Table des matières

Table des figures

Chapitre 1

Introduction Générale

L'aléa ou l'observation du phénomène aléatoire était utilisé dans le cadre de nombreuses applications et cela depuis plusieurs siècles déjà.
En effet autrefois l'aléa était utilisé pour départager les hommes de manière incontestable et c'est pour cette raison qu'il était considéré comme l'élément le plus important dans les jeux de hasard. Il était également utilisé dans les processus de gestions publiques car dans l'antique démocratie athénienne certaines fonctions politiques étaient confiées à des citoyens choisis aléatoirement.
Mais à partir du 20^{iem} siècle on note l'apparition de nouvelles méthodes dites de Monte-Carlo qui à l'aide de processus aléatoire permettaient de calculer la valeur numérique d'une intégrale dont le calcul formel est quasi-impossible.
De nos jours avec les besoins de sécuriser l'information ces techniques ne sont plus utilisées et c'est dans ce contexte que Auguste Kerckhoffs proposa une règle fondamentale de la cryptographie moderne qui va marquer le début des générateurs pseudo-aléatoires.
Actuellement dans un grand nombre de ces applications cryptographiques, il serait très commode mais également souhaitable que le générateur soit aussi simple que possible.
Cependant les postulats de Golomb résument les propriétés de bases exigées sur une telle construction et toute séquence possédant ces propriétés est appelée une séquence pseudo-aléatoire.
En effet les générateurs pseudo-aléatoires doivent passer d'autre tests statistiques comme ceux du NIST dont la base repose sur ces trois postulats.
Dans ce mémoire, nous avons développé l'étude des tests du NIST pour les séquences pseudo-aléatoires. Ce travail ainsi élaboré a été subdivisé en quatre chapitres :
✓ Dans le premier chapitre, nous avons présenté les différents types de générateurs, leurs problématiques mais également proposée des méthodes pour l'évaluation de la qualité et de la performance de ces générateurs .
✓ Dans le chapitre suivant , nous avons choisi de parler des générateurs à bases de registres

filtres dû fait de leur simplicité et de leur rapidité à un temps réel.
✓ Au niveau du troisième chapitre , nous avons proposé pour chaque test un principe , un algorithme et des exemples.
✓ Et enfin dans le dernier chapitre , nous avons proposé une simulation de quelques uns de ces tests.
Cependant les problèmes de performances que peuvent rencontrer ces générateurs sont les suivants :

- Comment mettre en place de tel générateur à travers une implémentation ?
- Comment s'assurer de la qualité et de la performance des générateurs de nombres pseudo-aléatoires ?
- Comment distinguer une sortie d'un générateur pseudo-aléatoire d'une sortie purement aléatoire
- Comment aussi prévoir le terme suivant d'une suite alors que l'on ne connait que les premiers termes seulement ?

Ainsi à travers ces contraintes, l'objectif de notre mémoire sera de proposer des méthodes d'évaluation de la qualité et de la performance des générateurs pseudo-aléatoires et parmi ces générateurs cités ci-dessous :
∗ le générateur de Von Neumann
∗ le générateur de congruence linéaire
∗ le générateur sur les registres filtres (générateur de Geff)
∗ le générateur cryptographiquement sûr(RSA)
nous avons préféré travailler avec celui du générateur de Geff et afin de tester sa qualité des tests statistiques comme ceux du NIST lui feront passer .

Chapitre 2

Rappels sur Les Corps Finis

Introduction

Les corps finis interviennent dans divers domaines des mathématiques et en particulier dans la théorie de Galois sur la résolution des équations algébriques.
Ils sont également utilisés dans la plupart des méthodes de construction des séquences pseudo-aléatoires.
Ces corps sont aussi présents dans d'autres applications telles que les algorithmes de cryptographie , les codes correcteurs d'erreurs , les codes compresseurs etc....
Dans ce chapitre nous allons d'abord commencer par un petit rappel sur les outils algébriques ensuite nous parlerons de l'application des suites récurrentes linéaires sur ces corps et enfin nous terminerons par la conception des générateurs pseudo-aléatoires.

2.1 Etude sur les corps finis

2.1.1 Structures Algébriques

2.1.1.1 Groupes

Définition 2.1.1. Un groupe est un ensemble non vide G muni d'une opération $\circ$:

$$\begin{array}{ccc} G \times G & \longrightarrow & G \\ (x, y) & \longmapsto & x \circ y \end{array}$$

telle que :

- $\circ$ est associative
  $\forall$ x , y , z $\in$ G : $x \circ (y \circ z) = (x \circ y) \circ z$

- il existe un unique élément neutre e ∈ G tel que :
  $x \circ e = e \circ x = x$, ∀ x ∈ G.
- Tout élément x ∈ G a un unique inverse $x^{-1} \in$ G tel que :
  $x \circ x^{-1} = x^{-1} \circ x =$ e

Remarque. De plus le groupe $(G, \circ)$ est dit abélien si l'opération ∘ est commutative :
$x \circ y = y \circ x$, pour tout x , y ∈ G.

2.1.1.2 Anneaux

Définition 2.1.2. Un anneau est un ensemble A muni de deux opérations + , ∘ :

$$A \times A \longrightarrow A$$

telle que :

- $(A, +)$ est un groupe abélien avec comme élément neutre 0 ∈ A et appelé zéro de l'anneau.
- L'opération ∘ est distributive par rapport à l'opération + :

$$\begin{bmatrix} (x+y) \circ z = x \circ z + y \circ z \\ x \circ (y+z) = x \circ y + x \circ z \end{bmatrix} \forall x, y, z \in A.$$

Remarque. Un anneau $(A, +, \circ)$ est dit commutatif si l'opération ∘ est commutative.
Un anneau $(A, +, \circ)$ est dit associatif si l'opération ∘ est associative.
Un anneau $(A, +, \circ)$ est dit unitaire si l'opération ∘ a un élément neutre noté 1 ∈ A et appelé unité de l'anneau.

2.1.1.3 Corps

Un corps est un anneau unitaire $(K, +, \circ)$ tel que si on note $K^* = K \backslash \{0\}$, alors $(K^*, \circ)$ est un groupe.
Si $(K, +, \circ)$ est un corps , un sous-corps de K est un sous-anneau K_1 de K tel que pour tout élément non nul $x \in K_1$ on a $x^{-1} \in K_1$; donc $(K_1, +, \circ)$ est aussi un corps.
On dit qu'un corps K est fini si son cardinal est fini.

2.1.1.4 Corps finis

Définition 2.1.3. Un corps fini est un corps $(K, +, \circ,)$ tel que card(K) = p ∈ $\mathbb{N}$

Proposition 2.1.1. *Un corps fini K contient un corps $\mathbb{F}_p$ où p est un nombre premier.*

Démonstration 2.1.1. Soit K un corps fini, et π l'application :

$$\begin{array}{rccc} \pi & : & \mathbb{Z} & \longrightarrow & K \\ & & n & \longrightarrow & n \cdot 1_K \end{array}$$

$ker\pi$ est un idéal de $\mathbb{Z}$, donc de la forme $p\mathbb{Z}$, avec $p \neq 0$ et $p \neq 1$.
Si $p = p_1 p_2$ non premier , on a $\pi(p) = \pi(p_1)\pi(p_2) = 0$ dans K ,
donc $\pi(p_1) = 0$ ou $\pi(p_2) = 0$,
donc $p_1 \in p\mathbb{Z}$ ou $p_2 \in p\mathbb{Z}$
donc $p_1 = p$ ou $p_2 = p$ car $p_i \leq p$
d'où p est premier.
Et par conséquent $\mathbb{F}_p \subset$ K.

