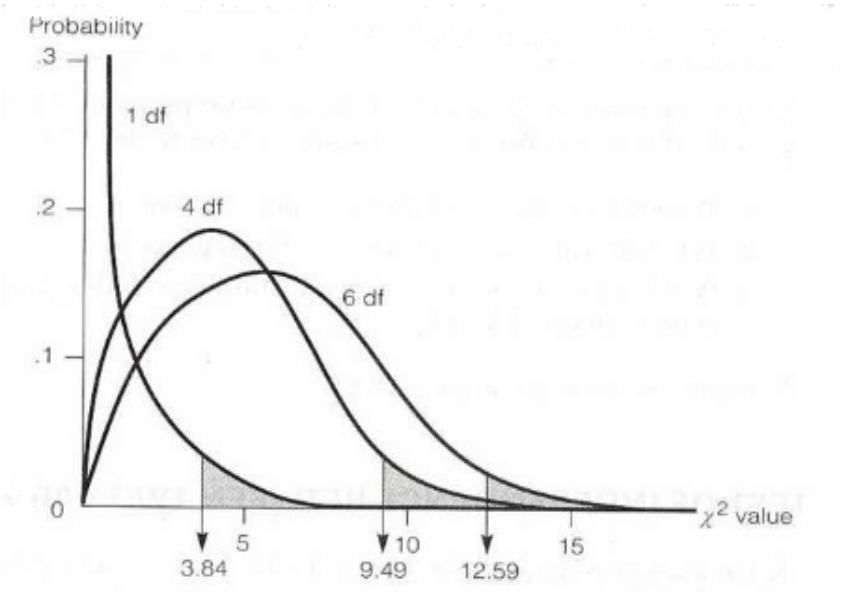

Figure 3.2: χ^2 distribution with 1, 4 and 6 degrees of freedom. Vertical arrows represents the critical value of the statistic which divides the acceptance and the rejection regions (darker in the figure), that corresponds to values with less probability than α.

As we can see, if $\chi^2 \leq \alpha$, our value will fall into the tail of the distribution which corresponds to low values of probability to be in that situation under randomness hypothesis, i.e. low *P-value*s will indicate that the sequence shows deviations from the expected behavior.

3.1.2 Complementary error function

According to central limit theorem (Theorem 2.1.2), cumulative sums of a stochastic process of independent and identically distributed random variable approaches to Normal Distribution as the number of random variables approaches to infinity. Since we can identify a random string with an i.i.d. stochastic process that theorem gives us another interpretation for studying the string. Indeed, in many tests the theoretical distribution T will have the Normal Distribution: in that cases we could find the *P-value* in a simply comparing the empirical frequencies $T(obs)$ to the Normal Distribution itself. In order do so we shall define the error function and its complementary.

Definition 3.1.8 (Error function)**.**
The error function erf (also called Gauss error function) is the function defined as

$$erf(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^{x} e^{-t^2} dt = \frac{2}{\sqrt{\pi}} \int_{0}^{x} e^{-t^2} dt \tag{3.8}$$

where the second integral comes from the fact that the function is symmetric with respect to the origin.

The error function is the probability of a random variable with normal distribution of mean 0 and variance $\frac{1}{2}$ of falling in the range $[-x, x]$.

Remark 3.1.4.
The cumulative distribution function Φ of the Standard Normal Distribution and the error function are closely related, indeed we have

$$\Phi(x) = \frac{1}{2}\left[1 + erf\left(\frac{x}{\sqrt{2}}\right)\right].$$

In the case of a generic normal distribution X with cumulative distribution function F, mean μ and variance σ^2 we can still use this relation in the form

$$F(x) = \Phi\left(\frac{x-\mu}{\sigma}\right) = \frac{1}{2}\left[1 + erf\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right]$$

since $Y = (X - \mu)/\sigma$ has a Standard Normal Distribution.

Since we'll be interested in defining in which sense this result is worse than the observed one, we'll use the following

Definition 3.1.9 (Complementary error function)**.**
The complementary error function $erfc$ is defined as

$$erfc(x) = 1 - erf(x) = \frac{2}{\sqrt{\pi}} \int_{x}^{\infty} e^{-t^2} dt$$

The complementary error function itself, is related with another function which represents the tail distribution of the Standard Normal Distribution, the

Definition 3.1.10 (Q-function)**.**
The Q-function is defined as

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty exp\left(-\frac{u^2}{2}\right) du$$

$Q(x)$ is the probability that a standard normal random variables takes a value larger than x. So if Y is a normal random variable with mean μ and variance σ^2, then $X = (Y - \mu)/\sigma$ has a Standard Normal Distribution and we have

$$P(Y > y) = P\left(X > \frac{x - \mu}{\sigma}\right) = Q(x),$$

which is exactly the formulation of the *P-value* we defined in (3.1.4).

From the definition of the Q-function, and observing that it's an even function, we can easily see the relation

$$Q(x) = 1 - Q(-x) = 1 - \Phi(x)$$

where Φ is the cumulative distribution function of the Standard Normal Distribution.

With the change of variables $u^2 = 2t^2$ in the definition of the Q-function, we also obtain that

$$Q(x) = \frac{1}{2}\left(\frac{2}{\sqrt{\pi}} \int_{x/\sqrt{2}}^\infty exp(-t^2)dt\right) = \frac{1}{2} - \frac{1}{2} erf\left(\frac{x}{\sqrt{2}}\right) = \frac{1}{2} erfc\left(\frac{x}{\sqrt{2}}\right)$$

which gives us a relation between the Q-function and the complementary error function.

Remark 3.1.5.
According to the above considerations, in tests where statistic has a normal distribution, we'll compute the *P-value* using the complementary error function $erfc$.
In these cases the *P-value* will be computed, up to multiplicative constants, as

$$P\text{-}value = erfc\left(\frac{v - \mu}{\sigma}\right), \tag{3.9}$$

where v is the observed statistic, μ is it's mean and σ^2 the corresponding variance.

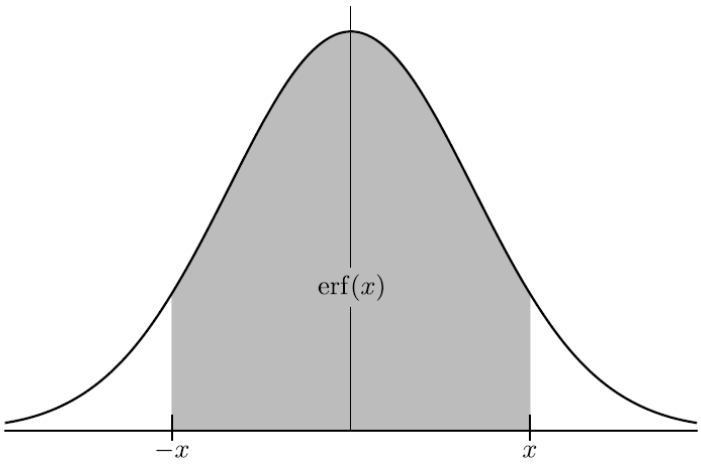

Figure 3.3: The error function erf can be seen as the probability that a Gaussian random variable falls into the into the $[-x, x]$ interval. Conversely, the complementary error function $erfc$ represents the probability that a Gaussian random variable lies outside that interval. *See [14].*

3.2 Tests

In this section we'll study the statistical test suite proposed by the National Institute of Standard and Technology (NIST) to validate Random Number Generators: the suite consists on 15 tests, each of them inspects the sequence from a different point of view, trying to detect a different weakness of the RNG. We decided to analyze NIST statistical tests since other test suites, such as the one proposed by the Bundesamt für Sicherheit in der Informationstechnik (BSI), the DieHard test suite or the Crypt-X package, are mostly based on that one.

The complexity, and in a certain sense the strength, of tests increases from first tests to the last ones so, then even if initial tests could appear trivial, their extremely low computational cost allows to use them to detect most common weakness of Random Number Generators which, in general, are not visible to the naked eye, due to the length of analyzed sequences.

In the following subsections we'll analyze how tests algorithmically work and, for each test, we'll study the theory behind it. After that we'll try to understand which is the best way to use these tests, if some of them are more useful then others and if we can choose a battery (or some) to make the testing the most computationally efficient.

First of all we have to make some hypothesis on random binary sequence to be tested, which are based on basic ideas of randomness:

- Uniformity: as we have seen in section (2.2.1) the probability of zeros and ones has to be the same and it's exactly 1/2 and this occurs at any point of the sequence, regardless of the fact the it has been generated from a true random generator or from a pseudo one. That implies that the expected number of zeros or ones, is $n/2$, where n is the sequence length;

- Scalability: if a sequence is random, then we can extract from it any sub-sequence and it still has to be random and to pass tests itself;

- Consistency: the behavior of a Random Number Generator has to be consistent with his starting value (seed). So, to obtain reliable information, it's necessary to test RNG's outputs obtained from different seeds (with respect to PRNG, since a good TRNG should not have a predictable seed) .

The 15 tests are:

1. Frequency (Monobit) test;
2. Frequency test within a block;
3. Runs test;
4. Test for the 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 (Cusums) test;
14. Random excursion test;
15. Random excursion variant test.

The theory and the results behind most of the tests are based on asymptotic properties or continue distributions, so that reference distributions are derived from the assumption that the sequence length is large (all NIST, BSI and DieHard recommend a length of the order from 10^3 to 10^7). Most test can also be computed for small values of n, but results could be inappropriate: in this case it should be required an exact distribution that,

in most cases, is too hard to determine; anyway, better to explain each test we'll use examples taken from [64] (note that several of them don't satisfy hypothesis on length: the only purpose is to make the test more clear).

We set, here, some notations that will be common to all tests, unless otherwise stated:

- $\epsilon = (\epsilon_1, \ldots, \epsilon_n)$ will be the sequence of bit, where $\epsilon_i = \{0, 1\}\ \forall i$ and n is the legth of the sequence;
- some tests require to partition the sequence into sub-blocks: typically, M will be the length of each block. The $n - NM$ remaining bits will usually be discarded, without loss of strength for the test, as stated in the assumption of scalability; note that for Maurer's universal test (test 9) the sequence legth will be indicated by L, as Maurer did in his original paper [50];
- χ^2 will stand for the the chi-squared statistic, as defined in section (3.1.1);
- K will be the number of degrees of freedom, when it has to be specified as an input of the test itself;
- α will be the significance level, as in (3.1.2), and will be taken equal to 0.01 within the examples.

3.2.1 The frequency (Monobit) test

Under randomness hypothesis H_0, the probability of zero and one is the same then, for a truly random sequence, the number of zeros and ones has to be nearly the same. The test inspects the entire sequence and evaluate the proportion of zeros and ones. If a sequence doesn't pass this simple test it won't pass the others with high probability, since this provides a great evidence of non-randomness. The following algorithm is consistent for any sequence length, but to be significant an $n > 100$ is recommended.

Algorithmic description

1. Convert the binary sequence ϵ into a corresponding ± 1 sequence X: this can be efficiently done by $X = 2\epsilon - 1$.
2. Compute the proportion of ± 1, which consists of $0, 1$ in the original string, by adding together bits of the sequence: $S_n = X_1 + \cdots + X_n$.
3. Compute the test statistic $s_{obs} = \frac{|S_n|}{\sqrt{n}}$, which for large values of n has a half normal distribution, since $\frac{S_n}{\sqrt{n}}$ has a normal distribution.
4. Compute $P\text{-}value = erfc\left(\frac{S_{obs}}{\sqrt{2}}\right)$.

Interpretation of results

A small *P-value* will occur when there is disproportion between the number of zeros and ones: this corresponds to large values of the statistic S_{obs} (in particular if there are too many ones $S_n >> 1$ and if there are too many zeros $S_n << -1$). If the sequence is random S_{obs} has to be near zero, since $+1$ and -1 will cancel one another.

An example

$\epsilon = 1011010101$ then $n = 10$.

1. $X = 1, -1, 1, 1, -1, 1, -1, 1, -1, 1$.
2. $S_n = 1 - 1 + 1 + 1 - 1 + 1 - 1 + 1 - 1 + 1 = 2$.
3. $s_{obs} = \frac{|2|}{\sqrt{10}} = 0.632455532$.
4. $P\text{-}value = erfc\left(\frac{0.632455532}{\sqrt{2}}\right) = 0.5270809 > \alpha$, so that the sequence is considered to be random.

$P\text{-}value = 0.527089 > 0.01$, then we don't reject the hypothesis of randomness and proceed with more sophisticated tests.

Mathematical background

Under randomness hypothesis H_0 the sequence can be considered as the realization of a Bernoulli process, i.e. a sequence of random variables where the probability of the outcomes (0 or 1) equals to $\frac{1}{2}$. The test is a direct application of the Central Limit theorem (Theorem 2.1.2) in the special case of the

Theorem 3.2.1 (De Moivre-Laplace)**.**
Let $X_1, X_2, \ldots, X_n$ be a sequence of Bernoulli random variables and let $S_n = X_1 + \cdots + X_n$ be their sum.
For each $a, b \in \mathbb{R}$ with $a < b$ we have

$$\lim_{n\to\infty} P\left(a \leq \frac{S_n}{\sqrt{n}} \leq b\right) = \frac{1}{\sqrt{2\pi}} \int_a^b e^{-\frac{x^2}{2}} dx.$$

In other words, the Normal distribution is a good approximation for the random variable defined by S_n, if n is sufficiently large.

The approximation provided by the theorem is used to assess the closeness of the fraction of 1's (and 0's) to $\frac{1}{2}$.
To follow the definition of the *P-value*, we can use the theorem to compute

$$\lim_{n\to\infty} P\left(\frac{S_n}{\sqrt{n}} \leq z\right) = \frac{1}{\sqrt{2n}} \int_{-\infty}^{z} e^{-\frac{x^2}{2}} dx = \Phi(z)$$

which, for positive z, implies that

$$P\left(\frac{|S_n|}{\sqrt{n}} \leq z\right) = 2\Phi - 1.$$

In our case z will be the observed value of the statistic $s_{obs} = |X_1 + \cdots + X_n|/\sqrt{n}$ and the *P-value* can be computed used the complementary error function $erfc(s_{obs}/\sqrt{n}) = 2[1 - \Phi(s_{obs})]$.

3.2.2 Frequency test within a block

By the scalability property, under hypothesis of randomness, we have that each subsequence of the random sequence has to be random itself so, if we split ϵ into sub-blocks, the proportion of ones (and zeros) has to be maintained. This test inspects whether the frequency of ones in each M-bit block of the sequence is approximately $M/2$. Note that if M is chosen to be equal to 1 this test degenerates to the first test, the Frequency test.

Algorithmic description

1. Partition the input sequence into $N = \lfloor n/M \rfloor$ sub-blocks and discard the surplus.
2. Compute the proportion of ones into each $M - bit$ block, by

$$\pi_i = \frac{1}{M}\sum_{j=1}^{M} \epsilon_{j+M(i-1)} \qquad \text{for } i = 1, \ldots, M.$$

3. Evaluate the χ-squared statistic for the entire sequence by

$$\chi^2(obs) = 4M\sum_{i=1}^{M}(\pi_i - \frac{1}{2})^2$$

.

4. Compute the *P-value* using the incomplete gamma function

$$P\text{-}value = igamc\left(\frac{N}{2}, \frac{\chi^2(obs)}{2}\right).$$

Remark 3.2.1.
The partition implies that M and N should be such that $n \geq MN$. NIST also recommends $M \geq 20, M > 0.01n$ and $N < 100$.

Interpretation of results

A small *P-value* will occur when there is a local deviation from the proportion of zeros and ones within at least one of the blocks (i.e. there are too many zeros or too many ones).

An example

$\epsilon = 0110011010$ then $n = 10$ and we choose $M = 3$.

1. Partitioning the sequence into $N = \lfloor \frac{n}{M} \rfloor = 3$ we obtain the blocks 011, 001 and 101. We discard the final 0.
2. Computing the partition of ones we obtain $\pi_1 = \frac{2}{3}, \pi_2 = \frac{1}{3}$ and $\pi_3 = \frac{2}{3}$.
3. The observed statistic will be $\chi^2(obs) = 4{\cdot}3{\cdot}((2/3 - 1/2)^2 + (1/3 - 1/2)^2 + (2/3 - 1/2)^2) = 1$.
4. *P-value* $= igamc\left(\frac{3}{2}, \frac{1}{2}\right) = 0.801252 > \alpha$, so the sequence is random.

Mathematical background

As for the Monobit test (3.2.1), this test is focused on the proportion of zeros and ones but looks for *local* deviation from the expected frequency, which is $\frac{1}{2}$. The main theoretical base for this test is the Central Limit theorem (Theorem 2.1.2) but, since we divided the sequence into sub-strings, we can compare expected probabilities and observed frequencies by the χ-squared statistic, as described in section (3.1.1), which will have a number of degrees of freedom equal to the number of sub-strings and we can evaluate the *P-value* using the incomplete gamma function.

3.2.3 Runs test

For this test (and for the "Longest run of ones within a block" test) we shall give the following

Definition 3.2.1 (Run)**.**
An uninterrupted sequence of identical bits, bounded after and before by a bit of the opposite value, is said to be a run. The length of a run is the number of equal bits which form a run; we call a run of length k a k-run. We call the change from zero to one, or vice versa, an oscillation.

Example 3.2.1.
The sequence 0111000011 is formed, in the order, by a 1-run of zeros, a 3-run of ones, a 4-run of zeros and a 2-run of ones, so it has 4 runs. Note that the number or runs equals the number of shifts between 0's and 1's plus one, and vice versa.

This test focuses on the number of runs of both ones and zeros to determine if they are nearly the same as for a random sequence. In particular, the number of runs corresponds to the frequency of oscillation between zeros and ones: if we have a large number of runs we'll have very frequent oscillations and vice versa. Even if they're obviously connected, this test is not interested on the length of runs, which will be the focus of the subsequent one.

Algorithmic description

1. Compute the proportion of ones through the entire sequence:

$$\lambda = \frac{1}{n}\sum_{j=1}^{n}\epsilon_j.$$

2. Evaluate the test statistic

$$V_n(obs) = \sum_{k=1}^{n-1} r(k) + 1$$

where $r(k)$ is the function that count jumps between different bits

$$r(k) = \begin{cases} 0 & \text{if } \epsilon_k = \epsilon k + 1 \\ 1 & \text{otherwise} \end{cases} \tag{3.10}$$

3. Compute *P-value* as

$$P\text{-}value = erfc\left(\frac{|V_n(obs) - 2n\lambda(1-\lambda)|}{2\sqrt{2n}\lambda(1-\lambda)}\right).$$

Interpretation of results

A small *P-value* will occur when there is a great number of runs or when runs are too few: this corresponds, respectively, to large or small value of the statistic V_n. In particular a large value of V_n would indicate that the oscillation is too fast and a small value that is too slow.

Example 3.2.2.
The string 0101010101010101, which is obviously non-random, has too fast oscillation. A 300-bit sequence formed, in the order, by 100 ones, 73 zeroes, 127 ones would have three runs, whereas we would expect a number of runs close to 150 for a random sequence.

An example

Let's take the 10-bit string $\epsilon = 1001101011$, so that $n = 10$.

1. The proportion of ones is $\lambda = \frac{1}{10} \cdot (1+0+0+1+1+0+1+0+1+1) = \frac{6}{10} = \frac{3}{5}$.
2. $V_n(obs) = (1+0+1+0+1+1+1+1+0)+1 = 7$.
3. $P\text{-}value = erfc\left(\frac{7-(2\cdot 10\cdot 3/5\cdot(1-3/5))}{2\cdot\sqrt{2\cdot 10}\cdot 3/5\cdot(1-3/5)}\right) = 0.147232 > \alpha$, so the sequence is considered to be random.

Mathematical background

The main basis of this test is the distribution of the total number of runs V_n, which is computed by the observation of oscillation among zeros and ones. Let's suppose to have a sequence of length $n = n_1 + n_2$, where n_1 is the number of 1's and n_2 is the number of 0's; respectively we call r_1 the number of runs of ones and r_2 the number of runs of zeros, so that $r = r_1 + r_2$ is the total number of runs, then we need the mean and variance of the associated distribution V_n.

As we said, V_n can be considered as the sum of indicator variables, by defining the random variables which corresponds to function (3.10) as

$$I_k = \begin{cases} 1 & \text{if the k-th element} \neq \text{the (k-1)-th element} \\ 0 & \text{otherwise} \end{cases}$$

so that $V_n = 1 + I_2 + \cdots + I_n$ where I_k are Bernoulli random variables with parameter $p = n_1 n_2 / \binom{n}{2}$, and their mean can be easily seen to be

$$E[I_k] = E[I_k^2] = \frac{2n_1 n_2}{n(n-1)}.$$

V_n is a combination of I_k so, by linearity, we have

$$E[V_n] = 1 + \sum_{k=2}^{n} E[I_k] = 1 + \frac{2n_1 n_2}{n_1 + n_2} \tag{3.11}$$

$$\begin{aligned} var(V_n) &= var\left(\sum_{k=2}^{n} I_k\right) = (n-1)var(I_k) + \sum_{2\leq j\neq k\leq n} cov(I_j, I_k) \\ &= (n-1)E[I_k^2] + \sum_{2\leq j\neq k\leq n} E[I_j I_k] - (n-1)^2[E[I_k]]^2 \end{aligned} \tag{3.12}$$

and, evaluating the joint moments $E[I_j I_k]$ in (3.12), we find that

$$var(V_n) = \frac{2n_1n_2(2n_1n_2 - n_1 - n_2)}{(n_1 + n_2)^2(n_1 + n_2 - 1)}. \tag{3.13}$$

Since we are interested to study very large samples, we'll need the asymptotic distribution of runs: let's suppose that the sample size n tends to infinity in such a way that $n_1/n \to \lambda$ and $n_2/n \to \lambda$, with $0 < \lambda < 1$. For $n \to \infty$, (3.11) and (3.13) become

$$\lim_{n\to\infty} E[V_n/n] = 2\lambda(1-\lambda) \qquad \lim_{n\to\infty} var(V_n/\sqrt{n}) = 4\lambda^2(1-\lambda)^2.$$

Finally, we can form the standardized random variable

$$Z = \frac{V_n - 2n\lambda(1-\lambda)}{2\sqrt{n}\lambda(1-\lambda)}$$

which, for the Central Limit theorem (Theorem 2.1.2), has a Standard Normal distribution, so that we can evaluate the *P-value* using the complementary error function $erfc$.

3.2.4 Test for the longest run of ones in a block

This test is focused on the length of runs along the sequence. The sequence is divided into M-bit blocks and the test looks for the longest run of ones into each sub-block. The goal is to determine whether the length of the longest run within the sequence is consistent from what we expect from a random sequence. We shall note that an irregularity in 1's runs corresponds to an irregularity in 0's runs, so we only need to check one of them.

Algorithmic description

1. Partition the input sequence into $N = \lfloor n/M \rfloor$ sub-blocks and discard the surplus. Here, M is chosen depending to the sequence length:

Minimum n	**M**
128	8
6272	128
$7.5 \cdot 10^5$	10^4

   Note that $n < 128$ is not permitted, since the test woldn't give reliable result as it is based on asymptotic properties; furthermore, for small values of n we'd be forced to choose short-length sub-sequences, so that this test would be too close to the previous, and not really useful.

2. Depending on M choose $k+1$ classes $v_0, \ldots, v_k$ to store the number of runs of ones of fixed length:

$\mathbf{v_i}$	$\mathbf{M = 8}$	$\mathbf{M = 128}$	$\mathbf{M = 10^4}$
$\mathbf{v_0}$	≤ 1	≤ 4	≤ 10
$\mathbf{v_1}$	2	5	11
$\mathbf{v_2}$	3	6	12
$\mathbf{v_3}$	≥ 4	7	13
$\mathbf{v_4}$		8	14
$\mathbf{v_5}$		≥ 9	15
$\mathbf{v_6}$			≥ 16

   So, if for example our sequence has length $128 < n \leq 6272$, we'll have the six classes $v_0, \ldots, v_5$ where v_0 will be the number of blocks which the longest run of ones is shorter or equal to 4, v_1 the number of blocks which the longest run of ones is a 5-length run and so on.

3. Compute the length of the longest run of ones inside each block and compute $v_0, \ldots, v_k$. If we are in the case $M = 128$ and the longest run of ones in the block we are inspecting is 6, we'll increase the value v_2 by one; similarly, if the length of the longest run of ones is greater than 9, we'll increase v_5 by one, and so on.

4. Compute the χ^2-statistic comparing obtained frequencies v_i and theoretical probabilities π_i

$$\chi^2(obs) = \sum_{i=0}^{k} \frac{(v_i - N\pi_i)^2}{N\pi_i}$$

   which has k degrees of freedom.

5. Compute the *P-value* using the incomplete gamma function

$$P\text{-}value = igamc\left(\frac{k}{2}, \frac{\chi^2(obs)}{2}\right).$$

Interpretation of results

If the sequence shows clusters of ones the χ^2 statistic will be large since, as we will see, the theoretical probabilities of very-long runs are smaller then probabilities of shortest runs. On the other hand, small values of probabilities would indicate that the sequence has too short runs.

An example

Let's consider the sequence

$$\begin{aligned}\epsilon =&11001100000101010110110001001100111000000000\\ &00100100110101010001000100111101011010000000\\ &11010111110011001110011011011000010110010,\end{aligned}$$

so that $n = 128$.

1. Depending on the sequence length, $M = 8$ and we can divide the sequence into 16 sub-sequence of length 8 and we'll have 4 classes, $v_0, \ldots, v_3$.
2. We obtain 16 sub-string each with its longest run of ones, as below

sub-string	Max-run length	sub-string	Max-run length
11001100	2	00010101	1
01101100	2	01001100	2
11100000	3	00000010	1
01001101	2	01010001	1
00010011	2	11010110	2
10000000	1	11010111	3
11001100	2	11100110	3
11011000	2	10110010	2

3. Counting the number of strings into each class, we obtain
$$v_0 = 4, v_1 = 9, v_2 = 3, v_4 = 0.$$
4. $\chi^2 = 4.882457$.
5. *P-value* $= igamc\left(\frac{3}{2}, \frac{4.882457}{2}\right) = 0.180609 > \alpha$, so the sequence is accepted as random.

Mathematical background

To perform this test, and in particular to compute the χ^2 statistic, we need to know the probabilities $\pi_0, \ldots, \pi_k$ we used to compare with the empirical frequencies $v_0, \ldots, v_k$. We will need the following

Definition 3.2.2 (Moment of a random variable)**.**
Let X be a discrete random variable with probability mass function P, then

$$E[X^k] = \sum_{x \in \mathcal{V}} x^k P(X = x)$$

is the k-th moment of X.

Definition 3.2.3 (Factorial moment)**.**
Let $r \in \mathbb{N}$, the r-th factorial moment of a probability distribution or, in other words, of a random variable X with that probability distribution, is

$$\mu_r[X] = E[(X)_r] = E[X(X-1)(X-2)\dots(X-r+1)]$$

where E is the mean and

$$(x)_r = x(x-1)(x-2)\dots(x-r+1) = \frac{x!}{(x-r)!}$$

is the falling factorial, expressed by the Pochammer symbol.

Definition 3.2.4 (Moment generating function)**.**
Let X be a discrete random variable with probability mass function p and support S, then

$$M_X(t) = E[e^{tX}] = \sum_{s \in S} e^{tx} p(x)$$

is the moment generating function of X.

Definition 3.2.5 (Factorial moment generating function)**.**
Let $t \in \mathbb{C}$ and X be a real-valued random variable, the factorial moment generating function of the probability distribution of X is

$$G_X(t) = E[t^X].$$

If X is a discrete random variable, $G_X(t)$ is said to be the probability generating function of X and it's indicated by $G(t)$.

We shall also make some remarks on properties of the factorial moment, for the proofs of which we refer to [18].

Remark 3.2.2.
Useful facts are:

i) A trivial observation is that the 1-st (factorial) moment is the expected value.

ii) $M_X(t) = G_X(e^t)$.

iii) The moment generating function is related to the cumulative distribution function by

$$M_X(t) = F_X(t) = P(X \leq t) \quad t \in \mathbb{R}.$$

iv) Factorial moments and comulative distribution function are related by the formula

$$G_X(t) = F_X(t-1) = \sum_{s=0}^{\infty} \frac{t^s}{s!} \mu_{[s]}.$$

If X is a Binomial random variable of parameters n and p. it holds

Remark 3.2.3.

$$E[(X)_r] = (n)_r p^r.$$

To model our situation we can think to an urn, containing n balls, r_1 white and $r_2 = n - r_1$ black. We sequentially extract n balls to form a n-long binary string.

The total number of combination is

$$\binom{n}{r_1} = \binom{n}{r_2} = \binom{n}{n - r_1}.$$

We call T the total number of white runs and T_0 the total number of white runs which length is equal or greater than $m + 1$: obviously $T \geq T_0$. We make use of characteristic random variables a_i, $i = 1, \ldots, T$ such that

$$a_i = \begin{cases} 1 & \text{if the length of the } i\text{-th is} \geq m + 1 \\ 0 & \textit{otherwise} \end{cases}$$

with each a_i attached to the corresponding i-th run. So the probability that any set of runs $j_1, \ldots, j_s$ (suppose that $j_1 \leq \cdots \leq j_s \leq T$) will be all of length $(m+1)$ or more it's equal to the probability that the product $a_{j_1} a_{j_2} \ldots a_{j_s} = 1$; for any of the other case the product will be 0.