Proposition 2.1.2. *Un corps fini K de caractéristique p admet p^n éléments où n est un entier.*

Démonstration 2.1.2. Soit K un corps fini de caractéristique p
Soit l'application

$$\begin{array}{rcc} \mathbb{F}_p \times K & \longrightarrow & K \\ (a, x) & \longrightarrow & a \cdot x \end{array}$$

une loi de composition externe sur K
K muni de l'addition et de cette loi externe est un $\mathbb{F}_p$-espace vectoriel
Comme K est fini, alors K est un $\mathbb{F}_p$-espace de dimension fini. Autrement dit $dim_{\mathbb{F}_p} K$ = n $< \infty$
De plus $(\mathbb{F}_p)^n \approx \underbrace{\mathbb{F}_p \times \mathbb{F}_p ..... \times \mathbb{F}_p}_{n facteurs}$
Et K est isomorphe à $(\mathbb{F}_p)^n$ Donc $card(K) = card(\mathbb{F}_p)^n = p^n$ car K fini et $(\mathbb{F}_p)^n$ fini.

Remarque. Tout K-ev de dimension n est isomorphe à $(\mathbb{K}_p)^n$

Proposition 2.1.3. *Le groupe multiplicatif K^* de tout corps fini est cyclique.*

Démonstration 2.1.3. Soit K un corps fini tel que $card(K) = q$
et $card(K^*) = q - 1$
Soit a $\in K^*$ tel que $o(a) = d$ divise $q - 1$
• Si d divise $q - 1$ alors on a :
Soit $H_d^* = \{x \in K^* \backslash o(x) = d\}$
H_d= ensemble des racines du polynome $X^d - 1$ et
H_d est un sous groupe de K^*
• • Si $H_d^* \neq \emptyset$ alors soit $a \in H_d^* \subset H_d$
$\Rightarrow H_d$ contient un sous groupe cyclique engendré par a qui a d éléments.
$\Rightarrow H_d$ est isomorphe à $\mathbb{F}_d$

$\Rightarrow card(H_d)$=d et $card(H_d^*) \leq \varphi(d)$
où $\varphi(d)$ = indicatrice d'Euler défini comme étant le nombre d'éléments inversibles dans l'anneau $\mathbb{Z}/n\mathbb{Z}$.
$\Rightarrow q-1 = card(K^*) = card(\bigcup_{d\setminus q-1} H_d^*) = \sum_{d\setminus q-1} card(H_d^*) \leq \sum_{d\setminus q-1} \varphi(d) = q-1$
Donc le nombre d'éléments de K^* d'ordre $d \setminus q-1$ est égal à $\varphi(d)$. Ainsi on note il existe un élément dans K^* d'ordre $q-1$.
D'où cet élément est un générateur de K^* et par conséquent K^* est cyclique.

Remarque. Les générateurs de K^* sont appelés éléments primitifs de K.

2.1.1.5 Définitions

Soit P un polynôme à coefficients dans le corps $\mathbb{F}_p$ et α un élément primitif de ce corps.
• On dit que α est un élément primitif du corps $\mathbb{F}_q$ s'il est le générateur de son groupe multiplicatif $\mathbb{F}_q^*$.
• On dit aussi que P est un polynôme primitif s'il est le polynôme minimal d'un élément primitif de $\mathbb{F}_{p^n}$.

Remarque. Un polynôme primitif est par définition irréductible.

• P est aussi appelé polynôme minimal de l'élément primitif α s'il est unitaire et de plus petit degré annulant α.
• Et enfin on dit que P est un polynôme irréductible, si pour tout $Q \in \mathbb{K}[X]$ et divisant P on a :
$*$ soit $Q \in \mathbb{K}^*$
$*$ soit $\exists\ \lambda \in \mathbb{K}^*$ tel que $Q = \lambda P$
Avec $P \in \mathbb{K}[X]$ et degré de $P \geqslant 1$.

Exemple 2.1.1. Soient $p = 2$, $n = 4$ et $P(X) = X^4 + X^3 + 1$
Le tableau suivant nous permet de mieux comprendre la définition du polynôme primitif donnée ci-dessus

i	0	1	2	3	4	5
α^i	1	α	α^2	α^3	α^3+1	$\alpha^3+\alpha+1$
i	6	7	8	9	10	11
α^i	$\alpha^3+\alpha^2+\alpha+1$	$\alpha^2+\alpha+1$	$\alpha^3+\alpha^2+\alpha$	α^2+1	$\alpha^3+\alpha$	$\alpha^3+\alpha^2+1$
i	12	13	14	15	16	17
α^i	$\alpha+1$	$\alpha^2+\alpha$	$\alpha^3+\alpha^2$	1	α	α^2

$\Rightarrow \alpha$ est un élément primitif et $P(X) = X^4 + X^3 + 1$ un polynôme primitif dans $\mathbb{F}_2[X]$.

Proposition 2.1.4. *Soit p un nombre premier et n un entier strictement positif.*
Alors il existe un corps à p^n éléments. Et ce corps est unique.

Démonstration 2.1.4. La démonstration sera divisée en deux partie :

• Existence

Considérons le polynôme $f(x) = x^{p^n} - x$

Soit L un corps fini avec $\mathbb{F}_p \in L$. Le polynôme $f(x)$ se décompose en facteur de degré 1 sur L.

Ces facteurs sont au nombres de $p^n = degf$ et sont distincts car $f'(x) = p^n x^{p^n-1} - 1 = -1$ n'a pas de racine.

Soit K l'ensemble des racines du polynôme $f(x) = x^{p^n} - x$. K est un sous corps de L car stable par les lois de L. En effet

⋆ $\forall a, b \in K, (a+b)^{p^n} = a^{p^n} + b^{p^n} = a + b$

⋆ $\forall a, b \in K, (ab)^{p^n} = ab$

⋆ l'inverse de $a \neq 0 \in K^*$ est a^{p^n-2} car $a(a^{p^n-2}) = a^{p^n-1} = 1$

Donc K contient p^n éléments et a est une racine du polynôme f.

D'où K est un corps à p^n éléments.

• Unicité

Soient K et L deux corps ayant $p^n = q$ éléments.Montrons que K est isomorphe à L.

⋆ D'abord montrons que pour tout corps K à q éléments on a :

$$x^q - x = \prod_{a \in K} (x - a)$$

Le groupe K^* a $q-1$ éléments et pour tout $a \in K^*$, $a^{q-1} = 1$

Donc pour tout élément $a \in K$, $a(a^{q-1} - 1) = a^q - a = 0$

Donc $(x-a)$ divise $x^q - x$

D'où $\prod_{a \in K}(x-a)$ divise $x^q - x$.

Puisque ces deux polynômes ont même degré, alors il y'a égalité.

⋆ Maintenant montrons que K et L sont isomorphes.

Soit a un élément primitif de K et $f(x)$ son polynôme minimal

On a donc :

$$K \approx \mathbb{F}_p[X] / (f(x))$$

Puisque $a \in K$ et a est aussi racine de $x^q - x$, alors $f(x)$ divise $x^q - x = \prod_{a \in K}(x-a)$

Donc il va exister un élément $b \in L$ racine de $f(x)$

Donc le polynôme minimal de b noté $g(x)$ est un diviseur de $f(x)$

On en déduit donc que :

$$f(x) = g(x)$$

car $f(x)$ irréductible et unitaire.

Conclusion :

$$K \approx \mathbb{F}_p[X]/(f(x)) = \mathbb{F}_p[X]/(g(x))$$

Finalement on a un homomorphisme de corps :

$K \approx \mathbb{F}_p[X]/(f(x)) = \mathbb{F}_p[X]/(g(x)) \to L$, injectif mais aussi surjectif car $\sharp(K) = \sharp(L)$. D'où K et L isomorphes.

2.2 Applications des Suites Récurrentes Linéaires

Définition 2.2.1. Une suite $(U_n)_n$ d'ordre k est dite récurrente linéaire dans $\mathbb{F}_q$ s'il existe $a_0, a_1, ..., a_{k-1}$, b $\in \mathbb{F}_q$ tels que pour tout entier n , on a :

$$U_{n+k} = a_{k-1}U_n + ... + a_0U_{n+k-1} + b, \forall n \geq 0$$

Exemple 2.2.1. Soit $(U_n)_n$ une suite récurrente linéaire dans $\mathbb{F}_2$ d'ordre 4 et 2 définie par :

$$U_{n+4} = U_{n+1} + U_n + b$$

ou

$$U_{n+2} = U_{n+1} + U_n + b$$

avec $b \in \mathbb{F}_2$.

Remarque. Pour la définition ci-dessus on constate que :

⋆ si $b = 0$ la relation est dite homogène et dans le cas contraire elle est dite non homogène.

⋆ $U_0, U_1, ..., U_{k-1}$ détermine l'état initial de la suite.

2.2.1 Polynôme Caractéristique et Réciproque

2.2.1.1 Polynôme Caractéristique

On appelle polynôme caractéristique d'une suite récurrente linéaire $(U_n)_n$, tout polynôme f s'écrivant sous cette forme :

$$f(x) = x^k - a_0x^{k-1} - ..... - a_{k-1} \in \mathbb{F}_q[X]$$

et $(a_0, ..., a_{k-1})$ sont les coefficients de la suite.

2.2.1.2 Réciproque du Polynôme Caractéristique

g est appelé réciproque du polynôme caractéristique **si et seulement si** g est de la forme :

$$g(x) = x^k f(\frac{1}{x})$$

2.2.2 Ordre d'un polynôme et Période d'une suite

2.2.2.1 Ordre d'un polynôme

On appelle ordre du polynôme f d'une suite récurrente linéaire $(U_n)_n$, le plus petit entier o tel que :

$$x^o \equiv 1(f(x))$$

2.2.2.2 Période d'une suite

On appelle période d'une suite récurrente linéaire $(U_n)_n$, le plus petit entier positif p tel que pour tout $n \geq 0$:

$$U_{n+p} = U_n$$

Nous avons finalement $per(U) = 15 = Ord(f)$

2.2.3 Exemple et Remarque

2.2.3.1 Exemple

Dans cette partie nous avions proposé un exemple expliquant les deux sous-sections élaborées tout juste au haut.
En effet soit $(U_n)_n$ une suite récurrente linéaire dans $\mathbb{F}_2$ d'ordre 4 et 2 définie par :

$$U_{n+4} = U_{n+1} + U_n + b$$

ou

$$U_{n+2} = U_{n+1} + U_n + b$$

avec $b \in \mathbb{F}_2$.
Leurs polynômes caractéristiques sont respectivement :

$$f_1(x) = x^4 + x + 1$$

$$f_2(x) = x^2 + x + 1$$

Et leurs polynômes réciproques sont :

$$g_1(x) = x^4 f_1(\frac{1}{x}) = x^4(\frac{1}{x^4} - \frac{1}{x} - 1) = 1 - x^3 - x^4 = 1 + x^3 + x^4$$

$$g_2(x) = x^2 f_2(\frac{1}{x}) = x^2(\frac{1}{x^2} - \frac{1}{x} - 1) = 1 - x - x^2 = 1 + x + x^2$$

Cependant pour finir l'exemple nous avions donné tout juste après l'ordre des polynômes et la période des suites :
Pour les polynômes caractéristiques

$$f_1(x) = x^4 + x + 1$$

$$f_2(x) = x^2 + x + 1$$

On a respectivement

$$Ord(f_1) = 15$$

$$Ord(f_2) = 3$$

Maintenant on reconsidère la suite : $U_{n+4} = U_{n+1} + U_n$
avec $U_0 = 1$,$U_1 = 0$, $U_2 = 0$, $U_3 = 1$
On a donc $per(U) = 15$
En effet d'après ce tableau

U_0	U_1	U_2	U_3	U_4	U_5	U_6	U_7	U_8
1	0	0	1	1	0	1	0	1
U_9	U_{10}	U_{11}	U_{12}	U_{13}	U_{14}	U_{15}	U_{16}	U_{17}
1	1	1	0	0	0	1	0	0

nous avions $per(U) = 15 = Ord(f)$
De même pour l'autre suite.

2.2.3.2 Conséquence

A partir de l'exemple donné ci-dessus nous pouvons donc conclure que la période de la séquence est équivalente à l'ordre de son polynôme caractéristique.

2.2.4 Matrice de Compagnons

On appelle matrice de compagnons de la suite récurrente linéaire $(U_n)_n$, la matrice M définie par :

$$\begin{bmatrix} a_{k-1} & a_{k-2} & . & ... & a_0 \\ 0 & 0 & 0 & ... & 1 \\ 0 & . & ... & 1 & 0 \\ . & . & . & ... & . \\ . & . & . & ... & . \\ 0 & 1 & . & ... & 0 \end{bmatrix}$$

Exemple 2.2.2. Considérons la suite $U_{n+4} = U_{n+1} + U_n$, avec $a_0 = a_1 = 0$ et $a_3 = a_2 = 1$
La matrice de compagnons M est égale à :

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Après vérification on a :

$$\begin{bmatrix} U_4 \\ U_3 \\ U_2 \\ U_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} U_0 \\ U_1 \\ U_2 \\ U_3 \end{bmatrix}$$

Ainsi à partir de la vérification nous pouvons en tirer une expression générale :

$$\begin{bmatrix} U_{n+k} \\ U_{n+k-1} \\ . \\ . \\ . \\ U_{n+1} \end{bmatrix} = \begin{bmatrix} a_{k-1} & a_{k-2} & . & ... & a_0 \\ 0 & 0 & 0 & ... & 1 \\ 0 & . & ... & 1 & 0 \\ . & . & . & ... & . \\ . & . & . & ... & . \\ 0 & 1 & . & ... & 0 \end{bmatrix} \begin{bmatrix} U_n \\ U_{n+1} \\ . \\ . \\ . \\ U_{n+k-1} \end{bmatrix}$$

2.2.4.1 Polynôme Minimal d'une suite récurrente linéaire

On appelle polynôme minimal d'une suite récurrente linéaire, tout polynôme génératrice primitif à coefficients dans $\mathbb{F}_q$

Exemple 2.2.3. Reconsidérons la suite récurrente linéaire $U_{n+4} = U_{n+1} + U_n$

polynôme minimal sur $\mathbb{F}_2[X]$ est égale à :

$$f(x) = x^4 + x + 1$$

Remarque. Dans $\mathbb{F}_q[X]$, la période d'une suite récurrente linéaire de polynôme minimal f est égale à $q^k - 1$ avec f **irréductible**.
De telles suites sont dites des m-séquences.