Let's take into account s of T white runs; we remove from them $s \cdot m$ balls and we impose that $r_1 - sm \geq T$, so that we'll have $(r_1 - sm)$ white balls and r_2 black balls which form T (white) runs. If now we add m balls to each of our s run, we obtain again $r_1 + r_2$ balls which form T white runs where s of them are of length $(m+1)$ or more.

By a simple combinatorial argument we obtain that all the possible arrangements to be in this situation are

$$\binom{n-1}{T-1}\binom{r_2+1}{T}.$$

With the assumption that we have a total of T white runs we can compute the expected value of having s runs which length is $\leq m + 1$, obtaining that

$$E[a_{j_1} \ldots a_{js} | T] = E[a_{j_1} | T] \cdot \cdots \cdot E[a_{j_s} | T] = \binom{r_1 - sm - 1}{T - 1} \Big/ \binom{r_1 - 1}{T - 1} \tag{3.14}$$

where the first identity comes from the fact that a_{j_i}'s are independent (then the expected value of the product is the product of expected values) and the second ones comes from

a direct calculation.

By remark (3.2.3) we also obtain that the s-th factorial moment of T_0, which is a Binomial random variable, is

$$E[(T_0)_s|T] = (T)_s E[a_{j_1} \dots a_{j_s}|T]. \tag{3.15}$$

Using the Total Probability theorem and equation (3.14), to obtain the v-th factorial moment we just have to multiply (3.15) by $p(T)$ and summing over all T, obtaining

$$\mu_{[v]} = \sum_T p(T) E[(T_0)_v|T] = (r_2+1)_v \binom{r - v(m+1)}{r_2} \Big/ \binom{n}{r_2}$$

where (analyzing favorable cases and total cases)

$$p(T) = \binom{r_1 - 1}{T-1}\binom{r_2+1}{T} \Big/ \binom{r_1+r_2}{r_2}.$$

Finally, by remark (3.2.2) we obtain that the probability to have an m-run is

$$P(v \le m|r_1) = P(T_0 = 0) = \sum_{t=0} \frac{(-1)^t \mu_{[t]}}{t!} = \binom{n}{r_2}^{-1} \sum_{t=0} (-1)^t \binom{r_2+1}{t}\binom{r - t(m+1)}{r_2}.$$

Turning back to our case, and using the notation of the test, we can use this result to obtain (by the Total Probability theorem)

$$P(v \le m) = \sum_{r=0}^{M} \binom{M}{r} P(v \le m|r) \frac{1}{2^M}.$$

So, now, we can evaluate probabilities $\pi_0, \dots, \pi_k$ of our chosen classes and precompute them to embed in the test.

Example 3.2.3.
If we divide the sequence into 8-bit block (i.e. $M = 8$) the theoretical probabilities will be

Class	Probability
$\{v \le 1\}$	$\pi_0 = 0.2148$
$\{v = 2\}$	$\pi_1 = 0.3672$
$\{v = 3\}$	$\pi_2 = 0.2305$
$\{v \ge 4\}$	$\pi_3 = 0.1875$

It's important to observe that average cases have bigger probabilities than extreme ones, and the phenomenon is even more observable for bigger M (for example if $M = 10^4$ we'll have $\pi_0 = 0.0882$, $\pi_2 = 0.2483$ and $\pi_6 = 0.0727$): that is the behaviour one expects from a random sequence.

3.2.5 Binary matrix rank test

This test analyzes the rank of disjoint matrices created from fixed-length sub-strings of the binary sequence. The purpose is to check whereas this rank is consistent from what expected from a random sequence, i. e. from truly random binary matrices. Since the rank corresponds to independence of both rows and columns, this examination leads to examine linear dependencies through bits of each sub-sequence.

As well as in the NIST statistical test suite, the binary matrix rank test is also part of the DIEHARD battery of test [43], and actually comes from it.

Remark 3.2.4.
For this test, we need one more parameter Q, so that the length of sub-blocks will be $M{\cdot}Q$; since the theory of the test, analogously to others, is based on asymptotic properties, M and Q has to be chosen such that $n \geq 38MQ$ so that at least 38 matrices are considered. Since if $M \neq Q$ we certainly have $|M - Q|$ dependent vector, our implementation will have $M = Q = 32$, so that square matrices are created.

Algorithmic description

1. Partitioning the sequence into $M{\cdot}Q$-bit sub-sequences we obtain $N = \left\lfloor \frac{n}{MQ} \right\rfloor$ blocks and discard the surplus.
   From each block $\epsilon_{11}, \ldots, \epsilon_{1Q}, \ldots, \epsilon_{M1}, \ldots, \epsilon_{MQ}$ form an $M \times Q$ matrix, where the i-th row is given by $\epsilon_{i1}, \ldots, \epsilon_{iQ}$, for $i = 1, \ldots, M$, i.e. the first row of each matrix consists of the first Q bits of the block, the second row is formed by bits from $Q + 1$ to $2Q$, and so on till the last row, which consists of the last Q bits of the sub-sequence.

2. For each matrix compute the binary rank R_k, where $k = 1, \ldots, N$. For binary rank we mean the rank over the finite field $\mathbb{F}_2$.

3. Consider the following 3 classes

   - $F_M = \{$matrices which rank is maximum, i.e. $R_l = M\}$
   - $F_{M-1} = \{$matrices which rank is maximum -1, i.e. $R_l = M - 1\}$
   - $F_{M-2} = \{$other matrices$\}$

   and evaluate the cardinality of each set (for simplicity we'll name these cardinalities by the name of the set, so that F_M will be the number of matrices of full rank). Observe that $F_{M-2} = N - F_{M-1} - F_{M-1}$.

4. Compute the χ^2 statistic as

$$\chi^2(obs) = \frac{(F_M - 0.2888N)^2}{0.2888N} + \frac{(F_{M-1} - 0.5776N)^2}{0.5776N} + \frac{(F_{M-3} - 0.1336N)^2}{0.1336N}$$

5. Compute the P-value $= e^{-\chi^2(obs)/2}$.

Interpretation of results

Small P-values will occur when the value of the χ^2 statistic is low: this corresponds to deviations from distribution of rank of a truly random sequence. In particular, small values of F_M and F_{M-1} would indicate strong linear dependencies through the sequence. For example, if a binary sequence is created by a shift-register generator formed by less than M successive vectors, all created matrices will always have maximum rank while, for truly random data, the proportion of maximum rank should be only about 0.2888.

An example

Let's consider the sequence $\epsilon = 01011001001010101101$, so that $n = 20$, and we choose $M = Q = 3$.

1. We divide the sequence into $N = \lfloor \frac{20}{3 \cdot 3} \rfloor$ 9-bit blocs and discard the last two bits (0 and 1).
   Using the 9-bit created blocks we construct the following 3×3 two matrices:

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

2. Ranks of the two matrices are $R_1 = 2$ and $R_2 = 3$.

3. We have $F_M = F_3 = 1$, since only the second matrix has full rank, $F_{M-1} = F_2 = 1$ (the first matrix has rank=2), and $F_3 = 0$ since there isn't any matrix with lower rank.

4. $\chi^2(obs) = \frac{(1-0.2888N)^2}{0.2888N} + \frac{(1-0.5776N)^2}{0.5776N} + \frac{(0-0.1336N)^2}{0.1336N} = 0.596953.$

5. P-value $= e^{-0.596953/2} = 0.741948 > \alpha$ so that the sequence is accepted as random.

Mathematical background

To compare the empirical frequencies and the theoretical probabilities, we need to compute the distribution of ranks of random binary matrices. Let's take an $M \times Q$ binary random matrix: we are looking for the probability $P(\text{rank=r})$ that the rank of such matrix is r, with $r = 0, 1, \ldots, min(M, Q)$. We can think of our matrix as a sequence of Q binary column vectors of length M, so that we have $v_i \in \mathbb{F}_2^M$ for $i = 1, \ldots, Q$.

We recall the following

Remark 3.2.5.
Vectors $v_1, \ldots, v_n, v_{n+1}$ are independent if and only if vectors $v_1, \ldots, v_n$ are independent and $v_{n+1} \notin span(v_1, \ldots, v_n)$.
We will name A_n the event "$v_1, \ldots, v_n$ are independent" and B_n the event "$v_{n+1} \notin span(V_1, \ldots, v_n)$".

Let p_{n+1} be the probability that vectors $v_1, \ldots, v_{n+1}$ are independent, so that

$$p_{n+1} = P(A_{n+1}).$$

Events A_n and B_n are not independent so, by the definition of conditioned probability, we can compute p_{n+1} as

$$p_{n+1} = P(A_n) = P(A_n \cap B_n) = P(A_n)P(B_n|A_n). \tag{3.16}$$

By definition, a single vector is independent if and only if it is not the null vector: the probability to have a null vector is 2^{-M} (i.e. all values are fixed and equal to 0) so that

$$p_1 = 1 - 2^{-M}.$$

Each time we add an independent vector, we have to fix one more value so, by the recursion formula (3.16), we obtain that

$$p_n = \prod_{i=1}^{n} \left(1 - 2^{i-1-M}\right) = \prod_{i=0}^{n-1} \left(1 - 2^{i-M}\right) \text{ for } n \leq M. \tag{3.17}$$

An analogous argument can be used for row vectors. If we suppose, for the moment, that rows and columns are independent, we have that the probability to have both r independent row vectors and r independent column vectors is

$$\prod_{i=0}^{r-1} \left(1 - 2^{i-M}\right)\left(1 - 2^{i-Q}\right) \text{ for } r \leq min(M, Q).$$

Actually, rows and columns are not completely independent, since value we fixed for each column will be also fixed when we count the rows: first row vector will have a single

fixed value, second row vector will have 2 fixed values and so on till the desired rank r, so our probability becomes

$$\prod_{i=0}^{r-1} \frac{\left(1-2^{i-M}\right)\left(1-2^{i-Q}\right)}{\left(1-2^{i-r}\right)} \text{ for } r \leq min(M,Q).$$

Now, we have fixed the $min(M,Q) \cdot min(M,Q)$ values that concur to the rank of the matrix, but we also have to consider that values outside this sub-matrix remain free to be either 0 or 1.

Example 3.2.4.
Let's suppose that $M < Q$, and let's look for the maximum rank of the matrix, so $r = min(M,Q) = M$: by the argument above we'll have that values of the $M \times M$ sub-matrix (which values are indicated by a_{ij} for $i,j = 1,\ldots,r = M$) and other are free (indicated by *), as follows

$$\begin{pmatrix} a_{11} & \ldots & a_{1r} & * & * & * \\ \vdots & \ddots & \vdots & * & \ddots & * \\ a_{M1} & \ldots & a_{Mr} & * & * & * \end{pmatrix}$$

By direct computation one can see that these values are $r(Q+M-r)-MQ$, so that the probability to have a random matrix which rank is r is

$$P(\text{rank=r}) = 2^{r(Q+M-r)-MQ} \prod_{i=0}^{r-1} \frac{\left(1-2^{i-M}\right)\left(1-2^{i-Q}\right)}{\left(1-2^{i-r}\right)} \text{ for } r \leq min(M,Q). \qquad (3.18)$$

Note that if $r = max$ (i.e. $r = min(M,Q)$), than $r(Q+M-r)-MQ = 0$ so that the multiplicative factor is 1: extra values are free to be 0 or 1. On the other hand, if $r = 0$ we have that $r(Q+M-r)-MQ = -MQ$ and all extra values are fixed (the multiplicative factor is 2^{-MQ}.

By the use of formula (3.18) we can compute probabilities to compare with the empirical frequencies: we have $M = Q = 32$ so, by approximation, we obtain that

$$p_M \approx \prod_{j=1}^{\infty} \left(1-\frac{1}{2^j}\right) = 0.2888\ldots,$$

$$p_{M-1} \approx 2p_M \approx 0.5776\ldots,$$

$$p_{M-2} \approx \frac{4}{9}p_M \approx 0.1284\ldots$$

and all other probabilities are very small (≤ 0.005) if $M \geq 10$, so that it wouldn't be useful to include smaller classes for the statistic.

Finally, we can compute the χ^2 statistic and find the P-value.

Remark 3.2.6.
For this test the P-value is computed by the use of the exponential since, if the number of classes (i.e. the number of degrees of freedom) is low, it provides a good approximation of (3.6) and the value is very close to the customary incomplete gamma function which, in this case, would be $igamc\,(1, \chi^2(obs)/2)$. On the other hand, the use of the exponential is easier and computationally less expensive.

3.2.6 Discrete Fourier transform (Spectral) test

The scope of this test is to detect periodic features through the sequence which would indicate deviation from randomness. For periodic features we mean repetitive patterns that are near each other. To do that we use the discrete Fourier transform (DFT) of the sequence and analyze the pattern of frequencies: we fix a bound of 95% on peaks values and, to be random, we require that the number of all peaks exceeding this bound isn't significantly different from 5%.

For this test we will need the following

Definition 3.2.6 (Discrete Fourier transform)**.**
Let $x_1, \ldots, x_n$ be a succession of complex numbers. The discrete Fourier transform (DFT) of the succession is the succession $S_1, \ldots, S_n$ such that

$$S_j = \sum_{k=1}^{n} x_k e^{i2\pi \frac{(k-1)}{n} j} \tag{3.19}$$

$$= \sum_{k=1}^{n} x_k cos(2\pi \frac{(k-1)}{n} j) + i \sum_{k=1}^{n} x_k sin(2\pi \frac{(k-1)}{n} j) \tag{3.20}$$

where $i = \sqrt{-1}$ is the imaginary unit.

Algorithmic description

1. Convert the binary sequence ϵ into a corresponding ± 1 sequence X: this can be efficiently done by $X = 2\epsilon - 1$.

2. Apply a discrete Fourier transform to X, obtaining $S = DFT(X)$.
   S is a sequence of complex variables which represent periodic components of the binary sequence at different frequencies.

3. Let S' be the first half of the obtained sequence S, so that S' is a $\frac{n}{2}$-length sequence of values +1 and -1.
   Compute the modulus $M = abs(S') = |S'|$ to obtain a sequence of peak heights, as shown in figure 3.4.

4. Compute $T = \sqrt{ln\left(\frac{1}{0.05}\right) n}$ the bound of 95% on peaks value.

5. Compute $N_0 = 0.95 \cdot \frac{n}{2}$, the expected number of peaks which are less than T.

6. Compute N_1, the empirical number of peaks in M which are lass than T.

7. Compute $d = \frac{N_1 - N_0}{\sqrt{n \cdot 0.95 \cdot 0.05/4}}$.

8. Compute $P\text{-}value = erfc\left(\frac{|d|}{\sqrt{2}}\right)$.

Interpretation of results

A small *P-value* will occur when d is low: that would indicate that there are too many peaks (i.e. more than 5% of values are above the the bound T) and, correspondingly, that there are not enough values under the bound. This happens when the sequence shows too many periodic features, compared from what expected from a random sequence.

An example

Let's consider the sequence $\epsilon = 1001010011$, so that $n = 10$.

1. $X = 1, -1, -1, 1, -1, 1, -1, -1, 1, 1$.

2. $S = DFT(X) = 0, 1.618 + 1.175i, 1.382 + 4.253i, -0.618 + 1.902i, 3.618 - 2.628i, -2,$
$3.618 + 2.628i, -0.618 - 1.902i, 1.382 - 4.253i 1.618 - 1.175i.$

3. $M = 0,\ 2,\ 4.4721,\ 2,\ 4.4721.$

4. $T = 5.4733.$

5. $N_0 = 4.75.$

6. $N_1 = 5.$

7. $d = \frac{5 - 4.75}{\sqrt{10 \cdot 0.95 \cdot 0.05/4}} = 0.725476.$

8. $P\text{-}value = erfc(0.7254/\sqrt{2}) = 0.468159 > \alpha$, so the sequence is accepted as random.

Mathematical background

To perform this test we need to know what is the distribution of peaks which represents the distribution of the random variable N_1 and respectively N_0 which represents the number of peaks not exceeding the 95% threshold in the first part of the Fourier transform of the sequence. Note that we only need a half of it, since the discrete Fourier transform is symmetric.

Let n' be the length of the first half sequence ($n' = \frac{n}{2}$), we can think of N_1 as the sum of n' random variables which take on the value 1 with probability $p = 0.95$ and the value 0 with probability $q = 1 - p = 0.05$. Indeed if Y_j are random variables such that $Y_j = 1$ if $|S_j| < T$ and $|Y_j| = 0$ otherwise, $Y_j = 1$ with probability p, and $Y_j = 0$ with probability q, so that the $Y_j s$ are Bernoulli random variables: then we can write

$$N_1 = \sum_{j=1}^{n'} Y_j.$$

Since it is the sum of n' Bernoulli random variable, we can easily compute its expected value as

$$E[N_1] = \sum_{i=1}^{n'} E[Y_i] = n'E[Y_i] = n'p = \frac{n}{2}p$$

so that the mean is $\mu = 0.95 \cdot \frac{n}{2}$.
We could easily compute the variance in the same way, so that

$$Var(N_1) = \sum_{i=1}^{n'} Var(X_i) = nVar(X_i) = n'pq.$$

However Kim, Umeno and Hasegawa [36], through numerical simulation found that, even if the expected value found below is a good approximation, the variance tends to be

$$\sigma^2 = \frac{n'pq}{2} = \frac{npq}{4}$$

so that the variance we should use to perform the complementary error function is $\sigma^2 = \frac{n \cdot 0.95 \cdot 0.05}{4}$.

Let's see, now, how to find the threshold value T so that the number of peaks exceeding it is the 5% of the total: let's write the square modulus of S_j as

$$|S_j|^2 = c_j^2 + s_j^2$$

as (3.20). Under assumption of randomness of x_k (which is +1 or -1 for $k = 1, \ldots, n$) we have that c_j and s_j converge to a normal distribution whit mean zero and variance

$\sigma^2 = n/2$, so that $Y = (\frac{c_j}{\sigma})^2 + (\frac{s_j}{\sigma})^2$ converges to a χ^2 distribution with 2 degrees of freedom, since it's the sum of 2 independent standard normal random variable, so that

$$P(Y) = \frac{1}{2} exp(-\frac{Y}{2}).$$

If we set $Z = \frac{Y}{2}$, we obtain the distribution

$$P(Z) = exp(-Z).$$

We want that the number of peaks above the threshold T is the 5% of the total so, if Z_C is the upper bound of the random variable Z, we have

$$0.05 = \int_{Z_C}^{\infty} exp(-Z)dZ = exp(-Z_C)$$

so that $Z_C = -ln(0.05) = 2.995732274$.

By (3.20), we also have that $|S_j| = \sqrt{nZ}$ so, using the bound Z_C, we find that the threshold is

$$T = \sqrt{nZ_C} = \sqrt{2.995732274n}.$$

We set the bound T and we found values of the theoretical values of mean and variance, so we can find the P-value by the complementary error function using formula (3.9):

$$P\text{-}value = erfc\left(\frac{|d|}{\sqrt{2}}\right)$$

where $d = (N_1 - N_2)/\sqrt{n \cdot 0.95 \cdot 0.05/4}$.

Figure 3.4 shows the difference between frequencies of a 4096-bit generated from a good random number generator and the ones from a generator that produces periodic patterns.

3.2.7 Non-overlapping template matching test

Under assumption of randomness, if we choose a pattern, it has to have the same probability of any other pattern of the same length: this is the focus of this test and of the subsequent Overlapping template matching test. This test counts the number of occurrences of an aperiodic string and evaluate if that specific pattern occurs more or less than expected from a truly random sequence. The test analyzes m bits of the sequence (where m is the length of the chosen pattern): if the pattern is found it slides to the next m bits, otherwise it slides by only one bit. This is done partitioning the sequence into N sub-blocks of length M and the P-value is obtained by the χ^2 statistic.

We give the following

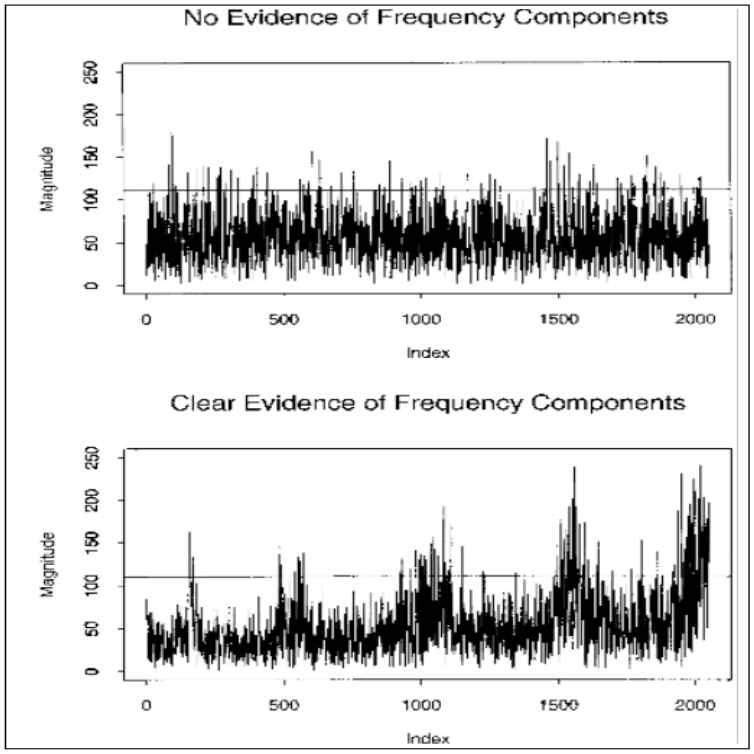

Figure 3.4: While very few magnitudes of the first sequence exceed the threshold value of 95%, represented by the horizontal line, more than 5% do in the second sequence. In the second plot a structure of magnitudes is also observable, while we can't see something like that on the first one. P-values of the first and the second sequence are, respectively, 0.8077 and 0.0001. *Credits to [64].*

Definition 3.2.7 (Aperiodic patter)**.**
We say that a pattern $B = \epsilon_1^0, \ldots, \epsilon_m^0$ is aperiodic if it doesn't exist a pattern C, shorter than B, such that $B{=}CC \ldots CC'$, where C' is a prefix of C.

Algorithmic description

First of all we choose an aperiodic template B and let m be its length.

1. The sequence is partitioned into N blocks of length M (possible surplus will be discarded).
2. For each block count the number W_j of occurrences of B inside the j-th block for $j = 1, \ldots, N$, as follows:
   - Compare the first m bits of the block with B;
   - if they're not equal, slide of 1 bit and analyze bits from second to $(m+1)$-th[2];
   - if they match, slide of m bits and analyze bits from $(m+1)$-th to $2m$-th.
3. Compute theoretical mean μ and variance σ^2 as
$$\mu = \frac{N - m + 1}{2^m} \qquad \sigma^2 = M\left(\frac{1}{2} - \frac{2m-1}{2^{2m}}\right).$$
4. Compute the observed χ^2-statistic as
$$\chi^2(obs) = \sum_{j=1}^{N} \frac{(W_j - \mu)^2}{\sigma^2}.$$
5. Compute P-value using the incomplete gamma function:
$$P\text{-}value = igamc\left(\frac{N}{2}, \frac{\chi^2(obs)}{2}\right).$$

Remark 3.2.7.
Chosen m, this test can be performed for each aperiodic template of length m: for example if $m{=}2$ the only non-periodic templates are 01 and 10, if $m = 3$ we have 001, 011, 100 and 110, but if we choose $m = 9$ or $m = 10$ we can perform the test, respectively, for 148 and 284 templates. Note that, each template provides a different P-value.

[2]For efficiency purpose is important to compare only *one* bit at a time, instead of the whole m-bits block, so that if the first bit doesn't match the comparison can straightaway proceed to the following m bits.

Interpretation of results

A small P-value (i.e. a rejection) is obtained when occurrences of the chosen template through the sequence are irregular, in particular if there are too many or too few occurrences, or if occurrences are located into restricted areas of the sequence.

An example

Let $\epsilon = 10100100101110010110$, so that $n = 20$. We take $N = 2$ and $M = 10$ and inspect the sequence for the template $B = 001$ (so $m = 3$).

1. Partitioning the sequence we obtain the two 10-bits blocks

$$1010010010, \qquad 1110010110.$$

2. Count occurrences of 001 through blocks; for the first block we find the first occurrence of B on bits 4-6, and the second one on bits 7-9: note that, as by the algorithm, bits from 5-7 and 6-8 are not examined, since we slide to bits 7-9 after the first match.
   We find $W_1 = 2$ and $W_2 = 1$ (there is only a single occurrence on bits 4-6 in the second block).

3. Theoretical mean and variance for m and M of this example are $\mu = (10 - 3 + 1)/2^3 = 1$ and $\sigma^2 = 10 \cdot \left(\frac{1}{2^3} - \frac{2\cdot 3-1}{2^{2\cdot 3}}\right) = 0.46875$.

4. $\chi^2(obs) = \frac{(2-1)^2+(1-1)^2}{0.46875} = \frac{1+0}{0.46875} = 2.133333$.

5. $P\text{-}value = igamc\left(\frac{2}{2}, \frac{2.133333}{2}\right) = 0.344154 > \alpha$ so the sequence is accepted as random.

Mathematical background

To perform this test we need to know the theoretical distribution of non-periodic patterns inside the sequence or simply their mean and variance in order to compute the χ^2-distribution.
We a-priori choose an aperiodic template $B = \epsilon_1^0, \ldots, \epsilon_m^0$ as in (3.2.7) of length m.

Example 3.2.5.
For $m = 4$ we have 6 aperiodic patterns: $0001, 0011, 0111, 1000, 1100, 1110$.
For $m = 5$ they're 12:
$00001, 00011, 00101, 01011, 00111, 01111, 11100, 11010, 10100, 1100, 10000, 11110$.

Since we chose a patter which is non-periodic, occurrences of B are non-overlapping, and can count the number of occurrences $W = W(m, n)$ through indicator variables as

$$W = \sum_{i=1}^{n-m+1} I_{\{\epsilon_{i+k-1}=\epsilon_k^0, \forall\, k=1,\ldots,m\}}.$$

Indicator functions can be seen as random variables which are m-dependent so that we can use a more general version of the Central Limit theorem (Theorem 2.1.2) for m-dependent random variables, as proved (for example) in [30] and [20], to say that the distribution of W tends to the normal distribution, when n tends to infinity, and its mean and variance are

$$\mu = \frac{n-m+1}{2^m} \qquad \sigma^2 = n\left[\frac{1}{2^m} - \frac{2m-1}{2^{2m}}\right].$$

In our case we partitioned the sequence into N blocks of length M: let $W_j = W_j(m, M)$ be the number of occurrences of the chosen pattern in the j-th block, with $j = 1, \ldots, N$ so that theoretical mean and variance are the ones referring to $W_j s$ (i.e. $\mu = E[W_j]$ and $\sigma^2 = \sigma^2(W_j)$). For large values of n, and consequently of M, W_j has a normal distribution and so the statistic

$$\chi^2(obs) = \sum_{j=1}^{N} \frac{(W_j - \mu)^2}{\sigma^2}$$

has a χ^2 distribution with N degrees of freedom, so that we can compute P-value using the incomplete gamma function in the usual way.

3.2.8 Overlapping template matching test

This test is the complementary of the previous Non-overlapping template matching test: the main idea is the same of the previous test, but the difference is on the type of template to look for (this time this will be a periodic strings) and on the way we count; in the previous test, if we had a match, we slid of the whole m bit, in this case we slide only one bit both if we don't find the pattern or if we have a match: this allows us to inspect the distribution of overlapping templates.

Algorithmic description

We choose a periodic template B of length m (we will consider a run of m ones, but the test can be performed using any assgned periodic template, by changing theoretical values).

1. Partition the sequence into $N = \lfloor \frac{n}{M} \rfloor$ blocks of length M (and discard the possible surplus).

2. For each block count the number of overlapping occurrences of B inside the j-th block, for $j = 1, \ldots, N$: the window slides each time of 1 bit, i.e. if we're analyzing bits fro k-th to $(k+m)$-th we'll next analyze bits from $k+1$ to $k+1+m$ both if patterns match or not.
   To record the number of occurrences in each block, we use classes $\nu_0, \ldots, \nu_5$: each block, after the check, if we didn't find occurrences of the template B we increment ν_0 by one, if we found 1 occurrence we increment ν_1 and so on; ν_5 is incremented if occurrence of B are 5 or more.

3. Compute values of λ and η:

$$\lambda = \frac{M - m + 1}{2^m} \qquad \eta = \frac{\lambda}{2}.$$

   This values are used to compute theoretical probabilities $\pi_0, \ldots, \pi_5$, which corresponds to $\nu_0, \ldots, \nu_5$ (if M and m are fixed, one can pre-compute these values, as we did in our implementation).

4. Compute the statistic

$$\chi^2(obs) = \sum_{i=0}^{5} \frac{(\nu_i - N\pi_i)^2}{N\pi_i}$$

   where, if $m = 9$ and $M = 1032$ as recommended by NIST, $\pi_0 = 0.364091$, $\pi_1 = 0.185659, \pi_2 = 0.139381, \pi_3 = 0.100571, \pi_4 = 0.0704323$ and $\pi_5 = 0.139865$. Obviously, probabilities for different m and M can be computed anyway, as explained in the mathematical background section of this test.

5. Compute P-value using the incomplete gamma function:

$$P\text{-}value = igamc\left(\frac{5}{2}, \frac{\chi^2(obs)}{2}\right).$$

Interpretation of results

A small P-value occurs when the number of periodic occurrences is far from the theoretical number of them, under randomness hypothesis: in particular this correspond to large values of ν_0 or ν_5 and, consequently on large values of the statistic.