2.2.5 Représentation en Circuit Electronique

Pour les suites récurrentes linéaires, on note l'existence de deux types de représentations :
$*$ Cas de Fibonocci
$*$ Cas de Galois

2.2.5.1 Cas de Fibonocci

Soit $(F_n)_n$ une suite récurrente linéaire satisfaisant à la relation de récurrence linéaire suivante :

$$F_{n+k} = a_{k-1}F_n + ... + a_0F_{n+k-1}$$

Sa table de vérité est donnée par :

Table de Vérité

C'est un tableau qui permet de caractériser le fonctionnement du circuit. En effet
Considérons la suite $U_{n+3} = U_{n+1} + U_n$, avec $U_0 = 0$,$U_1 = 1$ et $U_2 = 0$
On a donc :

$C \setminus K$	$U_{n+1} + U_n$	U_{n+2}	U_{n+1}	U_n	*Sortie*
0	1	$\underline{0}$	$\underline{1}$	$\underline{0}$	0
1	1	1	0	1	1
2	1	1	1	0	0
3	0	1	1	1	1
4	0	0	1	1	1
5	1	0	0	1	1
6	0	1	0	0	0
7	1	$\underline{0}$	$\underline{1}$	$\underline{0}$	0

Ainsi la séquence de sortie devient :0101110
Et les états initiaux non nuls sont :

010	101	110	111	011	001	100

2.2.5.2 Cas de Galois

On reconsidère la même suite récurrente linéaire $(F_n)_n$.
Pour ce cas la table de vérité devient :

Table de Vérité

$C \setminus K$	$U_{n+1} + U_n$	U_{n+2}	U_{n+1}	U_n	*Sortie*
0	1	$\underline{0}$	$\underline{1}$	$\underline{0}$	1
1	1	1	0	1	0
2	1	1	1	0	1
3	0	1	1	1	1
4	0	0	1	1	1
5	1	0	0	1	0
6	0	1	0	0	0
7	1	$\underline{0}$	$\underline{1}$	$\underline{0}$	1

La séquence de sortie devient :1011100
Et les états initiaux non nuls sont :

010	101	110	111	011	001	100

Ainsi nous pouvons conclure que la table de vérité est un tableau permettant de déterminer les différentes séquences et états initiaux de la suite.
★ Pour cette première partie nous avons brièvement rappelé quelques définitions et propositions sur les corps finis mais aussi l'application des suites récurrentes linéaires sur ces corps. Ces notions nous seront très utiles pour la génération et la représentation des séquences pseudo-aléatoires mais également pour la compréhension de la conception des générateurs aléatoires.

2.3 Conception de générateurs aléatoires

Un générateur de nombre aléatoire ou RNG(Random Number Generator) est un dispositif capable de produire une séquence dont on ne peut pas prédire la sortie.On dit qu'un tel générateur est idéal s'il est une construction mathématique générant des nombres aléatoires indépendants et uniformément répartis.

Il existe depuis longtemps des méthodes pour obtenir de tels nombres mais néanmoins de nombreux progrès ont été réalisés.
On distingue ainsi deux types de générateurs de nombres aléatoires avec une possibilité d'hybridation entre eux :

- les générateurs de nombres réellement aléatoires ou TRNG(Random True Number Generator) et
- les générateurs de nombres pseudo-aléatoires ou PRNG(Pseudo Random Number Generator)

Cependant leur critère de classification coincide avec celui des sources d'aléa (d'entropie).

2.3.1 Définition source d'entropie

Une source d'entropie est une source constituée généralement de certaines quantités physiques ou non.
Ainsi on note l'existence de deux types de sources d'entropie.

2.3.1.1 Source d'entropie non physique

Une source d'entropie non physique est une source qui essaie d'imiter le comportement des sources naturelles , mais avec moins de robustesse. Elle provient également des algorithmes mathématiques et est de nature séquentielle.

Exemple 2.3.1. le mouvement d'une souris , les jeux de dés...

2.3.1.2 Source d'entropie physique

Une source d'entropie physique est une source due aux caractéristiques intrinsèques des systèmes physiques.

Exemple 2.3.2. le vidéo projecteur , le bruit thermique...

$*$ Ainsi nous pouvons dire que le niveau de non-prédictibilité sur les sources d'entropie physiques est trés élevé et ne dépend que de la source d'aléa utilisée sur les générateurs aléatoires. En effet pour choisir un générateur de nombres aléatoires on se base principalement sur deux caractéristiques à savoir son débit de bits aléatoires mais également la qualité de l'aléa produit. Ce qui nous permettra tout juste après d'introduire la structure des générateurs aléatoires.

2.3.2 Structure des générateurs de nombres aléatoires

2.3.2.1 Générateurs de nombres réellement aléatoires (TRNG)

Définition 2.3.1. Un générateur de nombre réellement aléatoire est un composant générant des valeurs issues d'une source dont le niveau de non-prédictibilité est très élevé.

Exemple 2.3.3. Des photons envoyés contre un miroir semi-transparent , l'horloge système d'un ordinateur , le mouvement d'une souris , les jeux de hasard ,le vidéo projecteur ....

Propriété 2.3.1. Les propriétés fondamentales qu'un TRNG doit satisfaire sont les suivantes :

- posséder une distribution statistique uniforme.
- mais également contenir une source d'entropie responsable de la caractéristique aléatoire.

Et suivant la source utilisée on distingue deux types de générateurs de nombres réellement aléatoires :

TRNG non Physique

Ce sont des générateurs dont la source d'aléa n'est pas un phénoméne physique. Mais aussi ce sont des composants stockant d'autres information du systéme.

TRNG Physique

Contrairement au TRNG non physique les TRNG dits physiques sont des dispositifs dédiés uniquement à la génération de nombres aléatoires.
L'architecture suivante nous permet de mieux illustrer ces propos

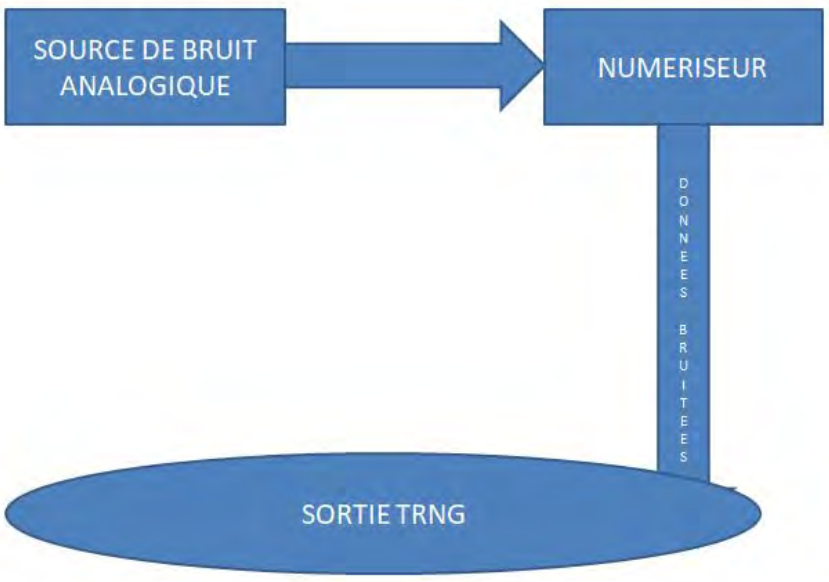


FIGURE 2.1 – Génération de nombres réellement aléatoires

Pour les générateurs aléatoires si le signal de bruit est en défaillance, alors se sera le rôle du numériseur de s'assurer que la sortie ne soit pas perturber par cette faille.
Cependant si le numériseur n'est pas capable de rééquilibrer cette disparité,alors un correcteur d'entropie également appelé post-traitement sera ajouté à la sortie.

Post-traitement

Un post-traitement est un traitement effectué sur les données avant leur utilisation.
Ils existent deux types de correcteurs d'entropie :

- les post-traitements arithmétiques : la méthode de Neumann ,le Correcteur XOR , les LFSR
- les post-traitements cryptographiques :la fonction de hachage , ceux basés sur le chiffrement par bloc , et ceux sur la théorie des nombres.

En dessous on note l'architecture des TRNG après correction

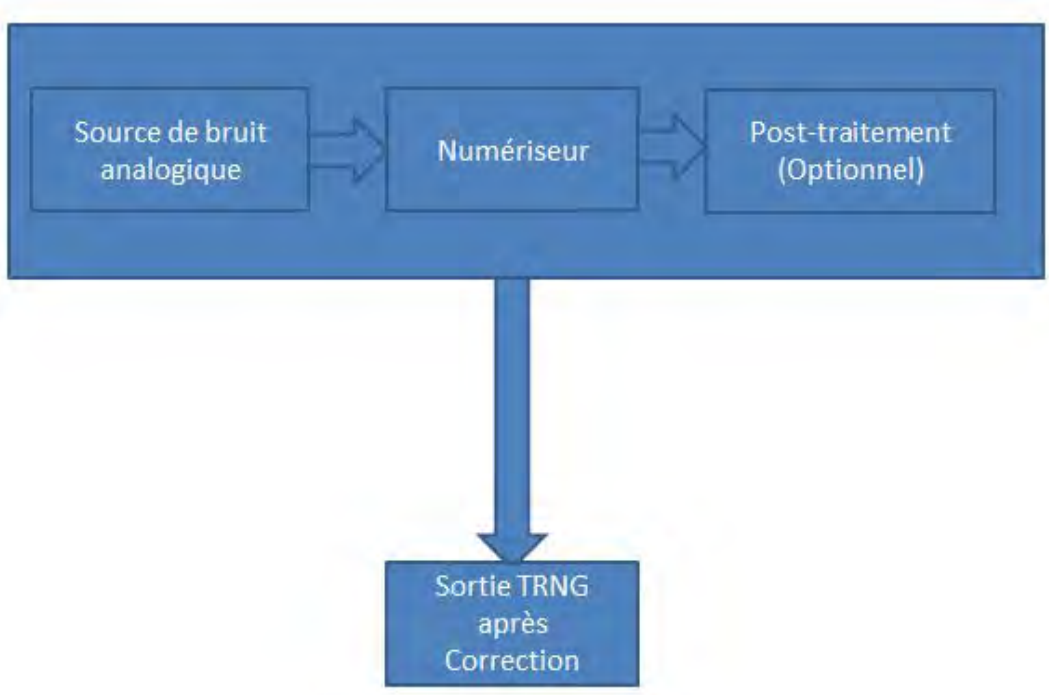


FIGURE 2.2 – Génération de nombres réellement aléatoires après correction

2.3.2.2 Générateurs de nombres aléatoires déterministes (DRNG)

Définition 2.3.2. Un générateur de nombres aléatoires déterministes est un générateur qui produit des nombres qui ont de très bonnes propriétés statistiques mais théoriquement prévisibles
.

Propriété 2.3.2. Les propriétés fondamentales qu'un DRNG doit satisfaire sont les suivantes :

- une bonne propriété statistique et d'uniformité : on veut obtenir une séquence de nombres qui passe avec succés la plupart des tests statistiques mais à un temps raisonnable
- une longue période : la période des valeurs produites doit être beaucoup plus grande que celle des valeurs aléatoires pour une simulation
- l'efficacité : le temps de calcul pour produire les valeurs doit être le plus petit que possible que le temps de simulation
- la répétabilité : L'utilisateur doit être capable de reproduire la même séquence de nombres facilement ce qui est important pour la vérification de programmes
- la facilité d'implémentation et la séparabilité : on doit pouvoir exécuter le générateur sur tous les types d'ordinateurs standards ; mais également ce générateur doit être capable de quitter une valeur (U_n) vers une autre valeur (U_{n+k}) à un temps suffisamment grand.

Ainsi on note l'existence de deux types de DRNG :

⋆ les **PRNG** également appelé générateur de nombres pseudo-aléatoire

⋆ et les **PRNG probabilistes**

2.3.2.3 Générateurs de nombres pseudo-aléatoires

Définition 2.3.3. Un générateur pseudo-aléatoire est un procédé qui à partir d'une initialisation de taille fixée appelée graine ou germe engendre de manière déterministe une suite de très grande longueur de séquences binaires dite aléatoire.

Méthodologie de génération de nombres pseudo-aléatoires

On définie un générateur de nombres pseudo-aléatoires(PRNG) comme étant une fonction :

$$\begin{array}{rcccc} f & : & U \subset \{0,1\}^k & \longrightarrow & \{0,1\}^l \\ & & X_0 & \longrightarrow & f(X_0) = (x_1, x_2, ..., x_l) \end{array}$$

où k et l sont fixés ,$k < l$ et X_0 un germe(tenu secret)
f est construite grâce à un calcul récurrent qui permet de calculer successivement les bits x_i de $f(X_0)$.
En effet soit deux fonctions convenablement construites

- $$\begin{array}{rcccc} u & : & \{0,1\}^k & \longrightarrow & \{0,1\}^k \\ & & X_{n-1} & \longrightarrow & u(X_{n-1}) = X_n \end{array}$$

permettant de calculer par récurrence un état interne X_n à partir de la valeur initiale X_0.
Et

- $$\begin{array}{rcccc} v & : & \{0,1\}^k & \longrightarrow & \{0,1\}^l \\ & & X_n & \longrightarrow & v(X_n) = x_n \end{array}$$

  permettant d'obtenir les bits successifs $(x_1, x_2, ..., x_l)$

Remarque. $*$ Si les fonctions u et v sont convenablement construites, alors l'attaquant ne pourra pas calculer facilement le bit x_{t+1} même s'il ne connait que les t premiers bits (mais pas le germe).
$*$ Et pour le reste de notre mémoire on ne travaillera qu'avec les générateurs pseudo-aléatoires et l'ensemble des tests traités seront implémentés sur Python.

2.3.2.4 Exemples de générateurs de nombres pseudo-aléatoires(PRNG)

Générateurs basés sur les registres à décalage à rétroaction linéaire (LFSR)

Un LFSR est un générateur qui permet de décrire l'évolution du système, sa périodicité et sa complexité.
Son principe repose sur des mécanismes d'algébres linéaires.
Cependant, il faut être prudent avec leur utilisation car leur comportement complexe peut masquer une faible entropie en entrée et ceci est dangereux car ces structures sont très prévisibles.

Principe 2.3.1. Le principe de base d'un registre de n bits à base de fonction de rétroaction linéaire(XOR) initialisée avec $(b_1, b_2, b_3, ..., b_{n-1})$ est le suivant :
• le 1^{ier} top d'horloge est rempli par le résultat d'une fonction linéaire qui prend en compte l'état d'un ou de plusieurs étages.
•arriver au $(n-1)^{ier}$ top d'horloge l'état du registre est identique à son état initial.
On dit alors que le LFSR est de période $(n-1)$.