An example

We test the sequence $\epsilon = 10111011110110110100011100101110111110000101101001$ (so that $n = 50$) for the template $B = 11$ (then $m = 2$), and fix $M = 10$ and $N = 5$.

1. The partition produces blocks $1011101111, 0110110100, 0111001011, 1011111000$, and 0101101001.

2. We count the number of overlapping occurrence of B through blocks. For the first block the algorithm will count as follows

Position (bit)	**Bits**	**Match of B**
1 - 2	10	
2 - 3	01	
3 - 4	11	✓
4 - 5	11	✓
5 - 6	10	
6 - 7	01	
7 - 8	11	✓
8 - 9	11	✓
9 - 10	11	✓

   For the first block the sequence has five occurrences of $B = 11$, then we increment ν_5 by one, while $\nu_0 = \nu_1 = \nu_2 = \nu_3 = \nu_4 = 0$.
   Similarly, block 2 has 2 occurrences of 11, so that $\nu_2 = \nu_2 + 1 = 1$; we have 3 occurrences in block 3, so that ν_3 is incremented; there are 4 occurrences in block 4 and one occurrence in block 5, so that after the examination of all blocks we'll have $\nu_0 = 0, \nu_1 = 1, \nu_2 = 1, \nu_3 = 1, \nu_4 = 1, \nu_5 = 1$.

3. $\lambda = (10 - 2 + 1)/2^2 = 2.25$ and $\eta = \lambda/2 = 1.125$.

4. Using λ and η we compute the theoretical probabilities which, in this example, will be different from the ones in the algorithm section since we have different M and m: we find $\pi_0 = 0.324652, \pi_1 = 0.182617, \pi_2 = 0.142670, \pi_3 = 0.106645, \pi_4 = 0.077147$ and $\pi_5 = 0.166269$ (see the Mathematical background section for formulas to compute probabilities).

$$\begin{aligned}\chi^2(obs) &= \frac{(0 - 5 \cdot 0.324652)^2}{-5 \cdot 0.324652} + \frac{1 - 5 \cdot 0.182617)^2}{5 \cdot 0.182617} + \frac{(1 - 5 \cdot 0.142670)^2}{5 \cdot 0.142670} \\ &+ \frac{(1 - 5 \cdot 0.106645)^2}{5 \cdot 0.106645} + \frac{(1 - 5 \cdot 0.077147)^2}{5 \cdot 0.077147} + \frac{(1 - 5 \cdot 0.166269)^2}{5 \cdot 0.166269} \\ &= 3.167729\end{aligned}$$

5. *P-value* $= igamc\left(\frac{5}{2}, \frac{3.167729}{2}\right) = 0.274932 > \alpha = 0.01$ and we conclude that the sequence is random.

Mathematical background

The test is based on the distribution of the number of runs of ones which length is m, that can also overlap. Let $W_j = W_j(m, n)$ be the number of those runs in the j-th block. The key of this test is the computation of the asymptotic distribution of W_j, from which we can evaluate probabilities $\pi_i,\ i = 0, \ldots, 5$. As proved, in the specific case of binary sequences, by Földes in [23] and, in the more general case of sequences created by any finite alphabet, by Chrysaphinou and Papastravridis in [16], $W_j s$ converges to a compound Poisson distribution, in particular to the so called Pólya-Aeppli distribution when $(M - m + 1)2^{-m} \to \lambda > 0$. The general Pólya-Aeppli distribution models the idea that counted objects occur in clusters, where the number of objects has a Poisson distribution (of parameter θ) and the number of objects per cluster has a geometric distribution of parameter p.

In the following it will be useful this

Definition 3.2.8 (Confluent hypergeometric function)**.**
For $x \in \mathbb{R}$ and $a, b \in \mathbb{Z}$ the confluent hypergeometric function (also called Kummer's function[3]) is

$$\begin{aligned} {}_1F_1[a; c; x] &= 1 + \frac{a}{c1!}x + \frac{a(a+1)}{c(c+1)2!}x^2 + \ldots \\ &= \sum_{j=0}^{\infty} \frac{(a)_j x^j}{(c)_j j!} \end{aligned}$$

where $(a)_j = a(a+1)\ldots(a+j-1)$ is the Pochammer's symbol for the ascending factorial.

Remark 3.2.8.
The Pólya-Aeppli distribution can be seen in two, equivalent, ways using the Poisson distribution and the geometric distribution, so that if X has a Pólya-Aeppli distribution then

$$X \sim Poisson(\theta) \bigvee Shifted\ geometric(p) \tag{3.21}$$

but also, using a non-shifted geometric distribution as

$$X \sim Poisson\left(\frac{\theta}{p}\right) \bigvee Geometric(p). \tag{3.22}$$

[3]Kummer introduced this function in 1837 as a solution to a differential equation, called the Kummer's differential equation.

Using the same parameter θ and p, is useful to define the Pólya-Aeppli distribution through its probability generating function:

$$G(z) = exp\left[\theta\left(\frac{(1-p)z}{1-pz} - 1\right)\right] \tag{3.23}$$

$$= exp\left[\theta\left(\frac{z-1}{1-pz} - 1\right)\right]$$

$$= exp\left[\frac{\theta}{p}\left(\frac{1-p}{1-pz} - 1\right)\right] \tag{3.24}$$

where expression (3.23) and (3.24) correspond respectively to models (3.21) and (3.22). By direct expansion of the probability generating function and denoting $q = 1 - p$, we can immediately obtain

$$P(X = 0) = e^{-\theta}$$

and

$$P(X = x) = e^{-\theta}p^x \sum_{j=1}^{x} \binom{x-1}{j-1} \frac{(\theta q/p)^j}{j!}, \quad x = 1, 2, \ldots \tag{3.25}$$

By the use of the confluent hypergeometric function ${}_1F_1[\cdot]$ defined in (3.2.8) we can also write

$$P(X = x) = e^{-\theta}\left(\frac{\theta q}{p}\right) p^x \, {}_1F_1[1-x; 2; -\theta q/p].$$

Using Kummer's transformation ${}_1F_1[a; b; y] = e^y \, {}_1F_1[b-a; b; -y]$ (proved by Kummer itself under the name of *Kummer's first theorem* in [41]) one can obtain

$$P(X = x) = e^{-\theta/p}\left(\frac{\theta q}{p}\right) p^x \, {}_1F_1[x+1; 2; \theta q/p].$$

From that, Kemp in [33] derived the following recurrence formula

$$(x+1)P(x = x+1) = \theta q \sum_{j=0}^{x} (x+1-j)p^{x-j}P(X = j).$$

Finally, we can use this formula to evaluate the theoretical probabilities $\pi_0, \ldots, \pi_5$ noting that, using notations of the algorithm section, $\theta = \eta = \lambda/2$ and $p = q = 1/2$, so that if X is a random variable with the compound Poisson asymptotic distribution

$$P(X = 0) = e^{-\eta},$$

$$P(X = 1) = \frac{\eta}{2}e^{-\eta},$$

$$P(X=2) = \frac{\eta e^{-\eta}}{8}(\eta+2),$$

$$P(X=3) = \frac{\eta e^{-\eta}}{8}\left(\frac{\eta^2}{6}+\eta+1\right),$$

$$P(X=4) = \frac{\eta e^{-\eta}}{8}\left(\frac{\eta^3}{24}+\frac{\eta^2}{2}+\frac{3\eta}{2}+1\right).$$

We can use above formulas to compute the probabilities $\pi_0, \dots \pi_4$ and, to evaluate the probability to have more than four overlapping runs in the block, we can complement the distribution function obtained in (3.25), obtaining that

$$P(X>x) = e^{-\eta}\sum_{j=1}^{x}\frac{\eta^j}{j!}\Delta(j,x)$$

where

$$\Delta(j,x) = \sum_{k=j}^{x}\frac{1}{2^k}\binom{k-1}{j-1}.$$

We can finally compute $P(X \geq 5)$ as

$$P(X \geq 5) = P(X=5) + P(X>5).$$

Chosen $K+1$ classes $\{X=0\}, \{X=1\}, \dots, \{X=K-1\}, \{X \geq K\}$, representing the number of overlapping runs inside blocks, one can compute respective theoretical probabilities $\pi_0, \pi_1, \dots, \pi_{K-1}, \pi_K$, using above formulas, and compute the value of the χ^2-statistic as

$$\chi^2(obs) = \sum_{i=0}^{K}\frac{(\nu_i - N\pi_i)^2}{N\pi_i}$$

where N is the number of block; after that we can compute the P-value using the incomplete gamma function, as usual.

Supposing to test sequences on the order of 10^6 bits, we implemented the test for $K=5, m=9$ and $M=1032$, as NIST did, so that theoretical probabilities derived by above formulas are

π_0	0.367879
π_1	0.183939
π_2	0.137954
π_3	0.099634
π_4	0.069935
π_5	0.140656

.

Values in the above table are the ones NIST inserted in the most recent version of their Overlapping template matching test, however Hamano and Kaneko, in [29], proved that these values are inaccurate since, setting the level of significance to 1%, the test produce a rejection rate of 98.8%, instead of the expected 99%.
They recomputed probabilities $\pi_i, i = 0, \ldots, 5$ by founding a set of recursive formulas to compute the number $T_i(n)$ of n-bit sequence with i occurrences of the template B (an m-bit run of ones), so that

$$\pi_i = \begin{cases} \frac{T_i(n)}{2^n} & \text{for} i = 0, \ldots, 4 \\ 1 - \sum_{j=0}^{4} \pi_j & \text{for} i = 5. \end{cases}$$

By the use of these formulas they computed values of the table below and, testing $4 \cdot 10^4$ sequences of length 10^6 taken from DES (used as a PRNG), showed that values proposed by NIST lead to a distribution of rejection rate which is far from the theoretical one (while, on the other hand, new values fits almost perfectly).

π_0	0.364091
π_1	0.185659
π_2	0.139381
π_3	0.100571
π_4	0.0704323
π_5	0.166269

.

NIST itself acknowledged that Hamano and Kaneko's values are more correct, but left the old ones in the implementation of the test suite. We decided to implement the test using both values: the first ones to check the implementation and compare our results with NIST's and the second to have a more reliable test for practical purposes.

3.2.9 Maurer's universal statistical test

As we said in section (2.2.2), a truly random sequence has high values of entropy and, for that reason, it couldn't be significantly compressed (i.e. compressed without losing information, contained in the sequence) by any algorithm of compression. Since 2008 a test based on a Lempel-Ziv compressor (an algorithm for data compression proposed by Ziv and Lempel in [71]) was included in the NIST statistical test suite, but then it was removed due to low efficiency, problems on implementation and, primarily, difficult on defining a statistic which can be determined and approximated. However the idea behind Maurer's universal statistical test is almost the same: it doesn't try to compress the sequence, as the Lempel-Ziv test proposed in [73] did, but analyzes the per-bit entropy of the sequence, which is strictly connected with compressibility. To achieve that, the test examines the distance (i.e. the number of bit) between matching patterns, since if there

are a lot of same pattern which are sufficiently close one-another, one can substitute the pattern with a shorter one and compress the sequence: that is the connection between distances and compression.

Maurer, in [50], states that the per-bit entropy is "the correct quality measure for a secret-key source in a cryptographic application" since it's "related to the running time of enemy's optimal key-search strategy" or to the effective key size of a cipher system. The author also report that, even if this test doesn't detect some specific defection of the sequence, it is "able to detect any one of the very general class of statistical defects that can be modeled by an ergodic stationary source with finite memory", so that this test should contain a wide number of standard statistical tests. We recall that, for this test, we will forget usual notations to conform to the ones Maurer used in the original paper, so that the length of block will be L, and the number of blocks will be $Q + K$.

Algorithmic description

The length of block L and the number of testing blocks Q is chosen depending on the sequence length, as in the following table:

n	**L**	$Q = 10 \cdot 2^L$
≥ 387840	6	640
≥ 304960	7	1280
≥ 2068480	8	2560
≥ 4654080	9	5120
≥ 10342400	10	10240
≥ 22753280	11	20480
≥ 49643520	12	40960
≥ 107560960	13	81920
≥ 231669760	14	163840
≥ 496435200	15	327680
≥ 1059061760	16	655360

After that, K is set as $K = \lfloor n/L \rfloor - Q$.

The test could be performed for all L, but choosing $L \geq 6$ would allow to have every L-bit pattern into the initialization segment and taking $L > 16$ would make the test too slow, since the initialization takes time exponential in L.

1. The sequence is split into two parts: first $Q \cdot L$-bits will form the initialization segment, and the following $K \cdot L$-bits will form the test segment: any extra bits are discarded.

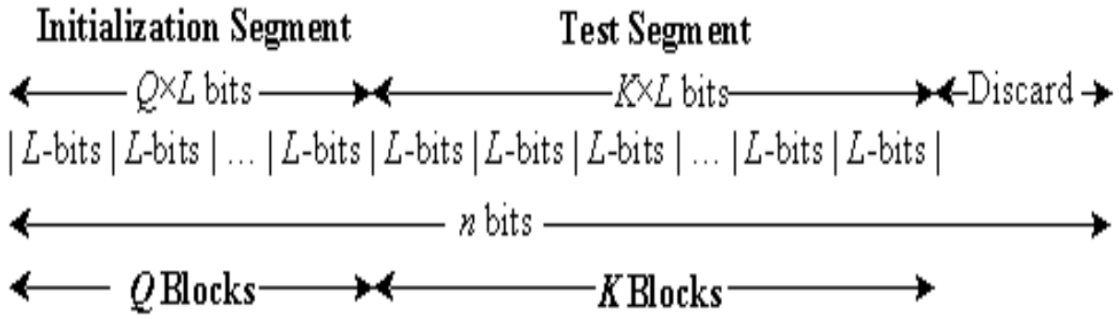

Figure 3.5: Partition of the sequence into $Q + K$ block of length L. *Credits to [64].*

2. Create a table containing each possible L-bit pattern and store the index of the last occurrence of each of these templates inside the initialization segment: in the following we will call T_j the index of the j-th template of the table. For example if the template a appears in s-th block and in t-th block, with $s < t$, store t as index of the template a.

3. Scan blocks of the testing sequence: if a block appears into the table, compute the log_2 distance between the index of the block and the index of the previous appearance of the template: i.e. if the i-th block matches the j-th template, compute $log_2(i - T_j)$. Replace the index T_j into the table with the last index of the template i. For each step sum all the distances together into a variable sum (i.e. each step $sum = sum + log_2(i - T_j)$).

4. Compute the statistic

$$f_n = \frac{1}{K} \sum_{i=Q+1}^{Q+K} log_2(i - T_j) = \frac{sum}{K}.$$

5. Compute theoretical expected value and standard deviation σ. Expected value and variance depends only on L so that we can pre-compute them:

L	expectedValue	variance
6	5.2177052	2.954
7	6.1962507	3.125
8	7.1836656	3.238
9	8.1764248	3.311
10	9.1723243	3.356
11	10.170032	3.384

L	expectedValue	variance
12	11.168765	3.401
13	12.168070	3.410
14	13.167693	3.416
15	14.167488	3.419
16	15.167379	3.421

The theoretical standard deviation is given from

$$\sigma = c(L,K)\sqrt{\frac{variance(L)}{K}}, \qquad \text{with } c(L,K) = 0.7 - \frac{0.8}{L} + \left(4 + \frac{32}{L}\right)\frac{K^{3/L}}{15}.$$

Compute P-value using the complementary error function:

$$P\text{-}value = erfc\left(\left|\frac{f_n - expectedValue(L)}{\sqrt{2}\sigma}\right|\right).$$

Remark 3.2.9.
In order to make the storing and the computation of step 2 and 3 more efficient, patterns are considered using the decimal value represented (on base 2) by the template itself. For example, if $L = 4$, the pattern "0010" is stored into the second entry of the storing table and the pattern "0111" is stored into the 7-th entry. "0000" can be stored into the last entry of the table or into the first one: in the second case everything has to be shifted by one.

Interpretation of results

A small P-value (i.e. a rejection of the sequence) occurs when the statistic of distances f_n differs significantly from the $expectedValue(L)$: that would indicate that the sequence can be significantly compressed, i.e. is not random.

An example

Let $\epsilon = 01011010011101010111$, then $n = 20$. Let's take $L = 2$ and $Q = 4$ so that $K = \lfloor n/L \rfloor - Q = \lfloor 20/2 \rfloor - 4 = 6$.

1. The sequence is split into an initialization segment 01011010 and a test segment 011101010111. No bits are discarded, since $Q \cdot L + K \cdot L = 20 = |\epsilon|$. L-bit created blocks are shown in the following tables:

Initialization segment

Block index	Content
1	01
2	01
3	10
4	10

Test segment

Block index	Content
5	01
6	11
7	01
8	01
9	01
10	11

2. Table with L-bit templates and respective index (into the initialization segment) is created:

Template	00	01	10	11
Last index	0	2	4	0

3. The sum of distances proceeds as follows:

Block 5: block 5 reports the template $'01'$, which last index is 2, so that $sum = log_2(5-2) = 1.584962501$.

Block 6: block 6 reports the template $'11'$, which last index is 0, so that $sum = sum + log_2(6-0) = 1.584962501 + 2.584962501 = 4.169925002$.

Block 7: block 7 reports the template $'01'$, which last index is 5, so that $sum = sum + log_2(7-5) = 4.169925002 + 1 = 5.169925002$.

Block 8: block 8 reports the template $'01'$, which last index is 7, so that $sum = sum + log_2(8-7) = 5.169925002 + 0 = 5.169925002$.

Block 9: block 9 reports the template $'01'$, which last index is 8, so that $sum = sum + log_2(9-7) = 5.169925002 + 0 = 5.169925002$.

Block 10: block 10 reports the template $'11'$, which last index is 6, so that $sum = sum + log_2(10-6) = 5.169925002 + 2 = 7.169925002$.

During the sum, the table of indexes is updated ad follows:

Iteration block	**L-bit value**			
	00	**01**	**10**	**11**
4	0	2	4	0
5	0	5	4	0
6	0	5	4	6
7	0	7	4	6
8	0	8	4	6
9	0	9	4	6
10	0	9	4	10

4. The test statistic $f_n = \frac{7.169925002}{6} = 1.1949875$.

5. For $L = 2$ expected value and variance can be computed: $expectedValue = 1.5374383$ and $\sigma^2 = 1.338$, then $P\text{-}value = erfc\left(\left|\frac{1.19498755-1.5374383}{\sqrt{2}\sqrt{1.338}}\right|\right) = 0.767189 > \alpha$ and the sequence is accepted as random.

Mathematical background

As we have seen in section (3.1.2), in order to perform the complementary error function $erfc$, we need to know the theoretical distribution of the statistic f_n, when data is truly random (i.e. we need to know its mean and variance).
Let $s^N = s_1, \ldots, s_N$ be the whole (random) sequence of length N, with $N = (Q+K)L$, we denote the n-th block of length L of the sequence as

$$b_n(s^N) = s_{L(n-1)+1}, \ldots, s_{Ln} \qquad \text{for } 1 \leq n \leq Q+K.$$

For $Q+1 \leq n \leq Q+K$ we can define the function that counts the distance from the previous template equal to $b_n(s^N)$ as

$$A_n(s^N) = \begin{cases} n & \text{if } \nexists\, i \leq n \text{ s.t. } b_n(S^N) = b_{n-1}(s^N), \\ min\{i \geq 1 \text{ s.t. } b_N(s^N) = b_{n-i}(s^N)\} & \text{otherwise.} \end{cases}$$

By this definition we can write the function f_n which form the statistic as

$$f_n(s^N) = \frac{1}{K} \sum_{n=Q+1}^{Q+K} log_2 A_n(s^N).$$

When it's not strictly necessary, in the following we'll neglect the input argument of function b_n, A_n, f_n and, unless specified, we'll mean that it is s^N.

We are ready to compute mean and variance of the function f_n (which can obviously be seen as a random variable): f_n is the average of several random variable, its expected value is equal to the average of the expected values, so that

$$E[f_n] = E[log_2 A_n].$$

The variance of the sum of independent random variables is the sum of variances of them, but our random variables A_n are not completely independent, so that the variance of f_n is smaller; we can write this fact as

$$var(f_n) = c(L,K)^2 \cdot \frac{var(log_2 A_n)}{K} \tag{3.26}$$

where the factor $c(L,K)$ denotes the reduction of the variance from the case of completely independent random variables.

Since we are inspecting very long strings, when computing mean and variance, it's reasonable to assume that Q tends to infinity. Let's suppose to have $U^N = U_1, \ldots, U_N$ a sequence of binary random variable with $P(U_j = 1) = P(U_j = 0) = 1/2\ \forall j = 1, \ldots, N$.

For $i \leq 1$ we can write the probability to find a distance between matching patterns of i as

$$P(A_n) = \sum_b P(b_n = b, b_{n-1} \neq b, \ldots, b_{n-i+1} \neq b, b_{n-i} = b).$$

Under randomness hypothesis the blocks b_n are independent and identically distributed, so that probability becomes

$$P(A_n = i) = \sum_b [P(b_n = b)]^2 \cdot [1 - P(b_n = b)]^{i-1},$$

but our sequence is binary and symmetric (i.e. it assumes both values 0 and 1 with probability 1/2), so that

$$P(A_n = i) = 2^{-L}(1 - 2^{-L})^{i-1}.$$

Finally, remembering that we assumed $Q \to \infty$, the expected value is

$$E[f_n] = E[log_2 A_n] = 2^{-L} \sum_{i=1}^{\infty} (1 - 2^{-L})^{i-1} log_2 i.$$

Using an analogous argument, the variance of $log_2 A_n$ is seen to be

$$var(log_2 A_n) = E[(log_2 A_n)^2] - (E[log_2 A_n])^2 = 2^{-L} \sum_{i=1}^{\infty} (1 - 2^{-L})^{i-1} (log_2 i)^2 - (E[f_n])^2.$$

As we said, if the theoretical mean is exactly the one we just computed, the variance depends on the constant $c(L, K)$. Maurer approximated the value by extensive simulations finding that, for $K \geq 2^L$, the constant is well approximated by

$$c(L, K) \approx 0.7 - \frac{0.8}{L} + \left(4 + \frac{32}{L}\right) \frac{L^{-3/L}}{15}.$$

Knowing the mean and a good approximation of the variance of the statistic f_n, we can finally compute P-value using the complementary error function as in the Algorithmic description section. As we just seen, even if the expected value is theoretically correct, the variance, and in particular the multiplicative constant $c(L, K)$, is obtained by simulations. Coron and Naccache, in [17], assess that the approximation for the variance "can make the test 2.67 times more permissive than what theoretically admitted", so that they re-computed the value of $c(L, K)$ by explicit calculation of the variance: to do that, they computed the probability $P(A_{n+k} = j, A_n = i)$ for all possible cases and found a more precise formula for $c(L, K)$. For $K \geq 33 \cdot 2^L$ they found the better approximation

$$\tilde{c}(L, K)^2 \approx d(L) + \frac{e(L) \cdot 2^L}{K} \tag{3.27}$$

where $d(L)$ and $e(L)$ are functions which take into account different case studies on variance of A_n (i.e. on L).

NIST embedded values of $c(L, K)$ derived by Maurer however, after checking the correctness of our code using these ones, we decided to use new approssimations from [17] in order to make the test more precise. Since values of mean and variance remain the same, we report, in the following table, values of $d(L)$ and $e(L)$ which are necessary to compute new values of the corrective factor $\tilde{c}(L, K)$, using formula (3.27):

L	$d(L)$	$e(L)$
6	0.3489769	0.3941338
7	0.3631815	0.3813210
8	0.3732189	0.3730195
9	0.3800637	0.3677118
10	0.3845867	0.3643695
11	0.3874942	0.3622979
12	0.3893189	0.3610336
13	0.3904405	0.3602731
14	0.3911178	0.3598216
15	0.3915202	0.3595571
16	0.3917561	0.3594040

so that one can compute the standard deviation as

$$\sigma = \tilde{c}(L, K)\sqrt{\frac{variance(L)}{K}}$$

and find the P-value using the complementary error function.

3.2.10 Linear complexity test

This test is focused on the fact that if a random binary sequence has to be unpredictable, it can't be created by a too simple algorithm. In particular the Linear complexity test detects if the sequence could be created by a linear feedback shift register (LFSR) or, alternatively, if the length of this LFSR is long enough to make the sequence random. Indeed random sequences have longer LFSR so that sequences with a short LFSR can be considered non-random. In other words, the test analyzes whether the sequence is complex enough to be considered random.

To better understand the test, we give here the following

Definition 3.2.9 (Feedback Shift Register)**.**
A feedback shift register (FSR) of length L is a stage recursive algorithm. At each stage,

a Boolean function $f : \mathbb{F}_2^L \mapsto \mathbb{F}_2$, called feedback function, generates a bit, taking values from the previous stage.
Starting from an initialization sequence $s_{L-1}, \dots, s_0$ of length L, each step of the algorithm, the following operation are performed:

i) the content of stage 0 is given as output and forms part of the output sequence;

ii) the content of stage i is moved to stage $i-1$ for each $i = 1, \dots, L-1$;

iii) the new content of stage $L-1$, called feedback bit, is created by the recursive formula $s_n = f(s_{n-1}, \dots, s_{n-L}) \mod 2$, for $n \geq L$.

At step $j \geq L$ the sequence $s_0, s_1, \dots, s_{j-1}, s_j$ is created.

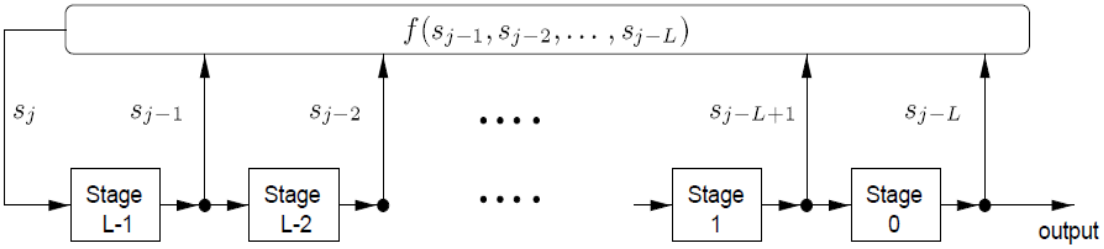

Figure 3.6: Scheme for a feedback shift register. At this step the bit s_{j-L} is outputed and added to the sequence. *Credits to [51].*

A special case of FSR is the

Definition 3.2.10 (Linear Feedback Shift Register)**.**
A linear feedback shift register (LFSR) consists of a feedback shift register where the feedback function f is linear, so that

$$f(s_{n-1}, \dots, s_{n-L}) = \sum_{j=1}^{n} a_j s_{n-j} \mod 2. \tag{3.28}$$

For suitably chosen parameters a_i and the initial state $s_{L-1}, \dots, s_0$, sequences created by a linear feedback shift registers are hardly distinguishable from uniformly distributed random sequence, so that LFSRs could be considered a good pseudo-random generators, if the the length L is sufficiently large: indeed, in this case the generated sequences have a period of 2^{L-1}, while the maximum possible period is $2^L - 1$, see [58] or [51].

Lemma 3.2.2.
A bit sequence $u \in \mathbb{F}_2^{\mathbb{N}}$ is generated by an LFSR if and only if it is (eventually) periodic.

This lead us to look for the shortest LFSR that generates a sequence and the following

Definition 3.2.11 (Linear complexity)**.**
The linear complexity $\lambda(s)$ of a bit sequence $s \in \mathbb{F}_2^n$ is the length of the shortest LFSR that generates a sequence having s as its first n terms, where s can be finite or infinite. By definition we set $\lambda(s) = 0$ if s is formed only by 0s and $\lambda(s) = \infty$ if s is non-periodic.

Some important fact are the followings:

- The sequence $s^L = 0, \ldots, 0, 1$ (formed by $L-1$ zeros, followed by an 1) has period L and linear complexity L.
- For $s \in \mathbb{F}_2^N$ we have $0 \leq \lambda(s) \leq N$.
- $\lambda(s) = N \Leftrightarrow s = (0, \ldots, 0, 1)$.
- If $\lambda(s)$ is the linear complexity of s, to predict each following bit of the sequence one need to know the first $2\lambda(s)$ bits.

Linear complexity can be efficiently determined by the Berlakamp-Massey algorithm (BM-algorithm in the sequel). Since we wont use any property of the field $\mathbb{F}_2$, we will work over an arbitrary field K and look for the shorter linear generator that generates a finite sequence $s \in K^N$. As in (3.28) our LFSR is determined by the recursive formula

$$s_k = a_1 s_{k-1} + \cdots + a_L s_{k-L} \quad \text{for } k = L, \ldots, N-1. \tag{3.29}$$

From the coefficient vector $(a_1, \ldots, a_L) \in K^L$ we can also consider the polynomial in the variable T

$$\varphi(T) = 1 - a_1 T - \ldots a_L T^L \in K[T],$$

called *feedback polynomial.*

We introduce, here, some lemmata, which will be useful for the construction of the BM-algorithm (we refer to [58] for proofs).

Lemma 3.2.3.
Let $s = (s_0, \ldots, s_{n-1}) \in K^n$ be a segment of the output of the linear generator described by (3.29), and let $\hat{s} = (s_0, \ldots, s_n) \in K^{n+1}$ be a sequence not generated by the algorithm. Then every homogeneous linear generator of length $m \geq 1$ that generates $\hat{s}$ satisfies $m \geq n + 1 - L$.

Lemma 3.2.4.
For every sequence $s \in K^N$ we have:

i) $\lambda_{n+1} \geq \lambda_n$ for all n.

ii) There is a homogeneous linear generator of recursion depth λ_n that produces $(s_0, \ldots, s_n)$ if and only if $\lambda_{n+1} = \lambda_n$.

iii) If there is no such generator, then $\lambda_{n+1} \geq n + 1 - \lambda_n$.

These two lemmata lead us to the

Theorem 3.2.5 (Massey)**.**
Let $s \in K^N$, $0 \leq n \leq N-1$ and $\lambda_{n+1}(s) \neq \lambda_n(s)$. Then

$$\lambda_n(s) \leq \frac{n}{2} \text{ and } \lambda_{n+1} = n + 1 - \lambda_n(u).$$

Proof.
The case $\lambda_n = 0$ is trivial, since we have $s_0 = \cdots = s_{n-1} = 0$ and $s_n = 0$, then $\lambda_{n+1} = \lambda_n$ and there's nothing to prove. Otherwise $s_n \neq 0$, then $\lambda_{n+1} = n + 1 = n + 1 - \lambda_n$.
We can observe that the first statement follows from the second one since, if $\lambda_n \neq \lambda_{n+1}$, then $\lambda_n < \lambda_{n+1}$, so that $2\lambda_n < \lambda_n + \lambda_{n+1} < n + 1$.
Let's prove the second statement by induction on n. If $n = 0$ we have $\lambda_0 = 0$ and we fall into the previous case, so let's suppose $n \geq 1$.
Assuming $L := \lambda_n \geq 1$, we have

$$s_j = a_1 s_{j-1} + \cdots + a_L s_{j-L} \quad \text{for } j = L, \ldots, n+1$$

and the corresponding feedback polynomial is

$$\varphi = 1 - a_1 T - \cdots - a_L T^L \in K[T].$$

We define the *n-th discrepancy* as

$$d_n := s_n - a_1 s_{n-1} - \ldots, -a_L s_{n-L}$$

so that if $d_n = 0$, the generator outputs u_n as the next element, and there is nothing to prove. Otherwise, if $d_n \neq 0$, let r be the length of the segment before the last increase of linear complexity, then

$$t := \lambda_r < L, \quad \lambda_{r+1} = L.$$

By induction $L = r + 1 - t$ and we have the relation

$$s_j = b_1 s_{j-1} + \cdots + b_t s_{j-t} \quad \text{for } j = t, \ldots, r-1.$$

The corresponding feedback polynomial is

$$\psi = 1 - b_1 T - \cdots - b_t T^t \in K[T]$$

and the r-th discrepancy is

$$d_r = s_r - b_1 s_{r-1} - \cdots - b_t s_{r-t} \neq 0.$$

If $t = 0$ we have $\psi = 1$ and $d_r = s_r$.
We form now the polynomial

$$\eta := \varphi - \frac{d_n}{d_r} \cdot T^{n-r} \cdot \psi = 1 - c_1 T - \cdots - c_m T^m \in K[T]$$

and let's compute the corresponding output. Directly following for $j = m, \ldots, n-1$ and taking into account that $d_n - (d_n/d_r) \cdot d_r$ for $j = n$, we have:

$$\begin{aligned} s_j - \sum_{i=1}^{m} c_i s_{j-i} &= s_j - \sum_{i=1}^{L} a_i s_{j-i} - \frac{d_n}{d_r} \cdot \left(s_{j-n+r} - \sum_{i=1}^{t} b_i s_{j-n+r-i} \right) \\ &= 0 \quad \text{for} j = m, \ldots, n. \end{aligned}$$

so that the output is $(s_0, \ldots, d_n)$.

By that

$$\lambda_{n+1} \leq m \leq max\{L, n - r + t\} = max\{L, n + 1 - L\},$$

but we also know that linear complexity is increasing, so that $m > L$ and lemma (3.2.3) tells us that $m \geq n + 1 - L$. We obtained that $m = n + 1 - L$ and $\lambda_{n+1} = m$, which concludes the proof. □

Useful consequences of this theorem are that:

- If $d_n \neq 0$ and $\lambda_n \leq \frac{n}{2}$, then $\lambda_{n+1} = n + 1 - \lambda_n > \lambda_n$.
- If $s \in \mathbb{F}_2^N$ is generated by an LFSR of length $\leq L$, then such a LFSR may be determined from $s_0, \ldots, s_{2L-1}$.

The proof of Theorem 3.2.5 gives us an algorithm that builds the LFSR which generates the sequence. We can summarize it as follows:

Algorithm 3.2.6.
Given a sequence $s^N = s_0, \ldots, s_{N-1}$ of length N, initialize $\varphi = 1, \lambda = 0, r = -1, \psi = 1$.
For $n = 0, \ldots, N-1$:
$d = s_n - a_1 s_{n-1} - \cdots - a_L s_{n-L}$
If $d = 1$
$\eta = \varphi - \frac{d}{d'} \cdot T^{n-r} \cdot \psi$
If $l \leq \frac{n}{2}$
$m = n + 1 - L$

$t = l$
$l = m$
$\psi = \varphi$
$r = n$
$d' = d$
$\varphi = \eta$

$\lambda_N(s) = L$ and φ are given as output.

The test uses this algorithm to compute the linear complexity blocks of the observed sequence and compares with the truly random one.

Algorithmic description

1. Partition the sequence into $N = \lfloor \frac{n}{M} \rfloor$ blocks of length M (and discard the possible remainder).

2. Determine the linear complexity L_i of each of the N created blocks through the Berlekamp-Massey algorithm.

3. Compute the theoretical mean as

$$\mu = \frac{M}{2} + \frac{\left(9 + (-1)^{M+1}\right)}{36} - \frac{\left(\frac{M}{3} + \frac{2}{9}\right)}{2^M}.$$

4. For each block compute the value

$$T_i = (-1)^M \cdot (L_i - \mu) + \frac{2}{9}.$$

5. Fixed classes $\nu_0, \ldots, \nu_6$ increment them by one in the following way

if	**Class to increment**
$T_i \leq -2.5$	ν_0
$-2.5 < T_i \leq -1.5$	ν_1
$-1.5 < T_i \leq -0.5$	ν_2
$-0.5 < T_i \leq 0.5$	ν_3
$0.5 < T_i \leq 1.5$	ν_4
$1.5 < T_i \leq 2.5$	ν_5
$T_i > 2.5$	ν_6

6. Compute the χ^2 statistic:

$$\chi^2(obs) = \sum_{i=0}^{K} \frac{(\nu_i - N\pi_i)^2}{N\pi_i}$$

where $\pi_0 = 0.010417, \pi_1 = 0.03125, \pi_2 = 0.125, \pi_3 = 0.5, \pi_4 = 0.25, \pi_5 = 0.0625$, $\pi_6 = 0.020833$.

7. Compute P-value using the incomplete gamma function:

$$P\text{-}value = igamc\left(\frac{K}{2}, \frac{\chi^2(obs)}{2}\right).$$

Interpretation of results

A small P-value (which leads to a rejection) would occur when observed frequency of T_i are too far from the theoretical probabilities. In particular that would indicate that the sequence is locally too simple or, on the other hand, that it has a too high complexity to be effectively random.

An example

Since the rest of the algorithm itself is almost the same of many other algorithms we already analyzed, this example will focus on the Berlekamp-Massey algorithm, which is the core (and the algorithmically most complicate part) of the test, and on what the test does inside one of the N created blocks.

1. Let's suppose $M = 9$ and one of the created blocks is $s_N = 001101110$.

2. The Berlekamp-Massey algorithm proceed as follows (rows of the table represent steps of the algorithm):

s_N	d	η	φ	L	m	ψ	N
-	-	-	1	0	-1	1	0
0	0	-	1	0	-1	1	1
0	0	-	1	0	-1	1	2
1	1	1	$1 - T^3$	3	2	1	3
1	1	$1 - T^3$	$1 - T - T^3$	3	2	1	4
0	1	$1 - T - T^3$	$1 - T - T^2 - T^3$	3	2	1	5
1	1	$1 - T - T^2 - T^3$	$1 - T - T^2$	3	2	1	6
1	0	$1 - T - T^2 - T^3$	$1 - T - T^2$	3	2	1	7
1	1	$1 - T - T^2$	$1 - T - T^2 - T^5$	5	7	$1 - T - T^2$	8
0	1	$1 - T - T^2 - T^5$	$1 - T^3 - T^5$	5	7	$1 - T - T^2$	9

so that the linear complexity of the block is $L_i = 5$.

3. $\mu = \frac{9}{2} + \frac{(9+(-1)^{9+1})}{36} - \frac{(\frac{9}{3}+\frac{2}{9})}{2^9} = 4.7715$.

4. $T_i = (-1)^9 \cdot (5 - 4.7715) + 2/9 = -0.0063$.

5. The T_i we found is such that $-0.5 \leq T_i \leq 0.5$, so that ν_3 would be incremented by one.

After computing values T_i for each block, one can evaluate the χ^2 distribution and find the P-value through the incomplete gamma function, as described in the Algorithmic description section.

Mathematical background

To perform this test we need to know the distribution of linear complexity $\lambda(s)$ and, in particular, its mean, since we use it to form the successive statistic T_n.
Obviously, a N-bit sequence $s = (s_0, \ldots, s_{N-1}) \in \mathbb{F}_2^N$ can be extended to $\tilde{s} = (s_0, \ldots, s_N) \in \mathbb{F}_2^{N+1}$ depending on the bit $S_N \in \{0, 1\}$. The Berlekamp-Massey algorithm discussed above gives us a relation between the linear complexity $\lambda(s)$ and $\lambda(\tilde{s})$, depending on the discrepancy, that we can simply write as

$$\delta = \begin{cases} 0 & \text{if the prediction is correct} \\ 1 & \text{otherwise} \end{cases}$$

where for "prediction" we mean the one the Berlekamp-Massey algorithm do, i.e. if or not the previous founded LFSR can generate the extendend string $\tilde{s}$. Theorem 3.2.5 gave us the following relation

$$\lambda(\tilde{s}) = \begin{cases} \lambda(s) & \text{if } \delta = 0; \\ \lambda(s) & \text{if } \delta = 1 \text{ and } \lambda(s) > N/2; \\ N + 1 - \lambda(s) & \text{if } \delta = 1 \text{ and } \lambda(s) \leq N/2. \end{cases}$$

In order to find the number $\mu_N(L)$ of sequences of length N which have linear complexity L, let's denote

$$M_N(L) := \{s \in \mathbb{F}_2^N \text{ s.t. } \lambda(s) = L\},$$

so that we can think of $\mu_N(L)$ as the cardinality of $M_N(L)$. Obviously, we have the following conditions

i) $0 \leq \mu_n(L) \leq 2^N$,

ii) $\mu_N(L) = 0$ if $L > N$,

iii) $\sum_{L=0}^{N} \mu_N(L) = 2^N$.

From these considerations we can derive the following

Lemma 3.2.7.
The frequency $\mu_N(L)$ of bit sequences of length N and linear complexity L satisfies the following recursion:

$$\mu_{N+1}(L) = \begin{cases} \mu_N(L) & \text{if } 0 \leq L \leq N/2, \\ 2 \cdot \mu_N(L) & \text{if } L = (N+1)/2, \\ 2 \cdot \mu_N(L) + \mu_{N+1-L}(L) & \text{if } L \geq N/2 + 1. \end{cases}$$

Proof.
If $0 \leq L \leq N/2$, s can be continued in two different ways, but only one of that matches the prediction, leading to $\tilde{s} \in M_{N+1}(L)$. The other continuation lead to $\tilde{s} \in M_{N+1}(N+1-L)$, so that we can conclude that, in this case, $\mu_{N+1}(L) = \mu_N(L)$.
If N is odd and $L = \frac{N+1}{2}$ (note that this can't occur if N is even), both correctly predicted and incorrectly predicted sequences leads to $M_{N+1}(L)$ due to the Berlekamp-Massey algorithm. In this case, we have $\mu_{N+1}(L) = 2 \cdot \mu_N(L)$.
Finally, if $L \leq \frac{N}{2} + 1$, both predictions lead to $\tilde{s} \in M_{N+1}(L)$, but we also have to consider the wrong prediction of the first case. Hence $\mu_{N+1}(L) = 2 \cdot \mu_N(L) + \mu_{N+1-L}(L)$, which concludes the proof. □

From that, by induction on N, one could find the following

Theorem 3.2.8 (Rueppel)**.**
The frequency $\mu_N(L)$ of bit sequences of length N and linear complexity L is given by

$$\mu_N(L) = \begin{cases} 1 & \text{if } L = 0, \\ 2^{2-L} & \text{if } 1 \leq L \leq \frac{N}{2}, \\ 2^{2(N-L)} & \text{if } \frac{N+1}{2} \leq L \leq N, \\ 0 & \text{if } L > N. \end{cases}$$

From the distribution of linear complexity we just found, one can determine its mean and variance for sequences of fixed length N. The key is to observe that, under randomness hypothesis, our sequence $s \in \mathbb{F}_2^N$ can be seen as a sequence of N independent binary random variables, so that the mean can be computed as

$$E_N = \sum_{s \in \mathbb{F}_2^N} \lambda(s) P(s) = \frac{1}{2^N} \sum_{s \in \mathbb{F}_2^N} \lambda(s)$$

where the second equality comes, obviously, from the hypothesis of randomness.
If we suddivide the set of all $s \in \mathbb{F}_2^N$ into equivalence classes, according to respective linear complexity, we can revrite the sum

$$\sum_{s \in \mathbb{F}_2^N} \lambda(s) = \sum_{L=1}^{N} \sum_{s \text{ s.t. } \lambda(s)=L} L.$$

Now each equivalent class contains exactly $\mu_N(L)$ elements, so that

$$\sum_{s\in\mathbb{F}_2^N} \lambda(s) = \sum_{L=1}^{N} L \cdot \mu_N(L)$$

and the expected value becomes

$$E_N = \frac{1}{2^N} \sum_{L=1}^{N} L \cdot \mu_N(L).$$

By explicitly distinguishing between cases N odd and N even and using the recursive formulas for μ_N, as in [62], one can prove the

Theorem 3.2.9 (Rueppel)**.**
The expected linear complexity of a sequence $s = (s_0, \ldots, s_{N-1})$ of N independent and uniformly distributed binary random variable is given by

$$E_N(L) = \frac{N}{2} + \frac{4 + R_2(N)}{18} - \frac{1}{2^N}\left(\frac{N}{3} + \frac{2}{9}\right)$$

and its variance is

$$Var_N(L) = \frac{86}{81} - \frac{1}{2^N}\left(\frac{14 - R_2(N)}{27}N + \frac{82 - R_2(N)}{81}\right) - \frac{1}{2^{2N}}\left(\frac{1}{9}N^2 + \frac{4}{27}N + \frac{4}{18}\right).$$

Here, and in the following, $R_2(n)$ is the parity function: it equals 0 if n is even and 1 if it's odd.

For moderately large N mean and variance can be approximated as

$$\mu_N \approx \begin{cases} \frac{N}{2} + \frac{2}{9} & \text{if } N \text{ is even,} \\ \frac{N}{2} + \frac{5}{18} & \text{if } N \text{ is odd;} \end{cases}$$

$$\sigma^2 \approx \frac{86}{81},$$

so that $(L_N - \mu_N)/\sigma^2$ is close to a standard normal variable, and P-values can be found by

$$P\text{-}value = erfc\left(\frac{L_N - \frac{N}{2}}{\sqrt{81/86}}\right).$$

Though Gustafson et al. in [27] assume that as a good approximation and the linear complexity test is embedded in the Crypt-X package [10] in that way, NIST assumes that this approximation is too inaccurate due to the difference on expected value between

the case of even/odd length, then they decided to adapt the statistic T_N to shift the distribution and make it as symmetric as possible defining

$$T_N = (-1)^N (L_N - \xi_N) + \frac{2}{9}$$

where

$$\xi_N = \frac{N}{2} + \frac{4 + R_2(N)}{18}.$$

The limiting distribution of this random variable T, which only assumes integer values, approximates the normal distribution, but it's still skewed to the right:

$$P(T = k) = \begin{cases} \frac{1}{2^{2|k|+1}} & \text{if } k < 0, \\ \frac{1}{2} & \text{if } k = 0, \\ \frac{1}{2^{2k}} & \text{if } k > 0. \end{cases}$$

From these probabilities we can compute the probabilities of $P(T \geq k > 0)$ as $(3 \cdot 2^{2k-2})^{-1}$ and of $P(T \leq k < 0)$ as $(3 \cdot 2^{2|k|-1})^{-1}$, by which we can compute probabilities π_i of classes ν_i of the algorithm, and apply the χ^2-test and find the P-value through the incomplete gamma function, as described in section (3.1.1). We shall note that, choosing block sufficiently large (i.e. choosing M sufficiently large) is really important, in order to make the approximation the more precise as possible: for this purpose NIST recommends $500 \leq M \leq 5000$.

Remark 3.2.10.
The choice between the Standard Normal Distribution and the distribution T is relevant, since the difference on probabilities are relatively large.
Comparing the probabilities of chosen classes intervals $\nu_0, \ldots, \nu_6$, we find that, for the statistic T, probabilities are $\pi_0 = 0.010417, \pi_1 = 0.03125, \pi_2 = 0.125, \pi_3 = 0.5, \pi_4 = 0.25, \pi_5 = 0.0625, \pi_6 = 0.020833$, while the ones from the Standard Normal Distribution would have been $\pi'_0 = 0.0041, \pi'_1 = 0.0432, \pi'_2 = 0.1944, \pi'_3 = 0.3646, \pi'_4 = 0.2863, \pi'_5 = 0.0939, \pi'_6 = 0.0135$.

3.2.11 Serial test

This test analyzes frequencies of all m-bit patterns of the sequence, counted with overlapping. m-bit binary patterns are 2^m in total and, under randomness assumption, any of them has the same probability to appear as the others, since the distribution as to be uniform. Note that this is a test based on frequency and that, if m is equal to 1, it degenerates to the frequency test (3.2.1).

Algorithmic description

1. Starting from $\epsilon = \epsilon_1 \dots \epsilon_n$, form an augmented sequence $\epsilon' = \epsilon_1 \dots \epsilon_n \epsilon_1 \dots, \epsilon_{m-1}$ by appending the first $m-1$ bits to the end of the sequence.

2. Count the frequency of

   - each possible overlapping m-bit block (pattern);
   - each possible overlapping $(m-1)$-bit block;
   - each possible overlapping $(m-2)$-bit block.

   This will provide:

   - $v_{i_1,\dots,i_m}$ with $i_j \in \{0,1\}$, the number of occurrences of the pattern $i_1, \dots, i_m$ for all $j = 1, \dots, m$;
   - $v_{i_1,\dots,i_{m-1}}$ with $i_j \in \{0,1\}$, the number of occurrences of the pattern $i_1, \dots, i_{m-1}$ for all $j = 1, \dots, m-1$;
   - $v_{i_1,\dots,i_{m-2}}$ with $i_j \in \{0,1\}$, the number of occurrences of the pattern $i_1, \dots, i_{m-2}$ for all $j = 1, \dots, m-2$.

3. Compute the statistics

$$\psi_m^2 = \frac{2^m}{n} \sum_{i_1,\dots,i_m} \left(v_{i_1,\dots,i_m} - \frac{n}{2^m} \right)^2 = \frac{2^m}{n} \sum_{i_1,\dots,i_m} v_{i_1,\dots,i_m}^2 - n;$$

$$\psi_{m-1}^2 = \frac{2^{m-1}}{n} \sum_{i_1,\dots,i_{m-1}} \left(v_{i_1,\dots,i_{m-1}} - \frac{n}{2^{m-1}} \right)^2 = \frac{2^{m-1}}{n} \sum_{i_1,\dots,i_{m-1}} v_{i_1,\dots,i_{m-1}}^2 - n;$$

$$\psi_{m-2}^2 = \frac{2^{m-2}}{n} \sum_{i_1,\dots,i_{m-2}} \left(v_{i_1,\dots,i_{m-2}} - \frac{n}{2^{m-2}} \right)^2 = \frac{2^{m-2}}{n} \sum_{i_1,\dots,i_{m-2}} v_{i_1,\dots,i_{m-2}}^2 - n,$$

   setting, by definition, $\psi_0^2 = \psi_{-1}^2 = 0$.

4. Compute distances (statistics themselves):

$$\nabla \psi_m^2 = \psi_m^2 - \psi_{m-1}^2$$
$$\nabla^2 \psi_m^2 = \psi_m^2 - 2\psi_{m-1}^2 + \psi_{m-2}^2$$

5. Compute P-values using the incomplete gamma function:

$$P\text{-}value_1 = igamc\left(2^{m-2}, \frac{\nabla \psi^2}{2}\right)$$
$$P\text{-}value_2 = igamc\left(2^{m-3}, \frac{\nabla^2 \psi^2}{2}\right)$$

Interpretation of results

A small P-value occurs when values of the two statistic $\nabla\psi_m^2$ and $\nabla^2\psi_m^2$ are large: that implies that there is large non-uniformity between pattern: some m-bit patterns occur too many (or too few) times, compared to other m-bit patterns and to what is expected from a random sequence.

An example

Let $\epsilon = 0011011101$ be the testing sequence (so that $n = 10$) and let $m = 3$.

1. $m = 3$, then we append the first $m-1 = 2$ bits of the sequence to its end, obtaining $\epsilon' = 001101110100$.
2. Possible 3-bit overlapping templates ($2^m = 8$) are $000, 001, 010, 011, 100, 101, 110$ and 111 so that respective frequencies are $v_{000} = 0, v_{001} = 1, v_{010} = 1, v_{011} = 2, v_{100} = 1, v_{101} = 2, v_{110} = 2, v_{111} = 0$; For the 4 possible 2-bit patterns we have $v_{00} = 1, v_{01} = 3, v_{10} = 3, v_{11} = 3$ and $(m-2)$-bit frequencies are $v_0 = 4$ and $v_1 = 6$.
3. $\psi_3^2 = \frac{2^3}{10}(0+1+1+4+1+4+4+1) - 10 = 12.8 - 10 = 2.8$
   $\psi_2^2 = \frac{2^2}{10}(1+9+9+9) - 10 = 11.2 - 10 = 1.2$
   $\psi_1^2 = \frac{2}{10}(16+36) - 10 = 10.4 - 10 = 0.4$.
4. $\nabla\psi_3^2 = 2.8 - 1.2 = 1.6 \qquad \nabla^2\psi_m^2 = 2.8 - 2 \cdot 1.2 + 0.4 = 0.8$
5. $P\text{-}value_1 = igamc\left(2, \frac{1.6}{2}\right) = 0.9057$ and $P\text{-}value_2 = igamc\left(1, \frac{0.8}{2}\right) = 0.8805$. Both P-values are greater than $\alpha = 0.01$, then the sequence is accepted as random.

Mathematical background

The serial test uses two different types of statistic, the one of ψ_m^2, ψ_{m-1}^2 and ψ_{m-2}^2 and the statistics of distances $\nabla\psi_m^2$ and $\nabla^2\psi_m^2$. We will show, here, the reason why $\nabla\psi_m^2$ and $\nabla^2\psi_m^2$ are used instead of ψ_ν^2 with $\nu \in \{m, m-1, m-2\}$ and we'll prove that used statistics have a χ^2 distribution, then the incomplete gamma function can be used to find the P-value.

First we used statistics of the form

$$\psi_m^2 = \frac{2^m}{n} \sum_i \left(v_i - \frac{n}{2^m}\right)$$

where, for briefness, we used i to indicate $i_1, \ldots, i_m$. Even though ψ_m^2 has the form of a χ^2 statistic with $2^m - 1$ degrees of freedom, in fact it hasn't a χ^2 distribution (not even asymptotically) since, as Good proved in [26], one can see that its expected value is

$$E[\psi_m^2] = 2^m - 1$$

while the real number of degrees of freedom of ψ_m^2 is strictly minor than $2^m - 1$, and the expected value of a χ^2 distribution equals the number of degrees of freedom. This depends on the fact that, under randomness hypothesis, we have some restraints on frequencies. For example, for 2-bit sequences, we have the following two linear restraints on ψ_2^2:

$$v_{i_1,0} + v_{i_1,1} = v_{0,i_1} + v_{1,1i} \quad \text{for } i_1 = 0, 1$$

i.e. the number of sequence of the form $i, 0$ or $i, 1$ has to be the same of the number of $0, i$ or $1, i$ and, similarly, for sequences of each fixed length m. This implies that, in fact, the number of degrees of freedom is not $2^2 - 1$ as expected, but only 2, for the example. In general the expected number of degrees of freedom was $2^m - 1$ but, in fact, they are 2^{m-1}, so that the statistic couldn't have a χ^2 distribution (see [5] for detailed proof).

To find a statistic that effectively has a χ^2 distribution, the two following statistics are formed:

$$\nabla\psi_m^2 = \psi_m^2 - \psi_{m-1}^2$$

and

$$\nabla^2\psi_m^2 = \psi_m^2 - 2\psi_{m-1}^2 + \psi_{m-2}^2.$$

Good in [26] decomposes ψ_m^2 into a sum of normal distributed functions so that $\nabla\psi_m^2$ and $\nabla^2\psi_m^2$ satisfy the definition of the χ^2 distribution. The decomposition is made by mean of

Definition 3.2.12 (m-dimensional discrete Fourier transform)**.**
The m-dimensional mod 2 *discrete Fourier transform of a function $v = v_i$ is defined by*

$$v_s^* = \sum_i v_i \omega^{<i,s>},$$

where $\omega = exp(2\pi i/2)$, $i = i_1, \ldots, i_m$ as above and $< i, s >= i_1 s_1 + \cdots + i_m s_m$ is the scalar product of vectors i and s.

Observing that the discrete Fourier transformation is linear, one can observe that the transform of $\left(v_i - \frac{n}{2^m}\right)$ is v_s^* minus the transform of the constant $\frac{n}{2^m}$. The transform of the latter is zero, unless $s = (0, \ldots, 0)$ when it equals to n; we can also observe that $v_0^* = \sum_i v_i = n$. By previous remarks, using the Rayleigh-Parseval formula ($\sum_i v_i^2 = 2^{-m} \sum_s v_s^* v_{-s}^*$) in its discrete version, we find that

$$\sum_i \left(v_i - \frac{n}{2^m}\right) = \frac{1}{2^m} \sum_{s \neq 0} |v_s^*|^2.$$

If we classify the 2^m vectors s into $m + 1$ classes depending on how many non-zero values they have, we can use the previous formula to rewrite ψ_m^2 as

$$n\psi_m^2 = \sum_{d=m} |v_s^*|^2 + 2 \sum_{d=m-1} |v_s^*|^2 + \cdots + m \sum_{d=1} |v_s^*|^2 \tag{3.30}$$

where $d = j$ indicates the number of non-zero values of the vector. Bartlett in [5] has proved that the joint distribution of v_i is a asymptotically multivariate normal distribution and, covariances, Good proved that sums in (3.30) have approximately independent χ^2 distribution with 2^{d-2} degrees of freedom, respectively. Following from that $\nabla\psi_m^2$ and $\nabla^2\psi_m^2$ have χ^2 distributions with, respectively, 2^{m-1} and 2^{m-2} degrees of freedom.

3.2.12 Approximate entropy test

As Maurer's universal statistical test, the approximate entropy test analyzes the entropy of the sequence $\epsilon = \epsilon_1 \dots \epsilon_n$ by the frequencies of m-bit patterns with the ones of $(m+1)$-bit patterns, in analogy with the serial test. Moreover, as in the overlapping template matching test, patterns can overlap.

Algorithmic description

1. Starting from $\epsilon = \epsilon_1 \dots \epsilon_n$, form an augmented sequence $\epsilon' = \epsilon_1 \dots \epsilon_n \epsilon_1 \dots, \epsilon_{m-1}$ by appending the first $m-1$ bits to the end of the sequence: this will create n overlapping sub-sequences of length m to analyze.

2. For each m-bit pattern i count the number of occurrences through the entire sequence. The count is made in an "overlapping way", i.e. if at j-th step bits from ϵ_j to ϵ_{j+m-1} are examined, the $(j+1)$-th step will analyze bits from ϵ_{j+1} to ϵ_{j+m}. Let $\#i$ be te number of occurrences of the pattern i through the sequence.

3. Compute $C_i^m = \frac{\#i}{n}$ the frequency of pattern i inside the sequence, for each possible i.

4. Compute
$$\varphi_m = \sum_i \pi_i log \pi_i, \quad \text{where } \pi_i = C^m_{log2i};$$
here i indicates the corresponding decimal value of the string i.

5. Repeat steps from 1 to 4 replacing m by $m+1$, obtaining φ_{m+1}.

6. Compute the approximate entropy
$$ApEn(m) = \varphi_m - \varphi_{m+1}.$$

7. Compute the statistic
$$\chi^2(obs) = 2n[log2 - ApEn(m)].$$

8. Compute P-value through the incomplete gamma function:
$$P\text{-}value = igamc\left(2^{m-1}, \frac{\chi^2(obs)}{2}\right).$$

Interpretation of results

A small P-value will occur when the value of the statistic $ApEn(m)$ is too small or too large: the first case would indicate that the sequence shows strong regularity between pattern (i.e. there is no change of entropy between m-bit strings and $m+1$ strings) and the second one would imply fluctuation or irregularity on patterns of the string, and in particular between patterns of subsequent length.