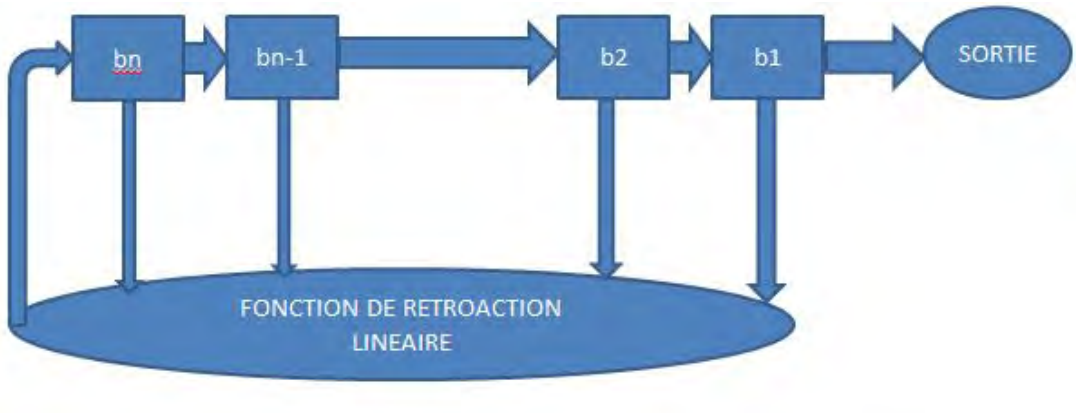


FIGURE 2.3 – Principe du LFSR

Générateurs basés sur la méthode de Neumann

Définition 2.3.4. C'est un générateur pseudo-aléatoire connu sous le nom de la méthode middle-square.Il est très simple et consiste à prendre un nombre à l'élever au carré et à prendre les chiffres au milieu comme sortie.
Autrement dit on élimine les $\frac{n^2}{4}$ nombres à gauche et les $\frac{n^2}{4}$ nombres à droite de la séquence et on prend le milieu.

Principe 2.3.2. Prenons la graine $X_0 = 1367$
ensuite on l'éléve au carré $X_0^2 = 1868689$
on élimine les $\frac{n^2}{4}$ nombres à gauche et les $\frac{n^2}{4}$ nombres à droite de la séquence et on prend le milieu $X_1 = 6868$
ensuite on l'éléve au carré $X_1^2 = 47169424$
on prend le milieu $X_2 = 1694$
ensuite on l'éléve au carré $X_2^2 = 2869636$
on élimine les $\frac{n^2}{4}$ nombres à gauche et les $\frac{n^2}{4}$ nombres à droite de la séquence et on prend le milieu $X_3 = 6963$
..
..
..
et ainsi de suite...

Remarque. si n^2 est un nombre impaire, alors on éliminera les $\lfloor \frac{n^2}{4} \rfloor + 1$ nombres à gauche et les $\lfloor \frac{n^2}{4} \rfloor$ nombres à droite de la séquence et on prend le milieu.

Pour mieux illustrer ces propos nous avons proposé tout juste après le résultat du test de l'implémentation proposé :

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:10) [MSC v.1900 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/user-pc/Desktop/Memoire Kancou/Kancou Implementation/Generate
ur de Neumann.py
>>> Neumann(123986,100)
([725281, 325289, 129335, 275422, 572780, 769284, 978726, 45830, 3889, 151243, 7
44450, 58025, 669006, 690280, 864784, 513666, 527595, 564840, 442256, 903695, 64
6530, 10409, 83472, 675747, 340080, 544064, 56360, 764496, 541340, 489956, 56881
9, 550547, 19992, 996800, 102400, 857600, 777600, 617600, 297600, 657600, 377600
, 817600, 697600, 457600, 977600, 17600, 97600, 257600, 577600, 217600, 497600,
57600, 177600, 417600, 897600, 857600, 777600, 617600, 297600, 657600, 377600, 8
17600, 697600, 457600, 977600, 17600, 97600, 257600, 577600, 217600, 497600, 576
00, 177600, 417600, 897600, 857600, 777600, 617600, 297600, 657600, 377600, 8176
00, 697600, 457600, 977600, 17600, 97600, 257600, 577600, 217600, 497600, 57600,
 177600, 417600, 897600, 857600, 777600, 617600, 297600, 657600, 377600, 817600]
, [252819], [1537252819])
>>> Neumann(2244,100)
([355, 1260, 5876, 5273, 8045, 7220, 1284, 6486, 681, 4637, 5017, 1702, 8968, 42
50, 625, 3906, 2568, 5946, 3549, 5954, 4501, 2590, 7081, 1405, 9740, 8676, 2729,
 4474, 166, 275, 756, 5715, 6612, 7185, 6242, 9625, 6406, 368, 1354, 8333, 4388,
 2545, 4770, 7529, 6858, 321, 1030, 609, 3708, 7492, 1300, 6900, 6100, 2100, 410
0, 8100, 6100, 2100, 4100, 8100, 6100, 2100, 4100, 8100, 6100, 2100, 4100, 8100,
 6100, 2100, 4100, 8100, 6100, 2100, 4100, 8100, 6100, 2100, 4100, 8100, 6100, 2
100, 4100, 8100, 6100, 2100, 4100, 8100, 6100, 2100, 4100, 8100, 6100, 2100, 410
0, 8100, 6100, 2100, 4100, 8100, 6100, 2100], [3553], [503553])
>>>
```

Générateurs basés sur la méthode de congruence linéaire

Définition 2.3.5. C'est un générateur dont la séquence de nombres aléatoires X_n est créée de la maniére suivante :

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

Cependant il nécessite peu d'opération et la production des nombres est rapide.

On a

a le multiplicateur,

b l'incrément

X_0 la graine

et De plus $a \geq 0, b < m$

Principe 2.3.3. Le schéma ci-dessous illustre le principe du générateur de congruence linéaire

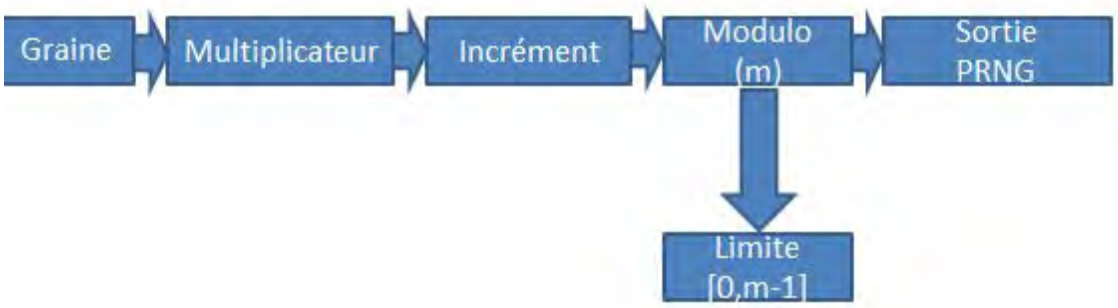


FIGURE 2.4 – Principe du GCL

Pour mieux illustrer cela nous avons proposé le résultat du test de l'implémentation :