An example

Let's take the sequence $\epsilon = 0100110101$, so that $n = 10$, and $m = 3$.

1. $\epsilon' = 010011010101$, appending the first $m - 1 = 2$ bit to the end of the sequence.

2. We count the number of occurrences of the $2^m = 2^3 = 8$ possible templates, finding that $\#000 = 0, \#001 = 1, \#010 = 3, \#011 = 1, \#100 = 1, \#101 = 3, \#110 = 1, \#111 = 0$.

3. $C_i^3 = \frac{\#i}{10}$ so that $C_{000}^3 = 0, C_{001}^3 = 0.1, C_{010}^3 = 0.3, C_{011}^3 = 0.1, C_{100}^3 = 0.1, C_{101}^3 = 0.3, C_{110}^3 = 0.1, C_{111}^3 = 0$.

4. $\varphi_3 = 0(log0) + 0.1(log0.1) + 0.3(log0.3) + 0.1(log0.1) + 0.1(log0.1) + 0.3(log0.3) + 0.1(log0.1) + 0(log0) = -1.64341772$.

5. Let's repeat steps from 1 to 4 by replacing m with $m + 1$:

   i) $m = 4$ so that we append the first $m-1 = 3$ bits, obtaining $\epsilon' = 0100110101010$.

   ii) Computed values are $\#0011 = 1, \#0100 = 1, \#0101 = 2, \#0110 = 1, \#1001 = 1, \#1010 = 3, \#1101 = 1$ and all the other values are 0.

   iii) $C_{0011}^4 = C4_{0100} = C_{0110}^4 = C_{1001}^4 = C_{1101}^4 = 0.1, C_{0101}^4 = 0.2, C_{1010}^4 = 0.3$, and all other values are zero.

   iv) $\varphi_4 = 0+0+0+0.1(log0.01)+0.1(log0.01)+0.2(log0.02)+0.1(log0.01)+0+0+0.1(log0.01)+0.3(log0.03)+0+0+0.1(log0.01)+0+0) = -1.83437197$.

6. $ApEn(3) = \varphi_3 - \varphi_4 = -1.643418 - (-1.834372) = 0.190954$.

7. $\chi^2 = 2 \cdot 10 \cdot (0.693147 - 0.190954) = 0.502193$.

8. $P\text{-}value = igamc\left(2^2, \frac{0.502193}{2}\right) = 0.261961 > \alpha$, and the sequence is accepted as random.

Mathematical background

The theory behind this test is mostly based on the concept of approximate entropy *ApEn*, introduced by Pincus in [55]. The purpose of approximate entropy is to characterize the idea of randomness and study what happens when the sequence grows, one bit after another. To better understand why the approximate entropy can be used to detect regularity, or irregularity, we shall give the following

Definition 3.2.13 (Distance between two blocks)**.**
Let $n, m \in \mathbb{N} \setminus \{0\}$ with $m \leq n$, $r \in \mathbb{R}$ and $s = s_1, \ldots, s_n$ be a sequence of real numbers. Given two blocks of the same length $x(i) = s_i, s_{i+1}, \ldots, s_{i+m-1}$ and $x(j)$ of s, the distance between blocks $x(i)$ and $x(j)$ is defined as

$$d(x(i), x(j)) = max_{k=1,\ldots,m}(|s(i+k-1) - s(j+k-1)|).$$

The previous notion allows to define a measure of how blocks of consecutive length (m and $m+1$) differ or remain the same through the entire sequence and to measure the frequency of how often this occurs, by the use of

$$C_i^m(r) = \frac{\#\{j \leq N - m + 1 \text{ s.t. } d(x(i), x(j)) \leq r\}}{N - m + 1}$$

We can now give the

Definition 3.2.14 (Approximate entropy)**.**

$$\varphi_m(r) = \frac{1}{n - m + 1} \sum_{i=1}^{N-m+1} logC_i^m(r)$$

and the approximate entropy is defined as

$$ApEn(m, r, N)(s) = \varphi_m(r) - \varphi_{m+1}(r)$$

with $ApEn(0) = -\varphi_1$.

For completeness we gave definitions for strings with values in $\mathbb{R}$ but, in our case, the sequence takes values in $\{0, 1\}$, so that the distance will only assume values 0 or 1, and could be defined as

$$d(x(i).x(j)) = \begin{cases} 0 & \text{if } x(i) = x(j), \\ 1 & \text{if } x(i) \neq x(j), \end{cases}$$

and r could only be equal to 1, so that we won't be interested in it anymore. As written by Pincus and Huang in [54] "analytic proofs of asymptotic normality and expecially explicit variance estimates for *ApEn* appear to be extremely difficult", so the key is to define a new type of approximate entropy, which will have the same asymptotic behavior of the approximate entropy but it is simpler to estimate. Indeed Pincus characterizes randomness for finite sequences as

Definition 3.2.15 ((m,n)-random sequence)**.**
A binary sequence ϵ of length n is said to be (m,n)-random if

$$ApEn(m,n)(\epsilon) = max_s ApEn(m,n)(s \in S),$$

where S is the set of all possible 2^n binary sequences of length n.

Let $\tilde{\epsilon} = \epsilon_1, \ldots, \epsilon_n \epsilon_1 \ldots, \epsilon_{m-1}$ be the augmented (or in a certain sense circular) version of the original string, as we do at the first step of the algorithm, and let the "modified" version of the empirical distribution be

$$\tilde{\varphi_m} = \sum_{i_1,\ldots,i_m} \nu_{i_1,\ldots,i_m} log \nu_{i_1,\ldots,i_m},$$

where $\nu_{i_1,\ldots,i_m}$ is the relative frequency of the pattern $i_1, \ldots, i_m$ in $\tilde{\epsilon}$. If $\omega_{i_1,\ldots,i_m} = n\nu_{i_1,\ldots,i_m}$ is the frequency of the pattern, by this definition we have that $\omega_{i_1,\ldots,i_m} = \sum_k \omega_{i_1,\ldots,i_m,k}$ so that $\sum_{i_1,\ldots,i_m} \nu_{i_1,\ldots,i_m} = n$.
We define the modified approximate entropy as

$$\widetilde{ApEn}(m) = \tilde{\varphi_m} - \tilde{\varphi_{m+1}}.$$

If we use Jensen's inequality ($\psi(\sum a_i x_i) \leq \sum a_i \psi(x_i)$) with the function ψ which extends the string, we obtain a first upper bound for the modified approximate entropy: $\widetilde{ApEn}(m) \leq logs$, where s is the cardinality of the set of value, so that in our case $s = 2$, so that the largest possible value of $\widetilde{ApEn(m)}$ is $log2$.
On the other hand, we also have that $ApEn$ and $\widetilde{ApEn}$ can't differ much, for large values of n: if $Y_i(m) = i_1, \ldots, i_m$ and $\nu'_{i_1,\ldots,i_m} = C_i^m$, we have

$$\varphi_m = \sum_{i_1,\ldots,i_m} \nu'_{i_1,\ldots,i_m} log \nu'_{i_1,\ldots,i_m},$$

then, if $\omega_{i_1,\ldots,i_m} = (n-m+1)\nu'_{i_1,\ldots,i_m}$ we have that

$$\sum_{i_1,\ldots,i_m} = n - m + 1$$

and $\omega_{i_1,\ldots,i_m} - \omega'_{i_1,\ldots,i_m} \leq m - 1$.

We finally obtain that

$$|\nu_{i_1,\ldots,i_m} - \nu'_{i_1,\ldots,i_m}| \leq \frac{m-1}{n-m+1}, \tag{3.31}$$

which implies that, if m is fixed, $ApEn(m)$ and $\widetilde{ApEn}(m)$ are close together for large values of n.

The last thing we need to know to perform the test is the distribution of the statistic $n[log2 - ApEn(m)]$ since, if we get that, we have the theoretical distribution to use to compute the P-value. Rukhin, in [63], proved that, whether we use the augmented sequence and the modified approximate entropy or the ones defined by Pincus, we always have that the statistic has a χ^2 distribution with $(s-1)s^m$ degrees of freedom. Indeed the following theorem holds:

Theorem 3.2.10 (Rukhin)**.**
For fixed m, as $n \to \infty$, one has the following convergence in distribution:

$$2n[logs - \widetilde{ApEn}(m)] \to \chi^2(s^{m+1} - s^m). \tag{3.32}$$

Also

$$n[ApEn(m) - \widetilde{ApEn}(m)] = O\left(\frac{1}{n}\right), \tag{3.33}$$

so that

$$2n[logs - ApEn(m)] \to \chi^2(s^{m+1} - s^m).$$

Proof.
Let $Z_{i_1,\ldots,i_m}$ be the difference between empirical and theoretical probabilities:

$$Z_{i_1,\ldots,i_m} = \sqrt{n}\left(\nu_{i_1,\ldots,i_m} - \frac{1}{s^m}\right);$$

The vector formed by $Z_{i_1,\ldots,i_m}$ has, by the central limit theorem, asymptotic multivariate distribution which mean is zero and variance of the form

$$\Sigma_m = \frac{1}{S^m} I_m - \frac{1}{2^{2m}} e_m e_m^T,$$

where I_m stands for the $s^m \times s^m$ identity matrix and e_m^T is the vector $(1,\ldots,1)$ which length is s^m.
Chaitin, in [13], showed that

$$\begin{aligned}\tilde{\varphi}_m &= -\sum_{i_1,\ldots,i_m}\left[\frac{1}{s^m} + \frac{Z_{i_1,\ldots,i_m}}{\sqrt{n}}\right]\cdot\left[-mlogs + \frac{s^m Z_{i_1,\ldots,i_m}}{2n} + O\left(\frac{1}{n^{3/2}}\right)\right] \\ &\sim mlogs + \frac{s^m}{2n}\sum_{i_1,\ldots,i_m} Z^2_{i_1,\ldots,i_m}.\end{aligned}$$

Using analogous notations for pattern of length $m+1$ one can easily see that

$$Z_{i_1,\dots,i_m,k} = \sum_{k=1}^{s} Z_{i_1,\dots,i_m,k}$$

and

$$\tilde{\varphi}_{m+1} \sum (m+1) logs + \frac{s^{m+1}}{2n} \sum_{i_1,\dots,i_m,i_{m+1}} Z^2_{i_1,\dots,i_m,i_{m+1}}.$$

Using the approximations for $\tilde{\varphi}_m$ and $\tilde{\varphi}_{m+1}$ we have

$$\begin{aligned}\tilde{\varphi}_m - \tilde{\varphi}_{m+1} &\sim logs - \frac{s^m}{2n}\left[\sum_{i_1,\dots,i_m}\left(\sum_k Z_{i_1,\dots,i_m,l}\right)^2 - s \sum_{i_1,\dots,i_m,i_{m+1}} Z^2_{i_1,\dots,i_m,i_{m+1}}\right] \\ &= logs - \frac{s^m}{2n} Z^T Q Z\end{aligned}$$

where Q is an $s^{m+1} \times s^{m+1}$ block-diagonal matrix, formed by s^m blocks Q_0 which have the form

$$Q_0 = sI_1 - e_1 e_1^T,$$

and Z is a normal vector of length s^{m+1}.
By spectral decomposition, the quadratic form $Z^T Q Z$ has the same distribution of $\sum l_{i_1,\dots,i_m,i_{m+1}} W^2_{i_1,\dots,i_m,i_{m+1}}$ where $W_{i_1,\dots,i_m,i_{m+1}}$ are independent standard normal variables and $l_{i_1,\dots,i_m,i_{m+1}}$ are the eigenvalues of the matrix $\Sigma^{1/2} Q \Sigma^{1/2}$.
By direct computing one can see that

$$\Sigma^{1/2}_{m+1} = \frac{1}{s^{(m+1)/2}} I_{m+1} - \frac{1}{s^{3(m+1)/2}} e_{m+1} e^T_{m+1},$$

and $\Sigma^{1/2}_{m+1} Q \Sigma^{1/2}_{m+1} = \frac{1}{s^{m+1}} Q$.
Directly computing the eigenvalues of $\Sigma^{1/2}_{m+1} Q \Sigma^{1/2}_{m+1}$ one can find that needed eigenvalues are equal to s with multiplicity $(s-1)s^m$ and 0 with multiplicity s^m, so that

$$\tilde{\varphi}_m - \tilde{\varphi}_{m+1} \sim logs - \frac{1}{2n}\chi^2((s-1)s^m)$$

and

$$n[logs - \widetilde{ApEn}(m)] \sim \frac{1}{2}\chi^2(s^{m+1} - s^m).$$

Now (3.31) shows that if $Z'_{i_1,\dots,i_m} = \sqrt{n}[\nu'_{i_1,\dots,i_m} - s^{-m}]$, then $|Z'_{i_1,\dots,i_m} - Z_{i_1,\dots,i_m}| \le (m-1)\sqrt{n}/(n-m+1)$ and

$$|\tilde{\varphi}_m - \varphi_m| \sim \frac{s^m}{2n}\left|\sum_{i_1,\dots,i_m} Z^2_{i_1,\dots,i_m} - \sum_{i_1,\dots,i_m} Z'^2_{i_1,\dots,i_m}\right| \le \frac{s^{2m}(m-1)^2}{2(n-m+1)^2}.$$

(3.33) comes directly from that last inequality so that the last convergence in distribution of the statement follows, using (3.32) and (3.33) together. □

This theorem, applied to our case where $s = 2$, tells us that the statistic χ^2, computed at step 7 of the algorithm, effectively has a χ^2 distribution with 2^m degrees of freedom, so that the P-value can be computed using the incomplete gamma function *igamc* as

$$P\text{-}value = igamc\left(\frac{2^m}{2}, \frac{\chi^2}{2}\right).$$

3.2.13 Cumulative sums (Cusum) test

In this test, the sequence is considered as a random walk by transforming the binary string into a string of -1 and $+1$. As a random walk, we analyze excursions from zero of the walk by computing partial sums and we check if these are too far from zero. Indeed the random walk created by a truly random sequence tends to stay near zero, since the number of zeros and ones has to be approximately the same and the alternation between them has to be relatively fast. Note that a sequence won't pass the test though its excursion in too low.

Algorithmic description

1. The sequence is transformed into a string of -1 and $+1$.

2. Partial sums are computed: this could be done both forward and backward, providing two different tests (and P-values):

Forward	**Backward**
$S_1 = X_1$	$S_1 = X_n$
$S_2 = X_1 + X_2$	$S_2 = X_n + X_{n-1}$
$S_3 = X_1 + X_2 + X_3$	$S_3 = X_n + X_{n-1} + X_{n-2}$
$\vdots$	$\vdots$
$S_k = X_1 + X_2 + X_3 + \ldots + X_k$	$S_3 = X_n + X_{n-1} + X_{n-2} + \ldots + X_{n-k+1}$
$\vdots$	$\vdots$
$S_n = X_1 + \ldots + X_k + \ldots + X_n$	$S_n = X_n + \ldots + X_k + \ldots + X_1$

   Note that, obviously, $S_k = S_{k-1} + X_k$ for the forward sum and $S_k = S_{k-1} + X_{n-k+1}$ for the backward one.

3. Compute the maximal excursion from zero:

$$z = \max_{k \in \{1,\ldots,n\}} |S_k|.$$

4. Compute P-value as

$$P\text{-}value = 1 - \sum_{k=\left(\frac{-n}{z}+1\right)/4}^{\left(\frac{n}{z}-1\right)/4} \left[\Phi\left(\frac{(4k+1)z}{\sqrt{n}}\right) - \Phi\left(\frac{(4k-1)z}{\sqrt{n}}\right)\right] + \sum_{k=\left(\frac{-n}{z}-3\right)/4}^{\left(\frac{n}{z}-1\right)/4} \left[\Phi\left(\frac{(4k+3)z}{\sqrt{n}}\right) - \Phi\left(\frac{(4k+1)z}{\sqrt{n}}\right)\right]$$

where Φ is the cumulative probability distribution function of the Standard Normal distribution, as defined in (2.1).

Interpretation of results

A small P-value (i.e. a rejection) can occur if 0 and 1 are intermixed too frequently, and this will correspond to small values of the largest excursion z. A large z in the forward testing would indicate that there are too many ones or too many zeros in the first part of the sequence and, on the other hand, that there are too many ones or zeros in the final part, if the computation is made backwards.

An Example

Let $\epsilon = 1011010111$ be the test sequence. The test proceeds as follows:

1. $X = 1, -1, 1, 1, -1, 1, -1, 1, 1, 1$
2. If we compute partial sums in forward mode the computation is:

$$\begin{aligned}
S_1 &= 1 \\
S_2 &= 1 + (-1) = 0 \\
S_3 &= 1 + (-1) + 1 = 1 \\
S_4 &= 1 + (-1) + 1 + 1 = 2 \\
S_5 &= 1 + (-1) + 1 + 1 + (-1) = 1 \\
S_6 &= 1 + (-1) + 1 + 1 + (-1) + 1 = 2 \\
S_7 &= 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) = 1 \\
S_8 &= 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) + 1 = 2 \\
S_9 &= 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) + 1 + 1 = 3 \\
S_{10} &= 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) + 1 + 1 + 1 = 4
\end{aligned}$$

3. $z = 4$ since 4 it's the largest value reached by partial sums.
4. $P\text{-}value = 0.4116588 < \alpha$ and the sequence is accepted as random.

Mathematical background

P-value for this test is computed in a way different to other tests, since we can't reduce the distribution of partial sums of a random walk to a Standard Normal Distribution or to a χ^2 distribution: we have to find the distribution of partial sums $S_k = X_1 + \ldots + X_k$. Through combinatorial argument one can prove the following

Proposition 3.2.11.
Let S_n with $n \in \mathbb{N}$ be the partial sum of the output of a Bernoulli process. The following equalities hold:

$$P(S_{2n} = 2k) = \binom{n}{k} 2^{-2n} \quad \text{for } k = -n, -n+1, \ldots, n;$$

$$P(S_{2n+1} = 2k+1) = \binom{2n+1}{n-k} 2^{-2n-1} \quad \text{for } k = -n-1, -n, \ldots, n.$$

These two equalities give

$$P(S_n = k) = \begin{cases} \binom{n}{(n-k)/2} 2^{-n} & \text{if } k \equiv n \mod 2 \\ 0 & \text{otherwise} \end{cases} \tag{3.34}$$

for $k = -n, -n+1, \ldots, n$.
From those

$$E[S_n] = 0, \quad E[S_n^2] = n, \quad E[e^{tS_n}] = \left(\frac{e^t + e^{-t}}{2} \right)^2. \tag{3.35}$$

Now we introduce a notation for the equivalent theoretical maximum and to consider the maximum positive (left) and negative (right) excursion of the random walk:

$$M = \max_{k \in \{1,\ldots,n\}} |S_k|, \quad M^+ = \max_{k \in \{1,\ldots,n\}} S_k, \quad M^- = -\min_{k \in \{1,\ldots,n\}} S_k.$$

Using these notations we can enunciate and prove the following theorem, first proposed by Renyi in [59]:

Theorem 3.2.12 (Renyi)**.**

$$\begin{aligned} p_{n,k} &= P(M^+ = k) \\ &= \binom{n}{[(n-k)/2]} 2^{-n} \end{aligned} \tag{3.36}$$

for $k = 0, 1, \ldots, n$.

Proof.
We prove the equality by induction. Let

$$\bar{M}^+ = \max_{1\le k\le n+1} \sum_{j=2}^{k} X_j$$

then, for $k \ge 1$,

$$\begin{aligned} p_{n+1,k} &= P(X_1 = 1, \bar{M}^+ = k-1) + P(x_1 = -1, \bar{M}^+ = k+1) \\ &= \frac{1}{2}(p_{n,k-1} + p_{n,k+1}) \end{aligned}$$

For $k = 0$, in a similar way

$$p_{n+1,0} = P(X_1 = -1, \bar{M}^+ \le 1) = \frac{1}{2}(p_{n,1} + p_{n,0}).$$

We know that $p_{1,0} = p_{1,1} = \frac{1}{2}$ so that we obtain (3.36) by induction. □

The theorem which follows is due to Billingsley: we omit the proof and refer to [6].

Theorem 3.2.13 (Billingsley)**.**
For any integers $a \le 0 \le b$, $a \le \nu \le b$ we have

$$\begin{aligned} p_n(a,b,\nu) &= P(a < -M^- \le M^+ < b, S_n = \nu) \\ &= \sum_{k=-\infty}^{\infty} q_n(\nu + 2k(b-a)) - \sum_{k=-\infty}^{\infty} q_n(2b - \nu + 2k(b-a)) \end{aligned} \tag{3.37}$$

where, for $j = -n, -n+1, \ldots, n$,

$$q_n(j) = P(S_n = j) = \begin{cases} \binom{n}{(n-j)/2} 2^{-n} & \text{if } j \equiv n \mod 2, \\ 0 & \text{otherwise.} \end{cases}$$

A direct consequence of this theorem is the following

Theorem 3.2.14.
For any integers $a \le 0 \le b$ and $a \le u \le \nu \le b$ we have

$$\begin{aligned} &P(a < -z^- \le M^+ < b, u < S_n < \nu) \\ &\quad = \sum_{k=-\infty}^{\infty} P(u + 2k(b-a) < S_n < \nu + 2k(b-a)) \\ &\qquad - \sum_{k=-\infty}^{\infty} P(2b - \nu + 2k(b-a) < S_n < 2b - u + 2k(b-a)), \end{aligned} \tag{3.38}$$

$$
\begin{aligned}
&P(a < -M^- \leq M^+ < b) \\
&\quad = \sum_{k=-\infty}^{\infty} P(a + 2k(b-a) < S_n < b + 2k(b-a)) \\
&\qquad - \sum_{k=-\infty}^{\infty} P(b - \nu + 2k(b-a) < S_n < b - a + 2k(b-a)) \qquad (3.39)
\end{aligned}
$$

and

$$
\begin{aligned}
P(M < b) = &\sum_{k=-\infty}^{\infty} P((4k-1)b < S_n < (4k+1)b) \\
&- \sum_{k=-\infty}^{\infty} P((4k+1)b < S_n < (4k+3)b). \qquad (3.40)
\end{aligned}
$$

Proof.
(3.38) follows directly from (3.37); (3.39) is the special case of (3.38) with $u = a$, $\nu = b$ and (3.14) follows from (3.39) taking $a = -b$. □

Formula (3.40) is what we need, since it is exactly the complementary of the P-value so, by complementing

$$
\begin{aligned}
\textit{P-value} &= P(\max_{1 \leq k \leq 1} |S_k| \geq z) \\
&= 1 - \sum_{k=-\infty}^{\infty} P((4k-1)z < S_n < (4k+1)z) \\
&\qquad + \sum_{k=-\infty}^{\infty} P((4k+1)z < S_n < (4k+3)z).
\end{aligned}
$$

As in (3.35) the distribution of S_n has mean $\mu = 0$ and variance $\sigma^2 = n$, so that we can use the Standard Normal cumulative distribution function Φ with $z = z/\sqrt{n}$ to compute probabilities of the summations. Obviously, in practice summations are truncated, however simulations showed that bound proposed in the algorithm provide a good approximation.

Remark 3.2.11.
Note that indices k of summations at 4th step of the algorithm would be non-integer, however the theory at the base of the test shows that they must be integers: that's why indices have to be approximated to obtain truthful results.

3.2.14 Random excursion test

As the previous one, this test considers the binary string as a random walk, inspecting cycles of the walk to check if their behavior is as expected from a true random walk. Cycles of a random walk are defined as

Definition 3.2.16 (Cycle of a random walk)**.**
A cycle of a random walk is a sequence of steps (each of length 1*) which start from and ends at the origin.*

The focus is the number of visits of the totality of the cycles to a particular state k and the test checks if this is as expected under hypothesis of randomness. Visits which are examined are the ones to states $-4, -3, -2, -1, 1, 2, 3, 4$ producing 8 P-values (one for each state checked).

Algorithmic description

1. The sequence is transformed into a string of -1 and $+1$.
2. Partial sums S_i are computed to obtain the set $S = \{S_i\}_{i=1,\ldots,n}$.

$$\begin{aligned} S_1 &= X_1 \\ S_2 &= X_1 + X_2 \\ S_3 &= X_1 + X_2 + X_3 \\ &\vdots \\ S_k &= X_1 + X_2 + X_3 + \ldots + X_k \\ &\vdots \\ S_n &= X_1 + X_2 + X_3 + \ldots + X_k + \ldots + X_n \end{aligned}$$

3. Create a new sequence S' from S by attaching a zero at the beginning and at the end, forming $S' = 0, S_1, S_2, \ldots, S_n, 0$.
4. Let $J = \#\{\text{zero crossing in } S' \text{ with the exception of the first } 0\}$. Observe that $J = \#\text{zero crossings} = \#\text{Cycles}$.
   If $J < 500$ do not perform the test, since it doesn't satisfy the empirical rule for χ^2 computations.
5. For each cycle and for each non-zero state (i.e. for $-4, -3, -2, -1, 1, 2, 3, 4$) count frequencies of state within each circle.
6. For each state x, and for $k = 0, 1, \ldots, 5$, compute

   $$v_k(x) = \text{total number of cycles where state } x \text{ occurs exactly } k \text{ times},$$

   with the convention that $v_5(x)$ also stores frequencies which are ≥ 5.

7. Compute the following statistic of each of the eight states:

$$\chi^2(obs)(x) = \sum_{k=0}^{5} \frac{(v_k(x) - J\pi_k(x))^2}{J\pi_k(x)},$$

where $\pi_k(x)$ is the probability that state x occurs k times (i.e. occurs k time inside a cycle) under hypothesis of randomness H_0. (A table with probabilities follows the algorithm.)

8. Compute the P-value for each state x through the incomplete gamma function, so that 8 P-values are computed:

$$P\text{-}value(x) = igamc\left(\frac{5}{2}, \frac{\chi^2(obs)}{2}\right).$$

Probabilities to compare by means of the χ^2 distributions are (for completeness it contains value for $1 \leq x \leq 7$. Our implementation, as of NIST's, embedded values from -4 to 4):

	$\pi_0(x)$	$\pi_1(x)$	$\pi_2(x)$	$\pi_3(x)$	$\pi_4(x)$	$\pi_5(x)$
$x = 1$	0.5000	0.2500	0.1250	0.0625	0.0312	0.0312
$x = 2$	0.7500	0.0625	0.0469	0.0352	0.0264	0.0791
$x = 3$	0.8333	0.0278	0.0231	0.0193	0.0161	0.0804
$x = 4$	0.8750	0.0156	0.0137	0.0120	0.0105	0.0733
$x = 5$	0.9000	0.0100	0.0090	0.0081	0.0073	0.0656
$x = 6$	0.9167	0.0069	0.0064	0.0058	0.0053	0.0588
$x = 7$	0.9286	0.0051	0.0047	0.0044	0.0041	0.0531

We only reported values for positive x since, due to randomness hypothesis, probabilities are symmetric with respect to 0, i.e. $\pi_k(x) = \pi_k(-x)$.

Interpretation of results

A small P-value of a chosen state x, corresponds to large values of the statistic χ^2 and denotes large deviations from randomness of cycles of the random walk, i.e. the analyzed state is visited too often or too rarely by the random walk.

An Example

Let $\epsilon = 0110110101$ so that $n = 10$.

1. $X = -1, 1, 1, -1, 1, 1, -1, 1, -1, 1$.

2. Partial sums are:

$$\begin{array}{ll} S_1 = -1 & S_6 = 2 \\ S_2 = 0 & S_7 = 1 \\ S_3 = 1 & S_8 = 2 \\ S_4 = 0 & S_9 = 1 \\ S_5 = 1 & S_{10} = 2 \end{array}$$

and $S = \{-1, 0, 1, 0, 1, 2, 1, 2, 1, 2\}$.

3. The extended random walk is $S' = 0, -1, 0, 1, 0, 1, 2, 1, 2, 1, 2, 0$.

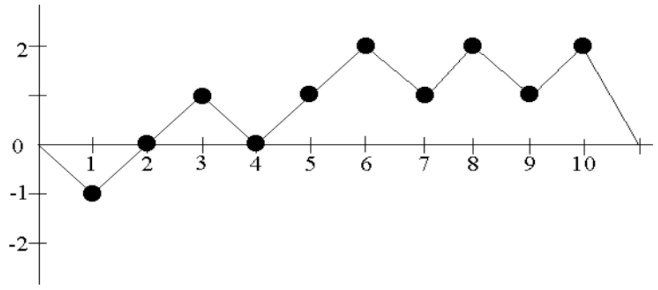

Figure 3.7: Random walk from the example. *Credits to [64].*

4. After the starting zero, we have $J = 3$ returns to the origin, since we have zeros in positions $3, 5$ and 12 of the extended walk. We therefore have 3 cycles, consisting of $\{0, -1, 0\}, \{0, 1, 0\}$ and $\{0, 1, 2, 1, 2, 1, 2, 0\}$.

5. Counting the frequency of each state we see that the first cycle has a single occurrence of -1, the second cycle has one occurrence of 1 and the last one has three occurrences of both 1 and 2. The following table provides a more clear visualization:

State x	**Cycle 1** $(\mathbf{0, -1, 0})$	**Cycle 2** $(\mathbf{0, 1, 0})$	**Cycle 3** $(\mathbf{0, 1, 2, 1, 2, 1, 2, 0})$
-4	0	0	0
-3	0	0	0
-2	0	0	0
-1	1	0	0
1	0	1	3
2	0	0	3
3	0	0	0
4	0	0	0

6. Computing $v_k(x)$ for $k = 0, 1, \ldots, 5$ we have that

- $v_0(-1) = 2$, since the state -1 occurs exactly 0 time ins two cycles;
  $v_1(-1) = 1$, since the state -1 occurs exactly one time in 1 cycle;
  $v_2(-1) = v_3(-1) = v_4(-1) = v_5(-1) = 0$, since the state -1 occurs exactly $2, 3, 4, \geq 5$ times, respectively, in 0 cycles.
- $v_0(1) = 1$ (the 1 state occurs exactly 0 times in 1 cycle);
  $v_1(1) = 1$ (the 1 state occurs only once in 1 cycle);
  $v_3(1) = 1$ (the 1 state occurs exactly three times in 1 cycle);
  $v_2(1) = v_4(1) = v_5(1) = 0$ (the 1 state occurs exactly $2, 4, \geq 5$ times in 0 cycles).

And so on, as in the following table:

State	**Number of cycles**					
x	**0**	**1**	**2**	**3**	**4**	**5**
-4	3	0	0	0	0	0
-3	3	0	0	0	0	0
-2	3	0	0	0	0	0
-1	2	1	0	0	0	0
1	1	1	0	1	0	0
2	2	0	0	1	0	0
3	3	0	0	0	0	0
4	3	0	0	0	0	0

7. For the state $x = 1$ the statistic is

$$\chi^2(1) = \frac{(1-3(0.5))^2}{3(0.5)} + \frac{(1-3(0.25))^2}{3(0.25)} + \frac{(0-3(0.125))^2}{3(0.125)} + \frac{(1-3(0.0625))^2}{3(0.0625)} + \frac{(0-3(0.0312))^2}{3(0.0312)} + \frac{(0-3(0.0312))^2}{3(0.0312)} = 4.333033.$$

8. *P-value*(1)= $igamc\left(\frac{5}{2}, \frac{4.333033}{2}\right) = 0.502529 > \alpha$ and the sequence is accepted as random, at least for what concerns the state $x = 1$: analogous computation has to be done for other values of x to find other P-values.

Mathematical background

In order to evaluate the χ^2 statistic at seventh step of the algorithm, we need to know the distribution of the number of visit for a random walk and, in particular, we need theoretical probabilities $\pi_j(x)$ for each state x and for each number of occurrences j. Let $S_k = X_1 + \ldots, +X_k$ be the k-th partial sum of the random walk where X_i are independent random variables which take values $+1$ and -1 with probability p and

$q = 1 - p$ respectively. Setting $S_0 = 0$ by definition, let $\rho_1 < \rho_2 < \cdots < \rho_J$ be the times when the random walk returns to the origin, i.e $\rho_1 = min\{k,\ k > 0 \text{ s.t. } S_k = 0\}$, $\rho_2 = min\{k,\ k > \rho_1 j \text{ s.t. } S_k = 0\} \ldots$.

The respective sequence of excursions will be

$$(S_0, \ldots, S_{\rho_1}), (S_{\rho_1}, \ldots, S_{\rho_2}), \ldots .$$

Let J denote the total number of such excursions in the string (corresponding to the number of zeros among partial sums S_k), which can be considered as a random variable itself, the limiting distribution of J is widely studied and known (see for example [60]) to be

$$\lim_{n\to\infty} P\left(\frac{J}{\sqrt{n}} < z\right) = \sqrt{\frac{2}{\pi}} \int_0^z e^{-u^2/2} du \quad \text{for } z > 0$$

If J is too small we'd have that the value of the integral is small and then the P-value would be small, since

$$P\text{-}value = P(J < J(obs)) \approx \sqrt{\frac{2}{\pi}} \int_0^J e^{-u^2/2} du,$$

that's why the randomness hypothesis is straight rejected if $J < max(0.005\sqrt{n}, 500)$ at fourth step of the algorithm and the test doesn't go on.

If the previous, preliminary, test is passed, let $\xi(x)$ be the number of visit to the state x (with $x \neq 0$, since this case is already evaluated), during a fixed excursion. The distribution of $\xi(x)$ is given by the following

Theorem 3.2.15.
For $p \neq 1/2$

$$P(\xi(x) = 0) = 1 - \frac{|p - q|}{\left|1 - \left(\frac{q}{p}\right)^x\right|} \tag{3.41}$$

and for $k = 1, 2, 3, \ldots$

$$P(\xi(x) = k) = \begin{cases} \frac{|p-q|^2}{|1-(q/p)^x|^2}\left[1 - \frac{|p-q|}{1-(q/p)^x}\right]^{k-1} & p > q, x > 0 \text{ or } p < q, x < 0, \\ \frac{|p-q|^2}{|1-(p/q)^x||1-(p/q)^x|}\left[1 - \frac{|p-q|}{1-(p/q)^x}\right]^{k-1} & p > q, x < 0 \text{ or } p < q, x > 0. \end{cases} \tag{3.42}$$

For $p = 1/2$

$$P(\xi(x) = 0) = 1 - \frac{1}{2|x|} \tag{3.43}$$

and for $k \geq 1$

$$P(\xi(x) = k)) \frac{1}{4x^2} \left(1 - \frac{1}{2|x|}\right)^{k-1}. \tag{3.44}$$

Proof.
First of all we shall observe that if the excursion is finite, then $\xi(x) = k$, with $k \geq 1$, if and only if the random walk S_k reaches the stage x for the first time, it fluctuates (without touching the stage 0) visiting the chosen state $k-1$ times and then it returns to the origin. Observe that if we consider the process $S_k - x$, that doesn't visit level $-x$ in its first $k-1$ excursions. Taking into account this fact, we can observe that, by independence of these excursions

$$p(\xi(x) = k, \rho < \infty) = P(\xi(x) > 0)[P(\xi(-x) = 0, \rho < \infty)]^{k-1} P(\xi(-x) > 0). \tag{3.45}$$

Let's suppose, for the moment, that $x > 0$, then

$$P(\xi(x) > 0) = P(S_1 = 1) P(\text{"x-1 is reached before -1"}).$$

The second factor of the right hand side it is equivalent to the probability of winning in the classical gambler's ruin problem with initial capital 1 and total capital x (see for example [22]), so that

$$P(\xi(x) > 0) = p \frac{q/p - 1}{(p/q)^x - 1} = \frac{q - p}{(q/p)^x - 1}.$$

Since we are considering a 1-dimensional random walk, to obtain probabilities of symmetrical (with respect to the origin) stage we can simply swap roles of p and q, obtaining

$$P(\xi(-x) > 0) = \frac{p - q}{(p/q)^x - 1},$$

and then

$$P(\xi(x) > 0) = \left| \frac{p - q}{(q/p)^x - 1} \right| \quad \forall\, x,$$

so that (3.41) is proved since, banally, $P(\xi(x) = 0) = 1 - P(\xi(x) > 0)$. To distinguish cases let's define I such that

$$I = \begin{cases} 0 & \text{if } p > q, x < 0 \text{ or } p < q, x > 0, \\ 1 & \text{otherwise,} \end{cases}$$

then

$$P(\xi(x) = 0, \rho = \infty) = (1 - I) P(\rho = \infty) = (1 - I)|p - q|.$$

By the previous and (3.41), taking into account that $P(\xi(x) = 0, \rho < \infty) = P(\xi(x) = 0) - P(\xi(x) = 0, \rho = \infty)$, we obtain

$$P(\xi(x) = 0, \rho < \infty) = 1 - \left|\frac{p-q}{(q/p)^x - 1}\right| - (1-I)|p-q|, \tag{3.46}$$

so that, by substituting it in (3.45), we obtain

$$P(\xi(x) = k, \rho < \infty) = (pq)^x \left(\frac{p-q}{p^x - q^x}\right)^2 \left[1 - \left|\frac{p-q}{(p/q)^x - 1}\right| - I|p-q|\right]^{k-1}.$$

Now, if the excursion is infinite, the case $\xi(x) = k$ occurs if and only if the random walk S_k visits the stage x for the first time, returns to it $k-1$ times (without visiting the origin) and than it continues without visiting both x and 0. Since we're considering an infinite excursion (so that infinite time), this case is only possible if $x > 0, p > q$ or $x < 0, p < q$ (i.e. the excursion continues above x, if $x > 0$, and below x, otherwise). This observation permits us to assert that $P(\xi(x) = k, \rho = \infty) = 0$ if $I = 0$.
Now, taking $-x$ in place of x in (3.46) we obtain

$$\begin{aligned} P(\xi(x) = k, \rho = \infty) &= IP(\xi(x) > 0)(P(\xi(-x) = 0, \rho < \infty))^{k-1} P(\rho = \infty) \\ &= \frac{I(p-q)^2}{1-(q/p)^x}\left(1 - \left|\frac{p-q}{(p/q)^x - 1}\right| - |p-q|\right)^{k-1}. \end{aligned}$$

Formula (3.42) is obtained by computing $P(\xi(x) = k, \rho = \infty) + P_{I_1}(\xi(x), \rho < \infty) + P_{I_2}(\xi(x), \rho < \infty)$, where I_1 and I_2 are used here to distinguish between the two possible cases.
Finally, by taking the limit $p \to \infty$ in (3.41) and (3.42), one obtains formulas (3.43) and (3.44) respectively. □

The case we are interested in is, obviously, the one with $p = q = \frac{1}{2}$: the theorem tells us that $\xi(x) = 0$ with probability $1 - \frac{1}{2|x|}$ and that, otherwise (with probability $\frac{1}{2|x|}$), it is a geometric random variable with parameter $\frac{1}{2|x|}$. From this we can easily evaluate its mean and variance, finding that

$$E[\xi(x)] = 1, \qquad Var(\xi(x)) = 4|x| - 2. \tag{3.47}$$

From considerations above, one can also find that

$$P(\xi(x) \le a+1) = 2xP(\xi(x) = a+1) = \frac{1}{|x|}\left(1 - \frac{1}{2|x|}\right)^a \quad \text{for } a = 0, 1, 2, \ldots \tag{3.48}$$

For each $x = -4, \ldots, -1, 1, \ldots, 4$, formulas (3.44) and (3.48) are used to evaluate probabilities π_j for $j = 0, 1, \ldots, 5$, which are necessary to compare to empirical frequencies v_j through the χ^2 distribution and, after that, to compute P-value using the incomplete gamma function *igamc*.

3.2.15 Random excursion variant test

Similarly to the previous test, this one focuses on the number of visits of the sequence, considered as a random walk, to different states. The purpose is to detect whether the behavior is compatible with the expected number of visits of a random sequence. This test analyzes stages $-9, -8, \ldots, -1, 1, \ldots, 8, \ldots, 9$, producing a total of 18 P-values.

Algorithmic description

1. The sequence is transformed into a string of -1 and $+1$.

2. Partial sums S_i are computed to obtain the set $S = \{S_i\}_{i=1,\ldots,n}$.

$$\begin{aligned}
S_1 &= X_1 \\
S_2 &= X_1 + X_2 \\
S_3 &= X_1 + X_2 + X_3 \\
&\vdots \\
S_k &= X_1 + X_2 + X_3 + \ldots + X_k \\
&\vdots \\
S_n &= X_1 + X_2 + X_3 + \ldots + X_k + \ldots + X_n
\end{aligned}$$

3. Create a new sequence S' from S by attaching a zero at the beginning and at the end, forming $S' = 0, S_1, S_2, \ldots, S_n, 0$.

4. For each state x, $x = -9, -8, \ldots, -1, 1, \ldots, 8, 9$ compute $\xi(x)$, the total number of times that state x occurs across all cycles (we indicate the number of cycles, i.e. the number of zero crossing but the first, with J).

5. For each x compute the P-value using the complementary error function:

$$P\text{-}value(x) = erfc\left(\frac{|\xi(x) - J|}{\sqrt{2J(4|x| - 2)}}\right).$$

Interpretation of results

As for the previous test, a small P-value occurs when there are strong deviations from the expected behavior of the random walk which, under randomness hypothesis, has to form a Gaussian curve.

An Example

Let $\epsilon = 0110110101$ so that $n = 10$.

1. $X = -1, 1, 1, -1, 1, 1, -1, 1, -1, 1$.

2. Partial sums are:

$$\begin{array}{ll} S_1 = -1 & S_6 = 2 \\ S_2 = 0 & S_7 = 1 \\ S_3 = 1 & S_8 = 2 \\ S_4 = 0 & S_9 = 1 \\ S_5 = 1 & S_{10} = 2 \end{array}$$

   and $S = \{-1, 0, 1, 0, 1, 2, 1, 2, 1, 2\}$.

3. The extended random walk is $S' = 0, -1, 0, 1, 0, 1, 2, 1, 2, 1, 2, 0$.

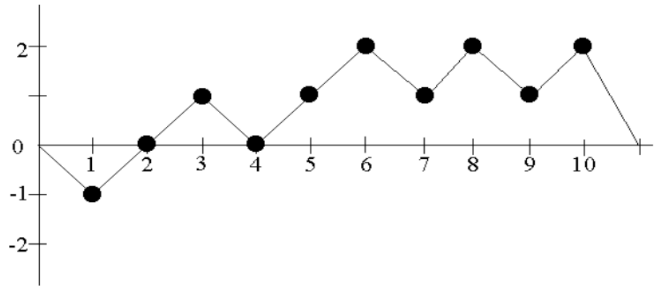

Figure 3.8: Random walk from the example. *Credits to [64].*

4. We have $J = 3$ returns to the origin and the total number of visits at each state are $\xi(-1) = 1, \xi(1) = 4, \xi(2) = 3$ and $\xi(x) = 0$ for all other x.

5. For example, for $x = 1$, $P\text{-}value(1) = erfc\left(\frac{|4-3|}{\sqrt{2 \cdot 3(4|1|-2)}}\right) = 0.683091 > \alpha$ and the sequence is accepted as random, at least for what concerns the state $x = 1$: analogous computation has to be done for other values of x to find other P-values.

Mathematical background

Using notations of the "Mathematical background" section of the previous test, let

$$\xi_J(x) = \#(\text{visits to stage } x \text{ during the } J\text{-th excursion}).$$

Under randomness hypothesis, for fixed x, $\xi_J(x)$ are independent and identically distributed, so that the limiting distribution $\xi(x) = \sum_J \xi_J(x)$ is normally distributed with mean and variance, respectively,

$$E[\xi(x)] = E[\sum_J \xi_J(x)] = J \cdot E[\xi_J(x)] = J$$

and

$$Var(\xi(x)) = Var(\sum_J \xi_J(x)) = J \cdot Var(\xi_J(x)) = J(4|x| - 2)$$

where mean and variance of $\xi_J(x)$ are exactly the ones of formulas (3.47) in the mathematical background of the previous test.

By direct use of the Central Limit Theorem (Theorem 2.1.2), we have that

$$\lim_{J\to\infty} P\left(\frac{\xi_J(x) - J}{\sqrt{J(4|x| - 2)}} < z\right) = \Phi(x),$$

where Φ is the cumulative distribution function of the Standard Normal distribution, as in (2.1). The P-value we are looking for is exactly the complementary of that probability, so that we can directly compare theoretical and empirical distribution by the complementary error function $erfc$.

Chapter 4

Properties of a RNG

In this section we want to study the efficiency of a RNG. As discussed in section (3.1), a single test on a single sequence is not enough to assess whether or not an RNG generates random sequences. A second level testing is necessary to give a response with a certain level of confidence, computing the probability that a truly random sequence s fails one or more tests. For briefness, in this section we will consider each sub-test as a different test so that we'll have a total of 188 test.

Let's suppose to test a single sequence, which length is consistent with tests requirements, on k tests. The probability that a truly random sequence s fails a test is α and the probability that it passes it is it's complementary $1-\alpha$, as from our hypothesis, so that the probability $P(k,\tilde{k})$ that $\tilde{k}$, with $\tilde{k} \leq k$, tests are passed has a binomial distribution (2.2.3), so that we obtain

$$P(k,\tilde{k}) = \binom{k}{\tilde{k}}(1-\alpha)^{\tilde{k}}\alpha^{k-\tilde{k}}. \tag{4.1}$$

By this formula, using $\alpha = 0.01$, the probability that a single sequence passes all the 188 tests (by considering each sub-test as a different test) is $0.99^{188} = 0.15 = 15\%$. This shows that we should take into account that, as a test-passing doesn't mean that our data is random, a fail doesn't give evidence of non randomness, but multiple tests have to be performed.

Sýs et al. ([69]) by using 100 GB of data from a good random generator ([19]) computed the empirical probability that a sequence fails exactly i test and the empirical probability that it fails i or more tests, comparing them with theoretical probabilities of formula (4.1). Their analysis was made both in the case that all the 188 test was applicable and in the case that Random Excursion test and its Variant were not applicable (since the Random Excursion tests require a sequences which length is on the order of

Failed tests (i)	0	1	2	3	4	5	6	7	8	9	10	11
theoretical	15.1	28.7	27.1	17.0	7.9	2.9	0.9	0.2	0.1	0.0	0.0	0.0
$P(i, 188)$	19.4	28.9	23.8	14.3	7.2	3.3	1.5	0.7	0.3	0.2	0.1	0.1
$C(i, 188)$	100	80.6	51.7	28.0	13.6	6.4	3.0	1.6	0.8	0.5	0.3	0.1
$P(i, 162)$	20.0	29.5	23.8	14.1	6.9	3.1	1.3	0.6	0.3	0.1	0.1	0.0
$C(i, 162)$	100	80.0	50.4	26.6	12.5	5.6	2.5	1.2	0.6	0.3	0.2	0.1

Table 4.1: Probabilities $P(i, 188)$ ($P(i, 162)$) that a truly random sequence fails exactly i of the whole 188 (162) tests. $C(i, 188)$ and $C(i, 162)$ are the probabilities that a sequence fails i or more tests of 188 and 162 respectively. Probability are reported in percentage. *Credits to [69]*

10^6). We report, in table 4.1, their results together with the theoretical probabilities obtained from the above formula.

These values can be used to make a first evaluation on randomness of a single sequence s: if for example a tested sequence, for which each tests are applicable, fails 5 of 188 tests, we have that 6.4% of random sequences are "equally or less random" than the sequence; $6.4\% > \alpha = 1\%$ so that randomness hypothesis could not be rejected, and the sequence is considered random.

Remark 4.0.12.
As the above table shows, a sequence should be considered non-random, with $\alpha = 0.01$, if it fails 8 or more tests.

The above discussion clarify that even if a sequence is random, we can decide if more specific test are needed to decide if a RNG efficiently generates random number, where more than one sequence is tested on a single test (4.1) and where more than one sequence is tested with more then one test, i.e. all tests (4.2).

4.1 Second level testing

Let's focus now on the case where a certain number k of sequences are tested using only one of the 188 available tests. NIST, in [64], proposes two ways to analyze results to perform a second level testing:

- to test the proportion of sequences passing a certain statistical test;
- to test the uniformity of obtained P-values from a certain statistical test.

We'll analyze, in the following, how to perform these tests and how to decide if the RNG generates random sequences.

4.1.1 Test for proportion of sequences passing a test

Let's suppose to test k sequences on a certain test and that the number of sequences passing it is $\tilde{k}$, whit $\tilde{k} \leq k$, so that $\nu = \frac{\tilde{k}}{k}$ is the proportion of sequences passing the test. This value has to fall between the confidence interval, which is defined by

$$(1 - \alpha) \pm c\sqrt{\frac{\alpha(1-\alpha)}{k}}, \tag{4.2}$$

where the value c is the critical value, i.e. the value which splits the domain of our statistic between the values which have a probability greater than the significance level α and lower than α, as shown in the figure below[1], and the term under the square root is the standard error of the statistic (standard normal, under randomness hypothesis). The two terms together give us the margin of error our analysis has: the more the sequences we test, the more this error is small.

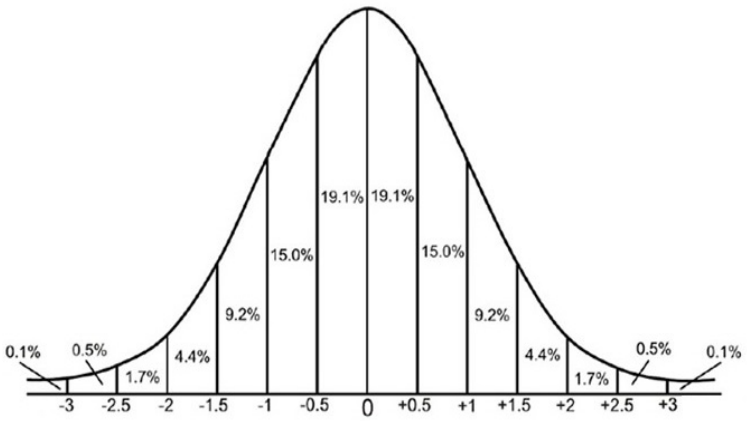

Figure 4.1: Standard Normal Distribution and critical values. *Credits to [47].*

If we choose $\alpha = 0.01$ and $c = 3$, as NIST, we have that, to pass a test, the proportion of sequence passing that has to be in the range

$$0.99 \pm 3\sqrt{\frac{0.01 \cdot 0.99}{k}} \tag{4.3}$$

so that if, for example, we are testing $k = 1000$ sequences, the passing rate has to fall in the interval 0.99 ± 0.0094392. If the proportion falls outside this interval, we consider the chosen test not passed by our RNG. We should observe that the value $c = 3$ is an

[1] Tables of critical values can be found, for example, in [1] or in `https://www.itl.nist.gov/div898/handbook/eda/section3/eda3671.htm`

approximation of the correct value $c = 2.6$ (reported in the tables) so that one should use it for a more precise testing. Note that the interval is computed approximating the Standard Normal Distribution, so that NIST recommends to test at least $k = 1000$ sequences. This test can be visualized as in figure (4.2).

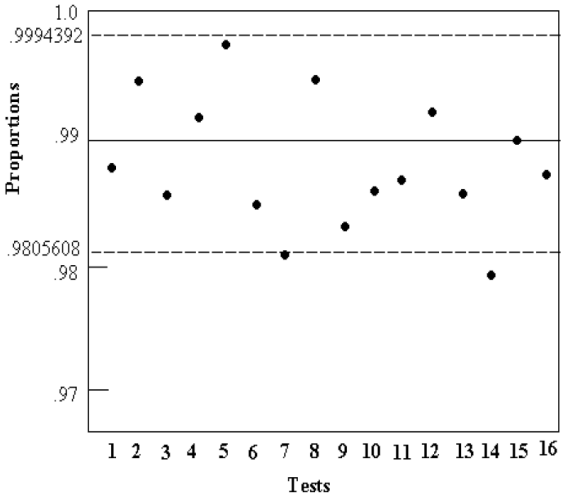

Figure 4.2: Proportion of sequences passing tests. In this example tests 7 and test 14 are not passed, according to this second level test. *Credits to [64].*

Since the confidence interval is computed by approximation of the Standard Normal Distribution, we couldn't evaluate cases where the number of testing sequence is lower than k=1000, however we could observe that the probability $P(k, \tilde{k})$ that $\tilde{k}$ of the sequences pass a test can be also modeled by the binomial distribution (2.2.3), so that we can find this probability using formula (4.1) again. Under randomness hypothesis, the distribution of probabilities is symmetric about the mean so that we could also compute the probability that the proportion ν of passing sequences falls into that interval by

$$P(\frac{k_1}{k} \leq \nu \leq \frac{k_2}{k}) = \sum_{i=k_1}^{k_2} \binom{k}{i} (1-\alpha)^i \alpha^{k-i}.$$

Unlike the one from NIST, this formula is modeled on the exact distribution and it isn't an approximation so that we can use it to test a smaller number of sequences. However we should take into account that, in this case, the loss on confidence could be significant.

4.1.2 Test for uniformity of P-values

Under randomness hypothesis H_0, P-values obtained by different sequences by the same test have to be uniformly distributed, so that this uniformity forms a hypothesis itself, which can be statistically tested. This can be done by a χ^2 goodness-of-fit test (3.1.1) by splitting the interval $[0, 1]$ which contains P-values into classes (NIST suite uses 10 classes), compute the frequency F_i of P-values falling into each classes e compare them by the use of a χ^2 statistic, which will have a number of degrees of freedom equal to the number of chosen intervals. By uniformity, if we are examining k P-values (i.e. k sequences), each theoretical frequency will be $k/10$, so that the statistic will have the form

$$\chi^2 = \sum_{i=1}^{10} \frac{(F_i - \frac{k}{10})^2}{\frac{k}{10}}.$$

As before we can compute the P-value of the statistic by the use of the incomplete gamma function, as

$$P\text{-}value_T = igamc(\frac{9}{2}, \frac{\chi^2}{2})$$

and see whether or not this P-values is greater than the significance level[2] α_T, which in this case should be set at 0.0001. If $P\text{-}value_T < 0.0001$ the generator is considered to create sequence too similar one another, and the test is considered not passed. We should note that this test provides a good approximation for $k/10 \geq 5.5$, so that at least 55 sequences as to be tested to obtain meaningful results.

This test can also be visualized by the use of an histogram, as in figure 4.1.2.

Remark 4.1.1.
We should take into account that failing this test doesn't necessarily mean that sequences are non-random themselves, but that the Random Number Generator is not a good random number generator since it creates sequences which are very similar one another. In other words, even if each sequence could behave as if it's effectively random, inspecting more sequences together shows evidence that created bits are non-random.

[2]Note that this significance level α_T refers the second level uniformity testing and it's independent by the level of significance α one chooses for individual tests

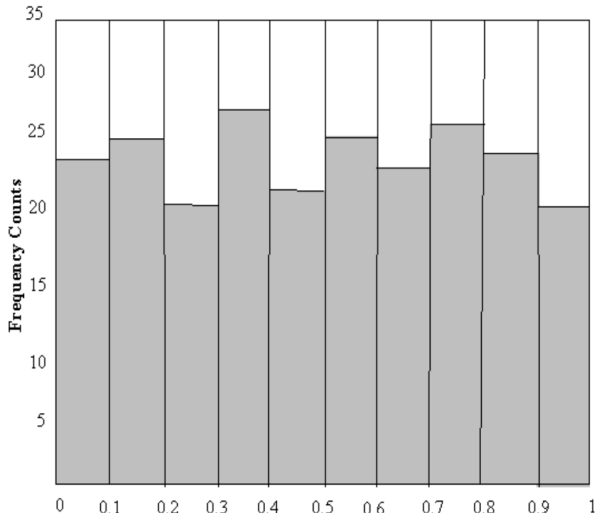

Figure 4.3: Visualization of the test for the uniformity of P-values. *Credits to [64].*

4.2 Complete testing

We want to discuss, in this section, the most common and complete case where many sequences are tested using all of the 188 tests of the suite. This final analysis can be performed by comparing P-values obtained from a set of tests by a set of sequence, by comparing the proportion of sequences passing more than one test and by comparing the P-values from the uniformity test.

4.2.1 Set of tests

If we are performing a set of tests on a set of sequences $s_1, \ldots, s_k$, we should take into account that on hypothesis of independence between tests, the probability to fail a test is the same of any other test and that the probability of passing more than one test (i.e. the intersection) factorizes. By this observation, it's clear that we can make an evaluation by the use of the table at the beginning of this chapter as follows:

i) For each test s_j, consider the number of tests $\hat{k}_j$ it failed.

ii) Let C_j be the probability that a test fails $\hat{k}_j$ or more tests (the applicability of the Random Excursion tests should be taken into account), compute the product of these probabilities

$$p = \prod_{j=1}^{k} C_j,$$

which is the probability that k random sequences are less random than $s_1, \ldots, s_k$.

iii) If $p > \alpha$, sequences $s_1, \ldots, s_k$ can be considered random, otherwise the randomness hypothesis is rejected.

For example, if we are testing two sequences s_1 and s_2 which failed respectively 1 and 2 tests we have that $C_1 = 0.806$ and $C_2 = 0.504$, so that the probability to have two sequences which are less random than the tested one is $p = 0.806 \cdot 0.504 = 0.406 > \alpha$, so that the two sequences are considered random.

4.2.2 Proportion of sequences passing more than one test

As we have seen in section (4.1.1), a set of k sequences could be examined through each test by the evaluation of the proportion of sequences passing that specific test. In order to give a more complete answer to the "does my RNG produces random sequences?" question, one should consider if the passing rate of RNG's output of the whole battery of tests (i.e. the inspection by any point of view) is consistent to what expected from a Random Number Generator which effectively generates random numbers. To do that, one can compare his results to the ones obtained by Sýs et al. in [69] by extensive simulations on sets of $k = 100$ and $k = 1000$ sequences. Results are reported, in the following table, by taking either the constant c of the test as approximated by NIST (i.e. $c = 3$) and the more accurate value $c = 2.6$.