```
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/user-pc/Desktop/Memoire Kancou/Kancou Implementation/Générateur_de_congruentiel_linéaire.py
>>> GCL(97,23,3,10)
[186, 185, 162, 657, 778, 489, 1010, 705, 858, 281, 322, 241, 426, 585, 146, 289, 506, 377, 482, 849, 74, 681, 306, 897, 154, 473, 642, 433, 746, 77
7, 466, 481, 826, 569, 802, 17, 394, 873, 626, 65, 474, 665, 962, 625, 42, 969, 786, 673, 122, 761, 98, 209, 714, 41, 946, 257, 794, 857, 258, 817,
362, 137, 82, 865, 442, 953, 418, 401, 10, 233, 242, 449, 90, 25, 578, 1009, 682, 329, 402, 33, 762, 121, 738, 593, 330, 425, 562, 641, 410, 217, 89
8, 177, 1002, 521, 722, 225, 58, 313, 34, 785, 650, 617, 882, 833, 730, 409, 194, 369, 298, 713, 18, 417, 378, 505, 354, 977, 970, 809, 178, 1, 26,
601, 514, 561, 618, 905, 338, 609, 698, 697, 674, 145, 266, 1001, 498, 193, 346, 793, 834, 753, 938, 73, 658, 801, 1018, 889, 994, 337, 586, 169, 81
8, 385, 666, 985, 130, 945, 234, 265, 978, 993, 314, 57, 290, 529, 906, 361, 114, 577, 986, 153, 450, 113, 554, 457, 274, 161, 634, 249, 610, 721, 2
02, 553, 434, 769, 282, 345, 770, 305, 874, 649, 594, 353, 954, 441, 930, 913, 522, 745, 754, 961, 602, 537, 66, 497, 170, 841, 914, 545, 250, 633,
226, 81, 842, 937, 50, 129, 922, 729, 386, 689, 490, 9, 210, 737, 570, 825, 546, 273, 138, 105, 370, 321, 218, 921, 706, 881, 810, 201, 530, 929, 89
0, 1017, 866, 465, 458, 297, 690, 513, 538, 89, 2, 49, 106, 393, 850, 97, 186, 185, 162, 657, 778, 489, 1010, 705, 858, 281, 322, 241, 426, 585, 146
, 289, 506, 377, 482, 849, 74, 681, 306, 897, 154, 473, 642, 433, 746, 777, 466, 481, 826, 569, 802, 17, 394, 873, 626, 65, 474, 665, 962, 625, 42,
969, 786, 673, 122, 761, 98, 209, 714, 41, 946, 257, 794, 857, 258, 817, 362, 137, 82, 865, 442, 953, 418, 401, 10, 233, 242, 449, 90, 25, 578, 1009
, 682, 329, 402, 33, 762, 121, 738, 593, 330, 425, 562, 641, 410, 217, 898, 177, 1002, 521, 722, 225, 58, 313, 34, 785, 650, 617, 882, 833, 730, 409
, 194, 369, 298, 713, 18, 417, 378, 505, 354, 977, 970, 809, 178, 1, 26, 601, 514, 561, 618, 905, 338, 609, 698, 697, 674, 145, 266, 1001, 498, 193,
346, 793, 834, 753, 938, 73, 658, 801, 1018, 889, 994, 337, 586, 169, 818, 385, 666, 985, 130, 945, 234, 265, 978, 993, 314, 57, 290, 529, 906, 361,
114, 577, 986, 153, 450, 113, 554, 457, 274, 161, 634, 249, 610, 721, 202, 553, 434, 769, 282, 345, 770, 305, 874, 649, 594, 353, 954, 441, 930, 913
, 522, 745, 754, 961, 602, 537, 66, 497, 170, 841, 914, 545, 250, 633, 226, 81, 842, 937, 50, 129, 922, 729, 386, 689, 490, 9, 210, 737, 570, 825, 5
46, 273, 138, 105, 370, 321, 218, 921, 706, 881, 810, 201, 530, 929, 890, 1017, 866, 465, 458, 297, 690, 513, 538, 89, 2, 49, 106, 393, 850, 97, 186
, 185, 162, 657, 778, 489, 1010, 705, 858, 281, 322, 241, 426, 585, 146, 289, 506, 377, 482, 849, 74, 681, 306, 897, 154, 473, 642, 433, 746, 777, 4
66, 481, 826, 569, 802, 17, 394, 873, 626, 65, 474, 665, 962, 625, 42, 969, 786, 673, 122, 761, 98, 209, 714, 41, 946, 257, 794, 857, 258, 817, 362,
137, 82, 865, 442, 953, 418, 401, 10, 233, 242, 449, 90, 25, 578, 1009, 682, 329, 402, 33, 762, 121, 738, 593, 330, 425, 562, 641, 410, 217, 898, 17
7, 1002, 521, 722, 225, 58, 313, 34, 785, 650, 617, 882, 833, 730, 409, 194, 369, 298, 713, 18, 417, 378, 505, 354, 977, 970, 809, 178, 1, 26, 601,
514, 561, 618, 905, 338, 609, 698, 697, 674, 145, 266, 1001, 498, 193, 346, 793, 834, 753, 938, 73, 658, 801, 1018, 889, 994, 337, 586, 169, 818, 38
5, 666, 985, 130, 945, 234, 265, 978, 993, 314, 57, 290, 529, 906, 361, 114, 577, 986, 153, 450, 113, 554, 457, 274, 161, 634, 249, 610, 721, 202, 5
53, 434, 769, 282, 345, 770, 305, 874, 649, 594, 353, 954, 441, 930, 913, 522, 745, 754, 961, 602, 537, 66, 497, 170, 841, 914, 545, 250, 633, 226,
81, 842, 937, 50, 129, 922, 729, 386, 689, 490, 9, 210, 737, 570, 825, 546, 273, 138, 105, 370, 321, 218, 921, 706, 881, 810, 201, 530, 929, 890, 10
17, 866, 465, 458, 297, 690, 513, 538, 89, 2, 49, 106, 393, 850, 97, 186, 185, 162, 657, 778, 489, 1010, 705, 858, 281, 322, 241, 426, 585, 146, 289
, 506, 377, 482, 849, 74, 681, 306, 897, 154, 473, 642, 433, 746, 777, 466, 481, 826, 569, 802, 17, 394, 873, 626, 65, 474, 665, 962, 625, 42, 969,
786, 673, 122, 761, 98, 209, 714, 41, 946, 257, 794, 857, 258, 817, 362, 137, 82, 865, 442, 953, 418, 401, 10, 233, 242, 449, 90, 25, 578, 1009, 682
, 329, 402, 33, 762, 121, 738, 593, 330, 425, 562, 641, 410, 217, 898, 177, 1002, 521, 722, 225, 58, 313, 34, 785, 650, 617, 882, 833, 730, 409, 194
, 369, 298, 713, 18, 417, 378, 505, 354, 977, 970, 809, 178, 1, 26, 601, 514, 561, 618, 905, 338, 609, 698, 697, 674, 145, 266, 1001, 498, 193, 346,
793, 834, 753, 938, 73, 658, 801, 1018, 889, 994, 337, 586, 169, 818, 385, 666, 985, 130, 945, 234, 265, 978, 993, 314, 57, 290, 529, 906, 361, 114,
577, 986, 153, 450, 113, 554, 457, 274, 161, 634, 249, 610, 721, 202, 553, 434, 769, 282, 345, 770, 305, 874, 649, 594, 353, 954, 441, 930, 913, 522
, 745, 754, 961, 602, 537, 66, 497, 170, 841, 914, 545, 250, 633, 226, 81, 842, 937, 50, 129, 922, 729, 386, 689, 490, 9, 210, 737, 570, 825, 546, 2
73, 138, 105, 370, 321, 218, 921, 706, 881, 810, 201, 530, 929, 890, 1017, 866, 465, 458, 297, 690, 513, 538, 89, 2, 49, 106, 393, 850, 97]
>>>
```

2.3.2.5 Générateurs de nombres pseudo-aléatoires probabilistes