Failed tests (i)	0	1	2	3	4	5	6	7	8	9	10	11	12
theoretical	15.1	28.7	27.1	17.0	7.9	2.9	0.9	0.2	0.1	0	0	0	0
observed	46.9	33.7	14.7	4.3	0.5	0	0	0	0	0	0	0	0
$C(10^3, 3)$	100	53.1	19.4	4.8	0.5	0	0	0	0	0	0	0	0
$C(10^3, 2.6)$	100	79.2	48.6	25.2	9.4	3.2	1.2	0.6	0.1	0	0	0	0
$C(10^2, 3)$	100	95.9	84.3	66.3	45.6	28.1	15.6	8.2	4.1	1.8	0.8	0.3	0.2

Table 4.2: Probabilities that random sequences fail exactly i out of 188 proportion tests, and cumulative probabilities that a set of $k = 100$ and $k = 1000$ sequences fail i or more proportion tests. Cumulative probabilities, obtained by extensive simulations, are computed both for $c = 3$ and $c = 2.6$. Values are reported in percentage. *Credits to [69]*

These probabilities can be used to assess the randomness of a set of sequences as follows: let's suppose to test $k = 1000$ sequences through the whole battery of tests and to perform the test for proportion to all of them. Assume that 4 tests of proportion are failed: using the constant $c = 3$ proposed by NIST, the probability to obtain an equivalent or worst case from a good-working RNG is 0.5% which is smaller than the significance level $\alpha = 1\%$; tested sequences can be considered non-random. On the other hand, if we assume the constant $c = 2.6$, the probability that sequences fail 4 or more test is 9.4% which is grater than the significance level α, so that we can consider our

sequences random. In cases like the one we just proposed is recommended to test another set of sequences obtained from the RNG, in order to make a more accurate answer.

Remark 4.2.1.
A set of $k = 100$ sequences should be considered non-random if it fails 11 or more test for the proportion of sequences passing a test.
Similarly, a set of $k = 1000$ sequences should be considered non-random if it fails 8 or more test for the proportion of sequences passing a test.
Take into account that the more the sequences we test, the more is the confidence we have from results: to test 1000 sequences is recommended, however, if for some reasons (such as computation limits) this is not possible, to make this test on a set of 100 sequences also produces reliable results.

4.2.3 Uniformity of P-values from multiple tests

Let's suppose to test $k = 100$ sequences and to make the test for uniformity of P-values (4.1.2) for each of the 188 test, obtaining 188 P-values (i.e. 188 passed/failed responses). Results can be compared to the probability obtained by Sýs et al. in [69] through extensive simulations on 8192 sets of $k = 100$ sequences, which are reported in the following table together with the theoretical probabilities.

Failed tests (i)	0	1	2	3	4	5	6	7	8	9
theoretical	15.1	28.7	27.1	17.0	7.9	2.9	0.9	0.2	0.1	0.0
observed	11.3	24.8	26.6	19.0	10.8	4.8	1.8	0.5	0.2	0.1
C, $\alpha_T = 1\%$	100	88.7	63.9	37.3	18.3	7.5	2.7	0.85	0.33	0.09
C, $\alpha_T = 0.1\%$	100	21.8	3.1	0	0	0	0	0	0	0
C, $\alpha_T = 0.01\%$	100	3.3	0.12	0	0	0	0	0	0	0

Table 4.3: Probabilities that 100 random sequences fail exactly i uniformity tests and cumulative probabilities that the set fails i or more uniformity tests for $\alpha_T = 0.01, 0.001, 0.0001$ (values are reported in percentage). Note that α_T is not the significance level of individual tests, but the significance level of the second level uniformity test. *Credits to [69]*

If, for example, we test 100 sequences using the whole 188 tests and the uniformity test is failed for 5 tests (i.e. 5 of the whole 188 P-values obtained from the test for uniformity of P-values are lower than 0.0001) the probability that this occurs for truly random sequences is 0, which is obviously smaller than $\alpha = 0.01\%$, so that the data could be considered non-random since the tested RNG produces sequences which are non uniform.

Remark 4.2.2.
A set of $k = 100$ sequences should be considered non-random if it fails 7 or more uniformity test (with the classical confidence level $\alpha = 0.01$). For smaller confidence levels, such as $\alpha = 0.001$ or $\alpha = 0.0001$, the maximum number of failed test to consider the set random is 2.
Testing only 100 sequences through this test could be considered enough since, as discussed in section (4.1.2), this test produces a good characterization of the random generator, however it doesn't bring a real answer on the randomness of the sequences. Anyhow, failing this test should be taken in account, since that indicates that the RNG produces sequences which could be similar one another, although their behavior could be considered random.

Chapter 5

Testing ECU's RNG

We studied how each of the 15 tests work, we saw how to perform a second level testing and how to evaluate whether a certain RNG is working properly or not, analyzing if randomness properties are satisfied and if generated sequences are sufficiently different one another. Now we want to describe how we checked the good behavior of our implementation and how to test the RNG embedded into an ECUs.

The main purpose of our work was to make an implementation which makes possible to completely test ECU's RNG by simply connect the target ECU to the computer, without the need of any other actions. To do that we decided, first of all, to implement the whole test suite by the use of MatLab [48] which allows to perform in the easiest way all mathematical tasks of each test. After that, we embedded the whole test suite in CANoe [70], a software which permits to interact with ECUs: we'll describe this step in section 5.2. We will report examples of each step of the implementation process in appendix A.

5.1 Validation of MatLab codes

To check our implementation we assumed that the one wrote by NIST in C language is correct and we compared P-values (and any other output) we obtained from our code to the ones one obtains testing the same sequences through the NIST test suite, available at `https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software`. Due to reproducibility reasons we decided to make the comparison using the binary expansion of some classical values, i.e. $\pi, e, \sqrt{2}$ and $\sqrt{3}$, truncated to 10^6 digits, which is the recommended length for sequences to test. Figures from 5.1 to 5.4 show the behavior of P-values for the four sequences and the relative error between our results and NIST's.

As figures show, the error of our implementation is attested between the machine

accuracy of MatLab, which is $2.2204 \cdot 10^{-16}$, and 10^{-8}. The grater error is obtained by test 11 and test 12, i.e. the serial test and the approximate entropy test: the error remains close to the MatLab epsilon and to 10^{-14} through the whole respective test, since values which carries the error (frequencies v_m, v_{m-1} and v_{m-2} for test 11 and the value of the approximate entropy $ApEn$) are multiplied, respectively, to $2^m = 2^{16} \sim 0.6 \cdot 10^5$ and to $n = 10^6$. According to error analysis theory, this multiplication produces the rising of errors to our values: anyway, we should observe that, since the level of significance α is typically chosen as 0.01 or 0.001 and it wouldn't be meaningful to choose a lower value, this error doesn't ruin our implementation since one would always be interested up to the third or fourth significant figure.

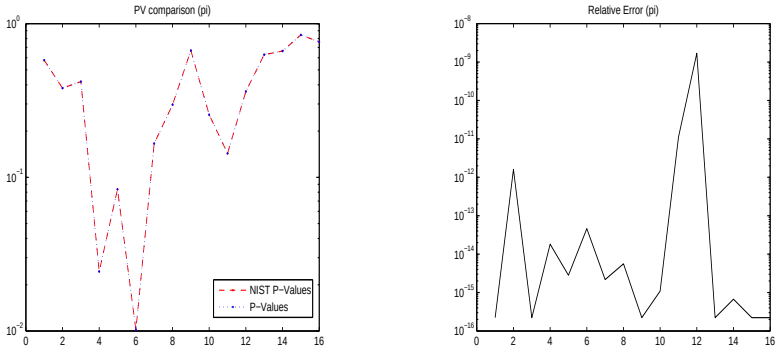

Figure 5.1: P-values and respective relative errors for binary expansion of π.

Remark 5.1.1.
Successive implementations (i.e. the Simulink models and their porting into CANoe) were also checked to ensure that each value one obtains using the CANoe implementation is correct: to do that we used values obtained from our MatLab code as the fixed point of the whole process, from single tests to the complete testing of 1000 sequences.

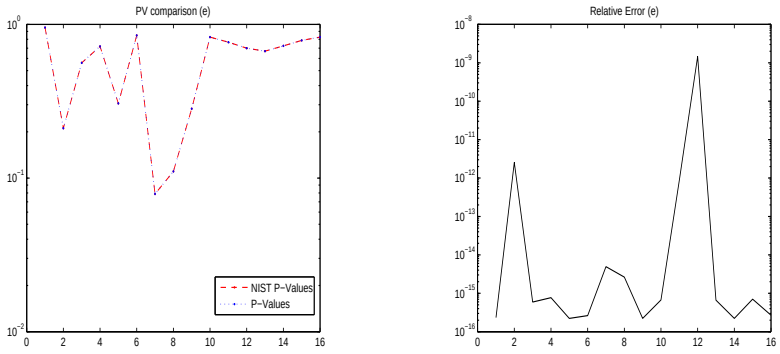

Figure 5.2: P-values and respective relative errors for binary expansion of e.

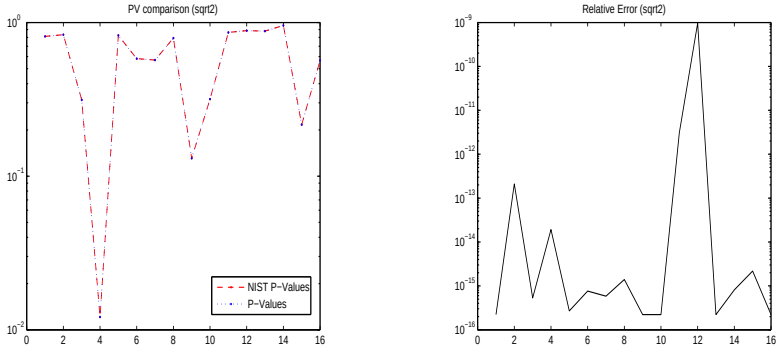

Figure 5.3: P-values and respective relative errors for binary expansion of $\sqrt{2}$

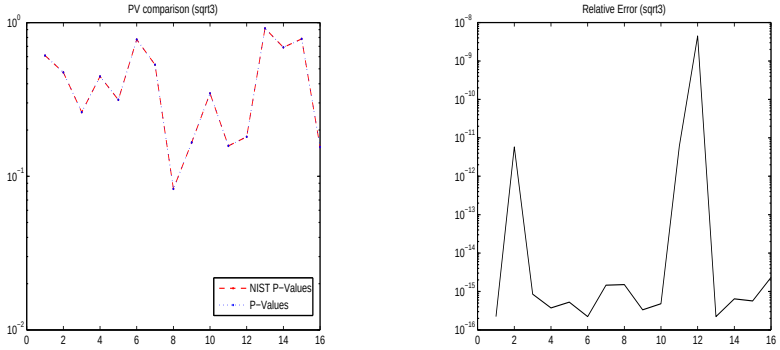

Figure 5.4: P-values and respective relative errors for binary expansion of $\sqrt{3}$

5.2 CANoe integration of test suite

Once we were sure that our MatLab codes were correct, we decided to bring that codes into CANoe and to use those same codes to test ECUs RNG. In this section we wish to give an overview of what CANoe is, how to use MatLab codes through the use of Simulink and Stateflow and how to embed and use these things into CANoe.

5.2.1 Vector CANoe

CANoe [70] is a software developed by Vector Informatik GmbH since 1996: it permits the user, principally automotive manufacturers and electronic control unit suppliers, to interface with an individual ECU or to a complete network in order to develop, make analysis, test and diagnosis. In particular CANoe can be used both to simulate a CAN-bus network (or a single ECU) without the need to materially build one, at the beginning of the production of a vehicle, and to diagnose malfunctions, when a car shows any type of issue.

Connecting to a network through CANoe (by the use of a computer or by any other dedicated hardware which supports it) it is possible to access the CAN-bus in order to read messages which flow in it or to send some to the system. Analogously, connecting an individual ECU allows to interact with it, to simulate the behavior when it is set together with other ECUs (the software could simulate them) or to (re)program it.

The user interface of CANoe is based on many windows which user can show, hide, move or customize: each of these windows have different functionality and permits both to extract information from the CAN-bus (e.g. to see messages on the bus, to read values of interest or to take trace of the telemetry) and to interact with the system (e.g. to push the brake pedal or to move the steering wheel, in a simulated environment or to send a specific message via CAN-bus, in both simulated or real environment). Figure 5.5 shows an example of these windows, together with their use.

Apart from windows showed in figure, another useful window is the *measurement* setup window, which permits, on one hand, to choose if the simulation is performed in online or offline mode and, on the other hand, to choose which value to observe or not; the measurement setup also manages the saving of logging, i.e. the trace of last measurement.

The most useful window for simulations is the *simulation* setup window, showed in figure 5.6: it permits to choose the environment of the simulation. This window is fundamental for development, since it allows the user to build the complete setup he wants to simulate. Through this window it is possible to add an arbitrary number of *blocks*: each block represents an element of the environment and could be a node (nodes represent ECUs of the simulated environments), a signal generator (it generates and sends

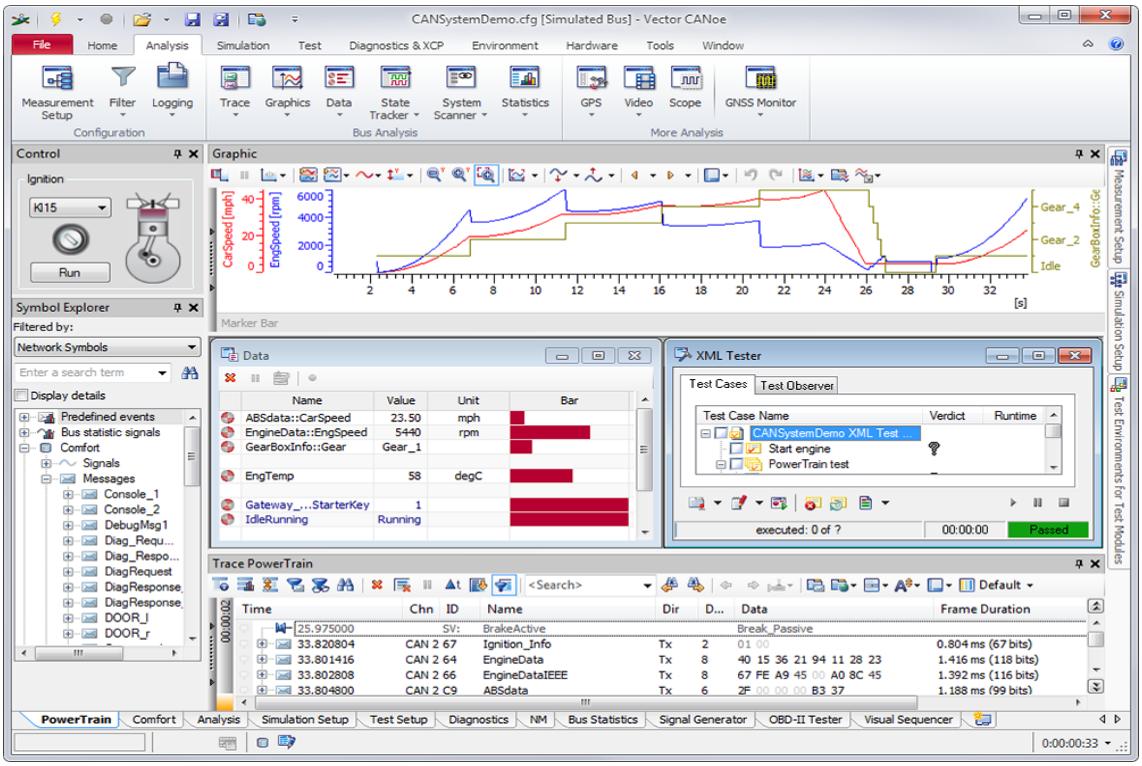

Figure 5.5: CANoe interface example. This example is made of a *control* interface by which user can simulate the ignition, a *graphics* and *data* interface by which it is possible to monitor values of interest (e.g. gear, engine and car speed as in this example), a *tester* window to choose the situation to test, an *explorer* to choose the object to observe and a *trace* window to trace messages which run inside of each CAN channel. *Credits to* [70].

signals to the bus; signals could be random or customized: they are used to simulate extra hardware or bus, in addition to the ones of interest), a replay block (it reproduces a previous simulation, sending the same signals) or logging block (stores signals/messages of interest). These blocks can be connected, in different ways, to create the desired simulation setup by simulated CAN-bus connections: it is possible to create (and eventually connect) different network, where each network is provided of a dedicated CAN-bus (each network can then be interfaced to one or more networks through an ECU which works as a gateway). The ECU node, which is the one we are most interested in, is the most customizable one, since it represents the control unit and could simulate each of its functions. It's possible to program each ECU node, defining its tasks, input and output: to do that each ECU node is provided with a script in CAPL[1] language.

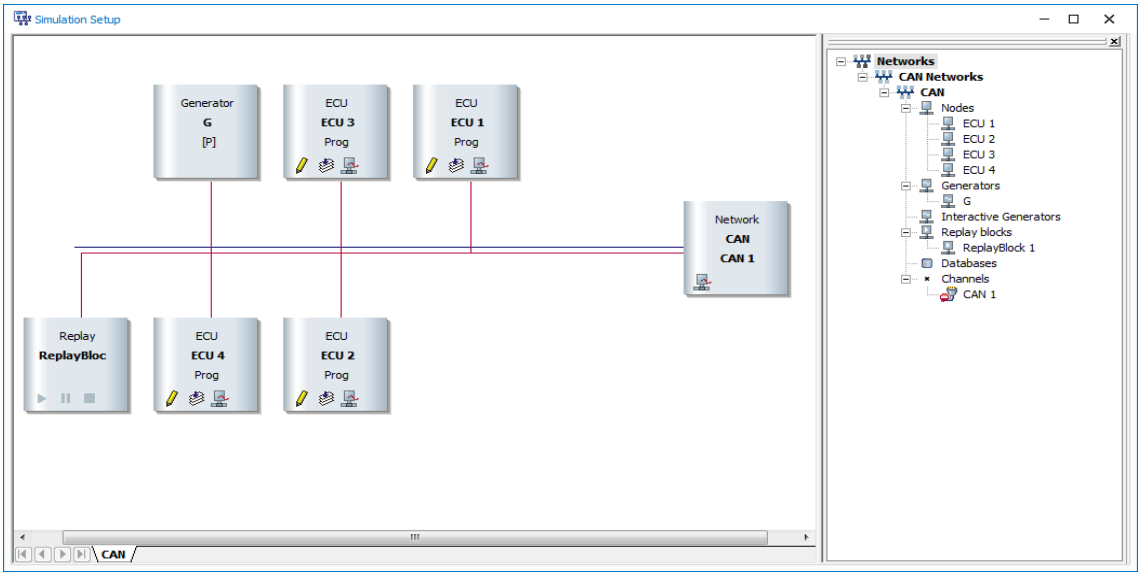

Figure 5.6: Example of a simulation setup window. This setup is made of four nodes which simulate ECUs (each of them endowed with its CAPL script), a signal generator and a replay block: all of these nodes are connected by a CAN-bus and form a unique network.

Finally, there is a *write* window where values from the execution of CAPLs are printed together with error in compilation or execution and any other information useful for the user.

[1]CAPL is a C-based programming language created by Vector Informatik GmbH: it contains most of C main functions to which new functions are annexed. CAPL specific functions permit to manage the communication between nodes, e.g. by input and output variables, detecting if external variable are changed or perform action when an external input is given (e.g. the vehicle is started).

5.2.2 From MatLab to CANoe

Since the way to directly communicate with an ECU is the CANoe software, together with a dedicated hardware to connect the ECU and the computer (showed in figure 5.7), we needed to export our MatLab functions to this software in order to use them. CANoe allows users to associate one or more DLLs[2] to each ECU node, and call it inside of the CAPL script: this feature permitted us to embed our MatLab functions inside of our configuration.

Figure 5.7: Vector Base Module: a dedicated hardware used to connect ECUs to a computer. *Credits to* [70].

To create DLLs to embed in the CANoe configuration we used *Simulink* [49], a MatWorks graphical programming environment that permits to create models where operations are performed by graphical objects in the form of blocks and arrows: a block could stand for a variable, a constant, an operator and so on, while arrows are used to connect blocks to define inputs, outputs and relations between them. Simulink, obviously, interface with MatLab: it is possible to use MatLab variables inside the Simuliunk model, to send values from the Simulink model to MatLab and also to use user defined MatLab functions through specific blocks (where the function can be wrote in a standard MatLab script). Simulink also interfaces with CANoe, allowing to convert models into DLLs which could be compiled and used by CANoe (through calls inside of the CAPL script): this can be done through the Vector CANoe add-on for Simulink, which provides the `cn.tlc` target for the compiler (we used the *Microsoft Visual C++ compiler* [52] to export Simulink models into DLLs).

Besides compatibility between Simulink and CANoe versions and formats, at this step, we had to take into account the limits of each software, in order to make the gen-

[2]Dynamic-Link Libraries: software libraries which can be dynamically loaded and compiled into a program or an executable file in order to execute them as part of the main software itself.

erated DLLs working on CANoe: the main problem we had to overcome was the fact that CANoe couldn't deal with too long variable arrays and sequences we had to test had to be at least 10^6 digits long. To overcome the problem we made use of the *Stateflow* logic tool for Simulink: Stateflow allows the user to create a finite-state machine[3] as a Simulink block (called *chart*); we inserted a Stateflow chart inside of our Simulink model so that CANoe could deal with and send shorter parts of the sequence: the chart accumulates the sequences till the end and, when it is complete, it sends the complete array to requested test(s).

Another requirement for the building of DLLs from Simulink was that, since generated DLLs use static memory allocation, we couldn't create a model which contained the whole test suite, so that we had to split tests inside of different models (depending on the size of each test code) and link them all to the CANoe node. Figure 5.8 shows created DLLs inside of the configuration window for a CANoe node.

Moreover, since Simulink allows to perform the discrete Fourier transform on vectors which length is a power of 2 only, our CANoe porting of the test suite requires it too: since, due to the asymptotic theory behind the test, 10^6 is the minimum allowed length we decided to require a length of $2^{20} = 1048576$ for sequence to test, that is the first power of 2 value which is greater than 10^6. This requirement only applies to the testing through CANoe (i.e. directly test ECU's RNG): indeed we used 2^{20}-long sequences when we tested ECU's RNG through the CANoe suite and 10^6-long strings in other cases, through the MatLab implemetation.

All other tasks are submitted to the CAPL script, inside of CANoe: reading sequences from files (or directly from the ECU), send inputs and outputs to models (that is done by the use of particular CANoe variables, called *system variables*[4]), store results into files and run the second level testing (also implemented by Simulink models) when a sufficient number of sequences was tested. After that, the CAPL checks if second level testings are passed or not, as described in the previous chapter, and writes final results on a file.

The general scheme of a complete testing execution could be summarized as follows:

1. the user chooses how many sequence to test (typically 100 or 1000 sequence are chosen), which test to perform, the desired level of significance and if he wants to

[3] A model of computation which passes from a state to another depending on external inputs (e.g. it performs the next step each time the function is called or each time a variable changes its value etc.).

[4] System variables are used in CANoe as variables which could be used in the whole configuration: while standard variables can be used inside each CAPL only (i.e. only for a node of the configuration), system variables can be used and modified by each node and sent to external functions, such as DLLs.

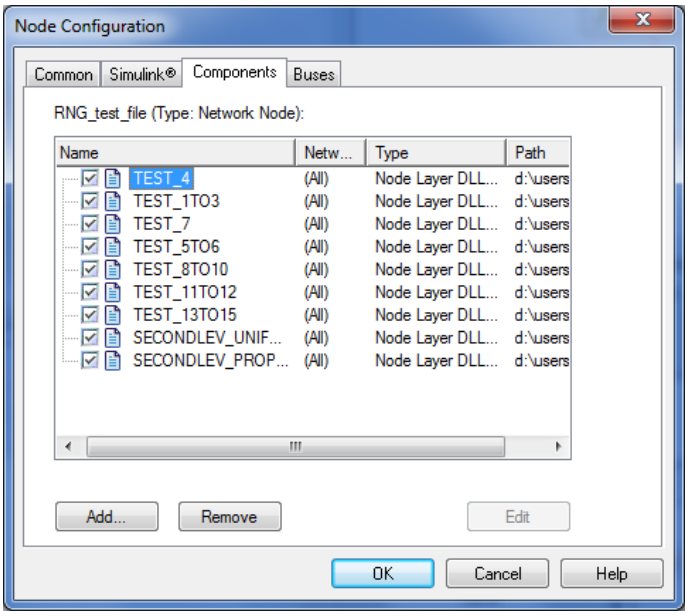

Figure 5.8: Linked DLLs to a CANoe node. This figure shows the configuration of our test suite.

perform the second level testing (in the case he tested enough sequence on every test);

2. CANoe reads the file which contains the binary sequence to test (or directly reads output values of ECU's RNG): a part of it is inserted into a system variables and sent to the DLL (i.e. the model), which waits and accumulate till the desired length is reached;

3. the Simulink model executes desired tests and sends results back to CANoe, by the use of some other system variables.

That is done for each requested test on the whole set of sequences and results are stored into file.

If the second level testing is requested, the process goes on:

4. results and P-values, of a specific test for all the tested sequences, are red from files which contains them and stored into dedicated system variables;

5. system variables are sent to other Simulink models that perform the second level testing and send results to CANoe.

This is performed for each test, obtaining 188 results from the test for proportion of passing and 188 results from the test for the uniformity of P-values: results are stored

into files.

After that, the CAPL (i.e. CANoe) reads results from this second step and writes a file which contains the final result (i.e. if or not the RNG passed the complete testing) using bounds of remarks 4.2.1 and 4.2.2, together with information from eventually not-passed tests and possible problems of the Generator.

5.2.3 Results from our implementation

Once we implemented the test suite, we decided to use it to test, apart from the Random Number Generator embedded into two different ECUs, the `randi` function of MatLab. This section reports main results of these validations.

`randi` results

The `randi` function creates random integers (single values, vectors or matrices) through a Pseudo Random algorithm: the PRNG used for this function is the same of the `rand` and `randn` functions. `randi` generates values between 1 and the input, so that to obtain a binary sequence one can use 2 as the max value and subtract it by a unitary vector, so that to generate a 10^6-length vector one can simply use the command $\texttt{randi}(2, [10^6, 1]) - 1$. We tested 1000 sequences, performing the whole test suite and, after that, the second level testing, in order to evaluate the goodness of the function. Table 5.1 shows the proportion of passing for each test, stored for type of test (i.e. test composed by more sub-tests are reported together)

For 1000 sequences, using bounds of formula (4.3), the proportion test is considered to be passed if the proportion of passing lies inside the interval $[0.9805, 0.9994]$, so that, to pass the proportion test, at least 981 of 1000 sequences have to pass a single test. In this case, the function `randi` passed 185 of 188 test for proportion (i.e. more than 981 strings on 1000 passed a single test) and 187 of 188 uniformity test so that both test were passed and the PRNG embedded in MatLab could be considered to be a well designed and good working Random Number Generator (the acceptance ranges for 1000 sequences are 180/188 and 182/188, respectively).

Histograms reported in figures 5.9 and 5.10 represent an example for the distribution of P-values of a passed and a failed uniformity test: respectively one of sub-tests of the random excursion test and one of the sub-test of the non-overlapping template matching test (the only one which didn't pass the test for uniformity of P-values). As for the test, P-values are stored into 10 classes between 0 and 1.

Test	Proportion of passing	Uniformity pass rate
Frequency	979/1000	1/1
Frequency within a block	990/1000	1/1
Runs	988/1000	1/1
Longest run of ones	990/1000	1/1
Matrix rank	987/1000	1/1
DFT	994/1000	1/1
Non-overlapping	146526/148000	147/148
Overlapping	992/1000	1/1
Maurer	987/1000	1/1
Linear complexity	991/1000	1/1
Serial	1982/2000	2/2
Approximate Entropy	991/1000	1/1
Cumulative sums	1969/2000	2/2
Random excursions	7893/8000	8/8
Random excursions variant	17786/18000	18/18

Table 5.1: Results from second level testing of `randi` output: eventual sub-tests are added together.

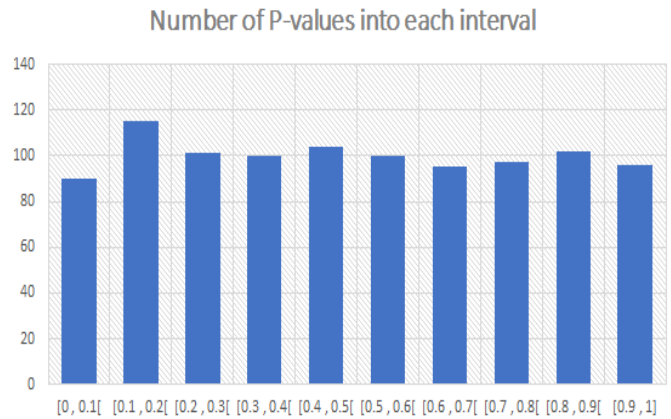

Figure 5.9: Histogram for a passed uniformity test: random excursion test (one of the 8 sub-tests). The number of P-values into each class is sufficiently close to a uniform distribution.

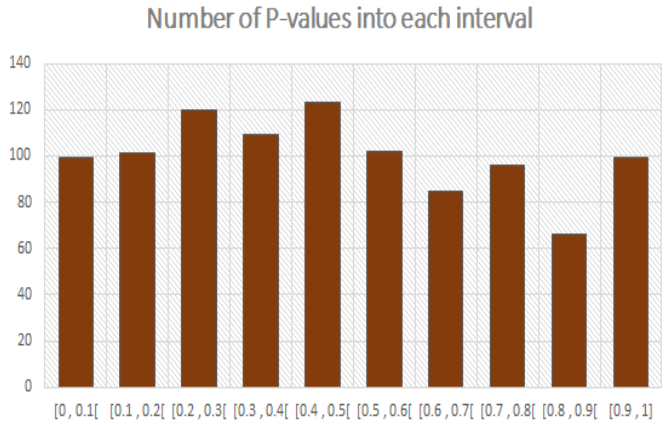

Figure 5.10: Histogram for a failed uniformity test: non-overlapping template matching test (one of the whole 148). The number of P-values into each class is far from a uniform distribution: in particular, compared to other classes, there is a huge disproportion between the number of P-values falling into the interval $[0.8, 0.9[$ (66 P-values).

TRNG results

ECUs at our disposal were provided with a True Random Number Generator, produced by Infineon Technologies [31], of which we was not aware of the structure: in this case it's very important to validate the Generator, in order to make sure that everything works as it has to do. Moreover, apart from the structure of the TRNG (which could be known or unknown), it often happens that RNG's output values are used and processed by various software functions: it becomes fundamental, then, to ensure that these functions don't spoil produced randomness, for example by adding some recurrences or hidden structures.

Due to the, relatively higher compared to the MatLab PRNG, time to generate a huge amount of digits and the need to convert these value from binary format to ASCII, we decided to perform the testing on 100 sequences only which length, due to the porting by the use of Simulink (as described in the previous section), was 2^{20}.
Table 5.2 reports results we obtained from the second level testing applied to ECU's TRNG, in the same format of results from the `randi` function.

For 100 sequences, using bounds of formula (4.3), the proportion test is considered to be passed if the proportion of passing lies inside the interval $[0.9641, 1.0159]$, so that, to pass the proportion test, at least 97 of 100 sequences have to pass a single test.
In this case, the TRNG passed 178 of 188 test for proportion and 183 of 188 uniformity test (the acceptance ranges for 100 sequences are 177/188 and 182/188, respectively), so that both test were passed and the TRNG could be considered to be a well designed and

good working Random Number Generator.

Test	Proportion of passing	Uniformity pass rate
Frequency	99/100	1/1
Frequency within a block	95/100	1/1
Runs	98/100	1/1
Longest run of ones	99/100	1/1
Matrix rank	99/100	1/1
DFT	96/100	1/1
Non-overlapping	14615/14800	145/148
Overlapping	99/100	1/1
Maurer	100/100	1/1
Linear complexity	99/100	1/1
Serial	198/200	1/2
Approximate Entropy	99/100	1/1
Cumulative sums	200/200	2/2
Random excursions	792/800	7/8
Random excursions variant	1764/1800	17/18

Table 5.2: Results from second level testing of TRNG's output: eventual sub-tests are added together.

As for the previous, histograms reported in figures 5.11 and 5.12 represent an example for the distribution of P-values of a passed and a failed uniformity test: respectively the linear complexity test and one of the sub-tests of non-overlapping template matching test.

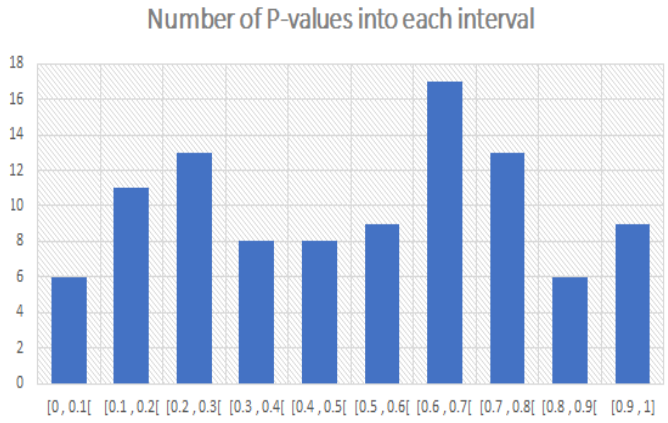

Figure 5.11: Histogram for a passed uniformity test: linear complexity test. The number of P-values into each class is sufficiently close to a uniform distribution.

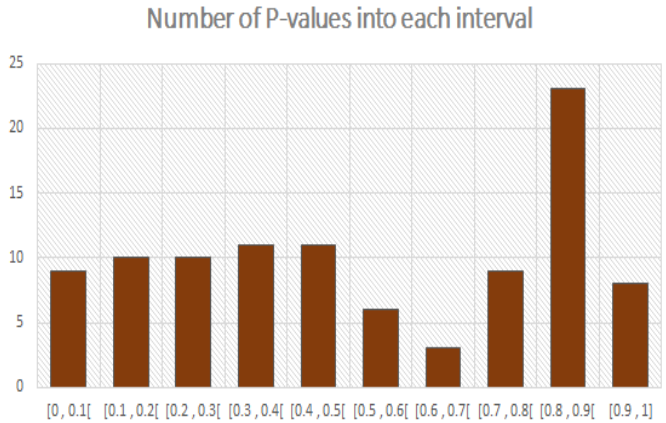

Figure 5.12: Histogram for a failed uniformity test: non-overlapping template matching test (one of the whole 148). The number of P-values into each class is far from a uniform distribution: in particular, compared to other classes, there is a disproportion between the number of P-values falling into intervals $[0.6, 0.7[$ (3 P-values) and $[0.8, 0.9[$ (23 P-values).

PRNG results

We also had the opportunity to test the PRNG embedded into an old ECU which generated numbers using a not performing method. The inspected PRNG generated numbers between 0 and 255 in hexadecimal format: in order to test them by the test suite, we converted numbers into binary format and we built 2^{20}-long sequences by concatenating them.

We tested, as for the TRNG, 100 sequences: the PRNG didn't pass the test, since generated sequences passed only 53 of 188 tests for proportion and 44 of 188 uniformity tests. That shows that generated numbers were non-random and that they were similar one another. Table 5.3 reports results from the second level testing.

Test	**Proportion of passing**	**Uniformity pass rate**
Frequency	0/100	0/1
Frequency within a block	0/100	0/1
Runs	0/100	0/1
Longest run of ones	75/100	0/1
Matrix rank	100/100	1/1
DFT	0/100	0/1
Non-overlapping	8198/14800	28/148
Overlapping	3/100	0/1
Maurer	51/100	0/1
Linear complexity	100/100	1/1
Serial	103/200	1/2
Approximate Entropy	0/100	0/1
Cumulative sums	0/200	0/2
Random excursions	755/800	3/8
Random excursions variant	1771/1800	10/18

Table 5.3: Results from second level testing of PRNG's output: eventual sub-tests are added together.

Histograms reported in figures 5.13 and 5.14 show an example of a passed and a failed uniformity test, respectively: the binary matrix rank test and the longest run of ones in a block test.

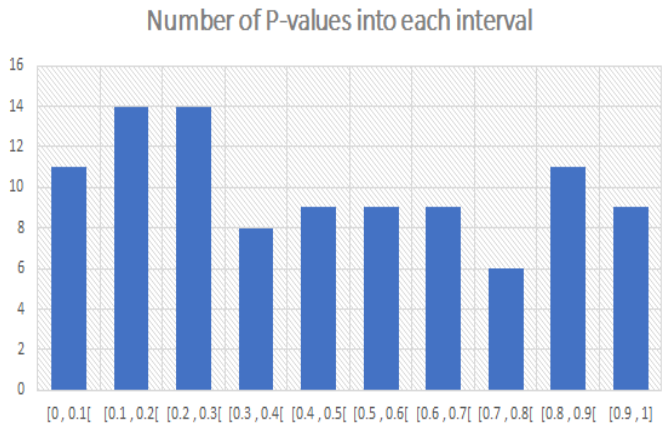

Figure 5.13: Histogram for a passed uniformity test: binary matrix rank test. The number of P-values into each class is sufficiently close to a uniform distribution.

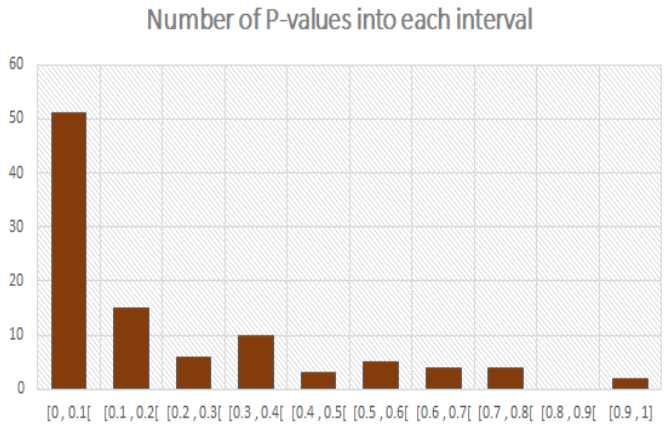

Figure 5.14: Histogram for a failed uniformity test: longest run of ones in a block. The number of P-values into each class is far from a uniform distribution: in particular, compared to other classes, there is a huge disproportion between the number of P-values falling into interval $[0, 0.1[$ (51 P-values) and other intervals.

Conclusions

We opened this thesis with an overview of the ECUs structure of modern vehicles, focusing on possible entry points and hackings. We described typical use of randomness in cryptography and, in particular, of how random numbers are used in the cyber security system of vehicles together with an idea of what a lack on randomness, derived from a problem of the embedded RNG, could imply.

The main aim of this thesis is to study randomness so that we analyzed the general procedure to perform a statistical test. That permitted us to focus on the more specific case of statistical tests for randomness, by describing it and proposing theoretical bases and proofs for each them. Tests are then placed together to obtain the answer we were looking for, i.e. to know whether or not the RNG is working properly, through a second level testing on previously obtained results.

We finally reached the point where we used all of these information to build a method to validate RNGs: we did that through MatLab and Simulink, in order to use them into CANoe. Finally we could use our implementation to, positively, evaluate randomness of the True Random Number Generator embedded into our ECU and of the MatLab's Pseudo Random Generator.

This constructions obviously has its limits: by this test suite it is impossible to distinguish a Pseudo Random Number Generator from a True one, and that could be a not trivial trouble for cryptographic purposes. Moreover, the 15 tests of this suite couldn't fulfill all possible aspect of randomness (in fact none finite number can do it) so that this suite could be extended, if new tests uncover.

Appendix A

Examples from implementation

This appendix is intended to give the reader an example of our implementation, presenting two samples of MatLab code for single test (in particular codes described in 3.2.11 and 3.2.14) and, after those, an example of one of the Simulink models (the one relevant to test from 1 to 3: other models are approximately the same).

```
function [R,PVs,S_obs,err]=serial_test(s,alpha,m)
% Test 11: Serial test
% INPUT:    s = binary string to test (array)
%           alpha = level of significance
%           m = length of templates to analyze
% OUTPUT:   R = results (#2)
%           PVs = P-value for each statistic (#2)
%           S_obs = values of the statistic (#2)
%           err = error from the execution
% RECOMM: choose m < log2(n)-2

% default length of window
if nargin==2
    m=16;
end

% pre-allocation of used values
R=NaN*zeros(2,1);
PVs=R;
S_obs=R;
err=ERRORS.NO_ERROR;

% sequence is non-empty
```

```
24 n=length(s);
25 if n==0
26     err=ERRORS.GENERIC;
27     return
28 end
29
30 % form augmented sequence and initialize frequency vectors
31 s=vertcat(s,s(1:m-1));
32 vm=zeros(2^m,1);
33 v1=zeros(2^(m-1),1);
34 v2=zeros(2^(m-2),1);
35
36 % count occurrences
37 i=1;
38 while i+m-1<=length(s)
39     if m==0 || m==-1
40         break
41     end
42     % convert patterns into respective decimal value (index)
43     index=binvec2dec(s(i:i+m-1))+1;
44     vm(index)=vm(index)+1;
45     i=i+1;
46 end
47 i=1;
48 while i+m-1<=length(s)
49     if m-1==0 || m-1==-1
50         break
51     end
52     index=binvec2dec(s(i:i+m-2))+1;
53     v1(index)=v1(index)+1;
54     i=i+1;
55 end
56 i=1;
57 while i+m-1<=length(s)
58     if m-2==0 || m-2==-1
59         break
60     end
61     index=binvec2dec(s(i:i+m-3))+1;
62     v2(index)=v2(index)+1;
63     i=i+1;
64 end
```

```
65
66  % compute values of statistics
67  psim=0;
68  psi1=0;
69  psi2=0;
70  if m~=0 && m~=-1
71      psim=2^m/n*sum(vm.^2)-n;
72  end
73  if m-1~=0 && m-1~=-1
74      psi1=2^(m-1)/n*sum(v1.^2)-n;
75  end
76  if m-2~=0 && m-2~=-1
77      psi2=2^(m-2)/n*sum(v2.^2)-n;
78  end
79
80  % compute distances
81  nabla=psim-psi1;
82  nabla2=psim-2*psi1+psi2;
83  S_obs(1)=nabla;
84  S_obs(2)=nabla2;
85
86  % compute P-values and check results
87  PVs(1)=gammainc(nabla/2,2^(m-1)/2,'upper');
88  PVs(2)=gammainc(nabla2/2,2^(m-2)/2,'upper');
89
90  if PVs(1)<alpha
91      R(1)=RESULTS.FAILED;
92  else
93      R(1)=RESULTS.PASSED;
94  end
95
96  if PVs(2)<alpha
97      R(2)=RESULTS.FAILED;
98  else
99      R(2)=RESULTS.PASSED;
100 end
```

```
function [R,PVs,S_obs,err]=random_excursions_test(s,alpha)
% Test 14: Random excursions test
% INPUT:    s = binary string to test (vector)
%           alpha = level of significance
% OUTPUT:   R = results (#8)
%           PVs = P-value for each state x (#8)
%           S_obs = values of statistics (#8)
%           error = error from the execution

% pre-allocation of used values
R=NaN*ones(8,1);
PV=R;
S_obs=R;
err=ERRORS.NO_ERROR;

% sequence is non-empty
n=length(s);
if n==0
    err=ERRORS.GENERIC;
    return
end

% Compute partial sums (random walk)
s=2*s-ones(size(s));        % 0,1 to -1,1
S=zeros(n,1);
S(1)=s(1);
for i=2:n
    S(i)=S(i-1)+s(i);
end

% Compute number of 0s in S2 (i.e. #cycles)
S2=[0; S; 0];
J=nnz(~S2)-1;
if J<500                    % check on number of cycles
    R=RESULTS.FAILED;
    return
end

% compute occurrences of values inside each cycle
table=zeros(8,J);
j=1;
```

```
42 i=2;
43 while j<=J
44     while S2(i)~=0
45         el=S2(i);
46         if el>-5 && el<5
47             if el<0
48                 table(el+5,j)=table(el+5,j)+1;
49             else
50                 table(el+4,j)=table(el+4,j)+1;
51             end
52         end
53         i=i+1;
54     end
55     i=i+1;
56     j=j+1;
57 end
58
59 % compute frequencies, statistic and P-value for each state
       x=-4,...,-1,1,...4
60 PVs=zeros(8,1);
61 S_obs=zeros(8,1);
62 for i=1:8
63     v=zeros(6,1);
64     for j=0:4                    % compute frequencies
65         v(j+1)=sum(table(i,:)==j);
66     end
67     v(6)=sum(table(i,:)>4);
68     p=p_table_aux(i);            % pre-allocate probabilities
69     chi=chi_stat(v,p,J);         % chi-squared statistic
70     S_obs(i)=chi;
71     PVs(i)=gammainc(chi/2,5/2,'upper');      % P-values
72     if PVs(i)<alpha                          % results
73         R(i)=RESULTS.FAILED;
74     else
75         R(i)=RESULTS.PASSED;
76     end
77 end
```

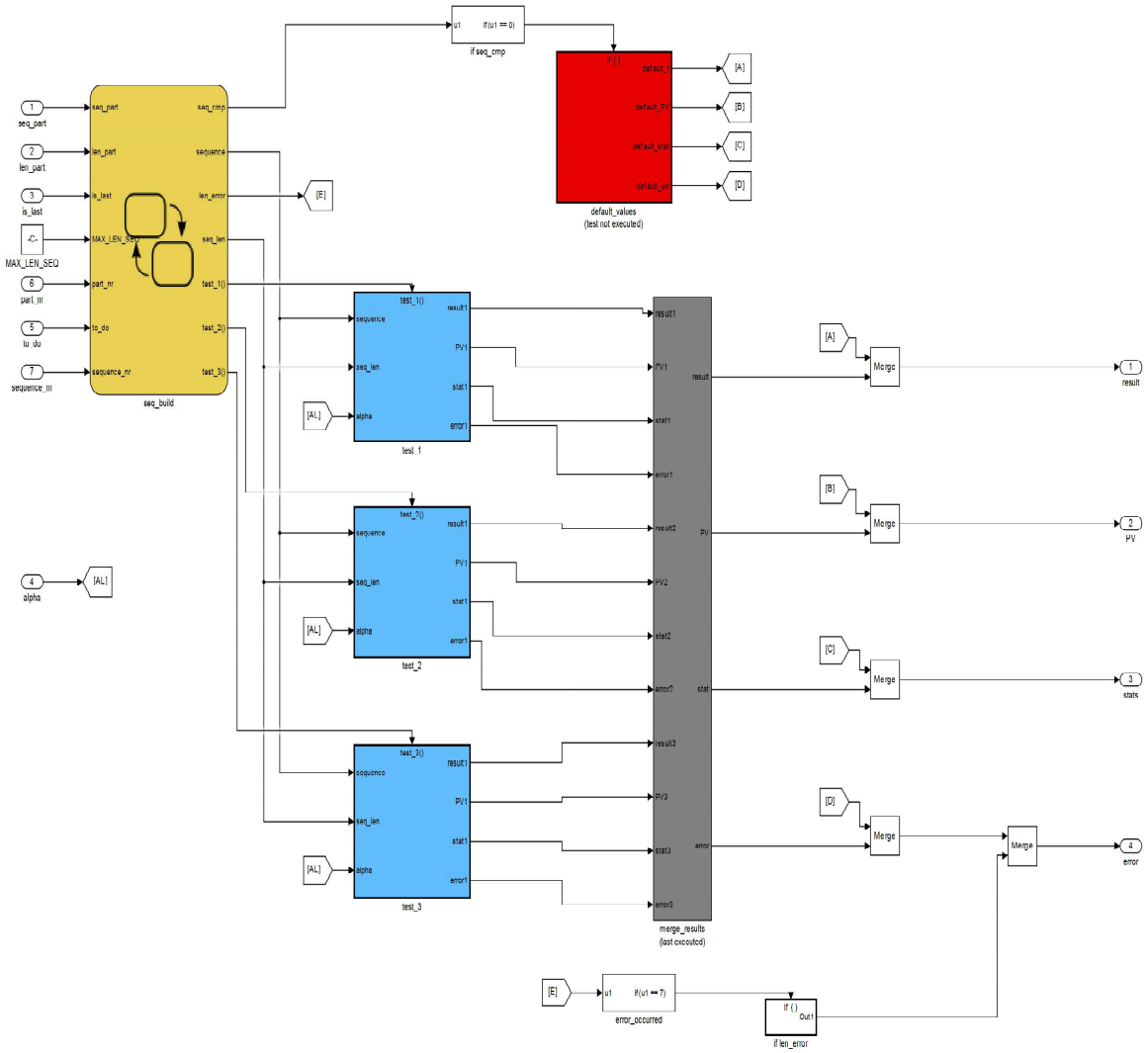

Figure A.1: Example of Simulink model. The yellow block is the Stateflow chart, where the sequence is build from 1000-bit sub-blocks; the red block initializes outputs to -1: if the test is not executed, the only value of an eventual error is updated; blue blocks contain MatLab functions for tests; the gray block updates outputs to the last executed test.

Bibliography

[1] Abramowitz M. and Stegun I., *Handbook of Mathematical Functions*, U.S. Government Printing Office, Washington, 1967.

[2] ANSI X9.62-2005, *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm*, American National Standard for Financial Services, 2005.

[3] Baron M. and Rukhin A., *Distribution of the Number of Visits for a Random Walk*, Communications in Statistics: Stochastic Models Vol. 15, 1999, pp. 593-597.

[4] Barker E. and Kelsey J., *NIST Special Publication 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators*, National Institute of Standards and Technology, 2012.

[5] Bartlett M. S., *The frequency goodness of fit test for probability chains*, Proc. Cambridge Philos. Soc. 47. 1951, pp. 86-95.

[6] Billingsley P., *Convergence of Probability Measure*, J. Wiley, New York, 1968.

[7] Bracewell R. N., *The Fourier Transform and Its Applications*, McGraw-Hill, New York, 1986.

[8] Bragaglia E., *Analysis and partial solutions of security problems in automotive environments*, Master's thesis in Mathematics, University of Bologna, 2013.

[9] Burgin M., *Generalized Kolmogorov complexity and duality in theory of computations*, Notices of the Russian Academy of Sciences, 1982.

[10] Caelli W., *Crypt-X package documentation*, Tech. Rep., Information Security Research, 1992.

[11] Calude C. S., *Information and Randomness: An Algorithmic Perspective*, Springer-Verlag, 2002, pp. 95-233.

[12] Chaitin G., *On the Simplicity and Speed of Programs for Computing Infinite Sets of Natural Numbers*, Journal of the ACM, 1969, pp. 407-422.

[13] Chaitin G., *Randomness and mathematical proof*, Scientific American. 1975, pp. 47-52.

[14] Chevillard S., *The functions erf and erfc computed with arbitrary precision and explicit error bounds*, Information and Computation 216, 2012, pp. 72-95.

[15] Chung K. L., *Elementary Probability Theory with Stochastic Processes*, Springer-Verlag, New York, 1979, pp. 210-217.

[16] Chrysaphinou O. and Papastavridis O., *A Limit Theorem on the Number of Overlapping Appearances of a Pattern in a Sequence of Independent Trials*, Probability Theory and Related Fields Vol. 79, 1988, pp. 129-143.

[17] Coron J-S. and Naccache D., *An accurate Evaluation of Maurer's Universal Test*, Selected Areas in Cryptography. SAC 1998. Lecture Notes in Computer Science, Springer, Berlin, 1999.

[18] David F. N. and Barton D. E., *Combinatorial Chance*, Hafner Publishing Co., Mew York, 1962, pp. 228-230.

[19] Department of Physics of Humboldt University and Pico-Quant GmbH: nano-Optics groups, *QRNG Service*, `www.qrng.physik.hu-berlin.de`.

[20] Diananda P. H., *The Central Limit Theorem for m-dependent variables asymptotically stationary to second order*, Proc. Camb. phil. Soc. 50, 1954, pp. 92-95

[21] Durrett R., *Probability: Theory and Examples*, Cambridge University Press, Cambridge, 2010.

[22] Feller W., *An introduction to probability theory and its applications*, J. Wiley, New York, 1968.

[23] Földes A., *The limit distribution of the length of the longest head-run*, Periodica Mathematica Hungarica Vol. 10, Budapest, 1979, pp. 301-310.

[24] Gibbons J. D. and Chakraborti C., *Nonparametric Statistical Inference*, Marcel Dekker, New York, 2003, pp. 50-58.

[25] Godbole A. P. and Papastavridis S. G., *Runs and Patterns in Probability: selected papers*, Kluwer Academic, Dordrecht, 1994.

[26] Good I. J., *The serial test for sampling numbers and other tests for randomness*, Proc. Cambridge Philos. Soc. 47, 1952, pp. 276-284.

[27] Gustafson H., Dawson E., Nielsen L. and Caelli W., *A computer package for measuring the strength of encryption algorithms*, Computer and Security 13, 1994, pp. 687-697.

[28] Hamano K., *The Distribution of the Spectrum for the Discrete Fourier Transform Test Includen in SP800-22*, IEICE Trans Fundamentals Vol. E88-A, 2005.

[29] Hamano K. and Kaneko T., *The Correction of the Overlapping Template Matching Test Included in NIST Randomness Test Suite*, IEICE Transaction of Electronics, Communications and Computer Sciences Vol. E90-A, 2007, pp. 1788-1792.

[30] Hoeffding W. and Robbins H., *The Central Limit Theorem for dependent random variables*, Duke Mathematical Journal 15, 1948, pp. 773-780.

[31] Infineon Technlogies, `www.infineon.com`.

[32] Johnson N. J., Kotz S. and Kemp A., *Discrete Distributions*, John Wiley & Sons, New York, 1996, pp. 378-379.

[33] Kemp C. D., *'Stuttering-Poisson' distributions*, Journal of the Statistical and Social Enquiry Society of Ireland Vol. 21, 1967. pp. 151-156.

[34] Khan D., *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*, Macmillan Publishers Ltd, 1967.

[35] Kim J., *How to Choose the Level of Significance: A Pedagogical Note*, MPRA No. 66373, Munich, 2015.

[36] Kim S., Umeno K. and Hasegawa A., *Correction of the NIST Statistical Test Suite for Randomness*, IACR Cryptology ePrint Archive, 2004.

[37] Kimberley M., *Comparison of two statistical tests for keystream sequences*, Electronics Letters. 23, 1987, pp. 365-366.

[38] Knuth D. E., *The Art of Computer Programming. Vol 2: Seminumerical Algorithms*, Addison-Wesley Professional, Harlow, 1998.

[39] Kovalenko I. N., *"Distribution of the linear rank of a random matrix"*, Theory of Probability and its Applications Vol. 17, 1972, pp. 342-346.

[40] Kraft L. G., *A Device for Quantizing Grouping and Coding Amplitude Modulated Pulses*, MS Thesis, MIT, Cambridge,1949.

[41] Kummer E. E., *Ueber die hypergeometrische Reihe*, J. Reine Ang. Math. 35, 1836, pp. 39–83.

[42] Maclaren N., *Cryptographic Pseudo-random Numbers in Simulation*, Cambridge Security Workshop on Fast Software Encryption, Cambridge, 1993, pp. 185-190.

[43] Marsaglia G., *DIEHARD statistical Tests*, `www.stat.fsu.edu/pub/diehard`.

[44] Marsaglia G. and Tsay L. H., *"Matrices and the structure of random number sequences"*, Linear Algebra and its Applications. Vol. 67, 1985, pp. 147-156.

[45] Martin-Löf P. E. R., *The Definition of Random Sequences*, Institute of Mathematical Statistic, Stockholm, 1966, pp. 602-619.

[46] Marton K.,Suchiu A. and Ignat I., *Randomness in Digital Cryptography: A Survey*, Romanian Journal of Information Science and Technology V. 13, 2010, pp.219-240.

[47] MathBits.com, `www.mathbitsnotebook.com`.

[48] MathWorks, *MatLab*, R2013b, `www.mathworks.com`.

[49] MathWorks, *Simulink*, V. 8.2, `www.mathworks.com`.

[50] Maurer U. M., *"A Universal Statistical Test for Random Bit Generators"*, Journal of Cryptology. Vol. 5, 1995, pp. 80-105.

[51] Menezes A. J., van Oorscht P. C., Vanstone S. A., *Handbook of Applied Cryptography*, CRC Press, Boca Raton, 1997.

[52] Microsoft, *Microsoft Visual C++ Compiler*, Microsoft Visual Studio 12.0, *www.visualstudio.microsoft.com.*

[53] Pearson K., *On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling*, Philosophical Magazine. Series 5. 50, 1900, pp. 157-175.

[54] Pincus S. and Huang W. M., *Approximate entropy, statistical properties and applications*, Communications in Statistics, 1992, pp. 3061-3077.

[55] Pincus S. and Singer B. H., *Randomness and degrees of irregularity*, Proc. Natl. Acad. Sci. USA. Vol. 93, 1996, pp. 2083-2088.

[56] Pincus S. and Kalman R. E., *Not all (possibly) "random" sequences are created equal*, Proc. Natl. Acad. Sci. USA. Vol. 94, 1997, pp. 3513-3518.

[57] Pitman J., *Probability*, New York: Springer-Verlag, New York, 1993, pp. 93-108.

[58] Pommerening K., *Lecture Notes of Cryptology curse:* `www.staff.uni-mainz.de/pommeren/Cryptology`.

[59] Renyi A., *Foundation of Probability*, Holden-Day, San Francisco, 1970.

[60] Revesz P., *Random Walk in Random and Non-Random Environments*, World Scientific, Singapore, 1990.

[61] Rivest R., Shamir A. and Adleman L., *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM. 1978.

[62] Rueppel R. A., *Analysis and Design of Stream Ciphers*, Springer, New-York, 1986.

[63] Rukhin A., *Approximate entropy for testing randomness*, Journal of Applied Probability. Vol. 37, 2000.

[64] Rukhin A. et al., *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, NIST SP 800-22 revision 1A, Gaithersburg, 2010.

[65] Shannon C. E., *A Mathematical Theory of Communication*, Bell System Technical Journal 27, 1948.

[66] Shannon C. E., *Communication Theory of Secrecy Systems*, Bell System Technical Journal. 28, pp. 656–715, 1949.

[67] Shum K., *Linear Feedback Shift Register*, IERG 6120 Coding for distributed storage systems, 2016, pp. 64-83.

[68] Spitzer F., *Principles of random walk*, D. Van Nostrand , Princeton, 1964.

[69] Sýs M. et al., *On the interpretation of results from the NIST Statistical Test Suite*, Romanian Journal of Information Science and Technology V. 18, 2015, pp. 18-32.

[70] Vector Informatik GmbH, *CANoe*, V. 10.0, `www.vector.com`.

[71] Ziv J. and Lempel A., *A universal algorithm for sequential data compression*, IEEE Tansactions on Information Theory Vol. 23, 1977.

[72] Ziv J. and Lempel A., *Compression of Individual Sequences via Variable-Rate Coding*, IEEE Tansactions on Information Theory Vol. 24, 1978, pp. 337-343.

[73] Ziv J., *Compression, tests for randomness and estimating the statistical model of an individual sequence*, Capocelli R.M. (eds) Sequences, Springer, New-York, 1990.