Définition 2.3.6. Un générateur de nombres pseudo-aléatoires probabilistes est un algorithme déterministe qui produit un résultat à partir d'une chaine aléatoire $r \in \{0,1\}^*$ dans l'ensemble des valeurs possibles.
Cet algorithme se comporte différemment lorsqu'il est appelé deux fois avec les mêmes paramétres.

Exemples de PRNG probabilistes

★ La méthode de quasi Monte-Carlo

Soit $F = (X_i = (u_0, u_1, ..., u_{t-1}) : X_0 \in \{0,1\}^k)$ l'ensemble de tous les points à t dimensions produites par les valeurs successives d'un générateur à partir de tous les états initiaux possibles X_0. Mais avec $k \geqq tl$ et de plus $\sharp(F) = 2^k$.
Donc

$$M = \frac{1}{2^k} * \sum_{h=1}^{2^k} f(X_i)$$

qui est la somme des moyennes de f sur tous les points de F.
Cependant la probabilité d'obtenir une réponse correcte augmente avec le temps disponible.

2.3.2.6 Preuve d'aléa des DRNG

Principe 2.3.4. Reconsidérons la fonction f utilisée pour la méthode de génération de nombres pseudo-aléatoires. Soit

$$\begin{array}{rcccl} f & : & U \subset \{0,1\}^k & \longrightarrow & \{0,1\}^l \\ & & X_0 & \longrightarrow & f(X_0) = (x_1, x_2, ..., x_l) \end{array}$$

où k et l sont fixés ,$k < l$ et X_0 un germe(tenu secret)

- Après avoir calculé $(x_1, x_2, ..., x_l) = f(X_0)$ on conserve les k premiers bits avec $(y_1 = x_1, y_2 = x_2, ..., y_k = x_k)$
- Puis on complète ces k bits par $(l-k)$ bits $(y_{k+1}, ..., y_l)$ pris au hasard dans $\{0,1\}^{l-k}$ suivant la loi uniforme.

Avec
P_u : la probabilité équirépartie sur U
Π_j : la probabilité équirépartie sur $\{0,1\}^j$
Q_f :la probabilité image par f de P_u
Si $Y \subseteq \{0,1\}^l$ alors $\Pi_l(Y) = \frac{1}{2^l} \cdot \sharp(Y)$ Et $Q_f(Y) = P_u(f^{-1}(Y)) = \frac{1}{\sharp(U)} \cdot \sharp(f^{-1}(Y))$
X_k et X : deux variables aléatoires indépendants sur un même univers $\Omega = (y_1, y_2, ..., y_l)$

Définition de Probabilité Equirépartie

Soit $\Omega = (x_1, x_2, ..., x_n)$ un ensemble fini et P une loi de probabilité sur Ω définie par les nombres $(p_1, p_2, ..., p_n)$.
On dit que la loi P est équirépartie sur Ω si tous les p_i sont égaux.
On dit aussi qu'il y'a équiprobabilité des issues, et dans ce cas pour chaque x_i on a :

$$P(x_i) = p_i = \frac{1}{n}$$

Généralisation de la formule des probabilités composées

Soit $(A_i)_{1\leq i\leq n}$ une famille d'événement tel que $P(A_1 \cap .... \cap A_{n-1}) > 0$
alors,

$$P(\cap_{i=1}^{n} A_i) = P(A_1) * P(A_2 / A_1) * P(A_3 / A_1 \cap A_2) * .... * P(A_n / A_1 \cap A_2 ... \cap A_{n-1})$$

Preuve

Définissons sur $\{0,1\}^l$ la probabilité $p_k = P(X_k = (x_1, x_2, ..., x_k))$ c'est la probabilité de prendre k bits dans un univers Ω. Elle est définie sur les singletons par :

$$p_k(y_1, ..., y_l) = P_u(X_k = f^{-1}((y_1, ..., y_k) * \{0,1\}^{l-k})) * P(y_{k+1}, ..., y_l)$$

Or
$Y_k = ((y_1, ..., y_k) * \{0,1\}^{l-k})$: l'événement les k premières composantes sont$(y_1, ..., y_k)$
Donc

$$p_k(y_1, ..., y_l) = P_u(f^{-1}(Y_k)) * \Pi_{l-k}(y_{k+1}, ..., y_l)$$

D'où

$$p_k(y_1, ..., y_l) = \frac{1}{2^{l-k}} * Q_f(Y_k)(1)$$

Ainsi

$$p_{k+1}(y_1, ..., y_l) = \frac{1}{2^{l-k-1}} * Q_f(Y_{k+1})$$

Or d'après l'égalité suivante :

$$Y_k \cap Z_{k+1} = Y_{k+1}(2)$$

Oú Z_{k+1} est l'événement de la composante d'indice $k+1$ (y_{k+1})
On a

$$P_k(y_1, ..., y_l) * Q_f(Z_{k+1}/Y_k) = \frac{1}{2^{l-k}} * Q_f(Y_k) * Q_f(Z_{k+1}/Y_k)$$

Or d'après l'égalité ci-dessus on a :

$$Q_f(Z_{k+1}/Y_k) * Q_f(Y_k) = Q_f(Y_{k+1})$$

Donc

$$P_k(y_1, ..., y_l) * Q_f(Z_{k+1}/Y_k) = \frac{1}{2^{l-k}} * Q_f(Y_{k+1})$$

De plus

$$P_k(y_1, ..., y_l) * Q_f(Z_{k+1}/Y_k) = \frac{1}{2} * \frac{1}{2^{l-k-1}} * Q_f(Y_{k+1})$$

D'òu

$$P_k(y_1, ..., y_l) * Q_f(Z_{k+1}/Y_k) = \frac{1}{2} * P_{k+1}(y_1, ..., y_l)$$

Donc par ittération on a :

$$P_k(y_1, ..., y_l) * Q_f(Z_{k+1}/Y_k) * ... * Q_f(Z_l/Y_{l-1}) = \frac{1}{2^l} * P_l(y_1, ..., y_l)$$

De plus

$$p_l(y_1, ..., y_l) = \frac{1}{2^{l-l}} * Q_f(Y_l) = Q_f(Y_l)$$

Donc

$$P_k(y_1, ..., y_l) * Q_f(Z_{k+1}/Y_k) * ... * Q_f(Z_l/Y_{l-1}) = \frac{1}{2^l} * Q_f(y_1, ..., y_l)$$

Or d'après la formule généralisée des probabilités composées on a :

$$Q_f(Z_{k+1}/Y_k) * ... * Q_f(Z_l/Y_{l-1}) = \frac{Q_f(y_l)}{Q_f(y_k)}$$

Car
$Y_k \cap Z_{k+1} = Y_{k+1}$
$Y_{k+1} \cap Z_{k+2} = Y_k \cap Z_{k+2} \cap Z_{k+1} = Y_{k+2}$
$Y_{k+2} \cap Z_{k+3} = Y_k \cap Z_{k+3} \cap Z_{k+2} \cap Z_{k+1} = Y_{k+3}$
..
..
..
$Y_{l-1} \cap Z_l = Y_k \cap Z_{k+3} \cap Z_{k+2} \cap Z_{k+1} \cap ... \cap Z_l$
Ce qui implique que

$$P_k(y_1, ..., y_l) * \frac{Q_f(y_l)}{Q_f(y_k)} = \frac{1}{2^l} * Q_f(Y_l)$$

D'òu

$$P_k(y_1, ..., y_l) = \frac{1}{2^l} * Q_f(Y_k) = \frac{1}{2^l} * \frac{\sharp(f^{-1}(Y_k))}{\sharp(U)} = \frac{1}{2^l} * \frac{k}{2^k} = \frac{k}{2^{l+k}}$$

Donc même si le statisticien connait les k premiers bits de la suite, la probabilité qu'il retrouve le terme suivant est de l'ordre de $\frac{k}{2^{l+k}}$.

• Ainsi pour résumer nous pouvons dire qu'il existe deux types de générateurs de nombres aléatoires :
les générateurs de nombres pseudo-aléatoires et les générateurs de nombres réellement aléatoires.
Cependant la manière de les classer coincide avec celui des sources d'entropie et pour mieux illustrer cela nous avons proposé tout juste après une architecture résumant la structure de ces générateurs.

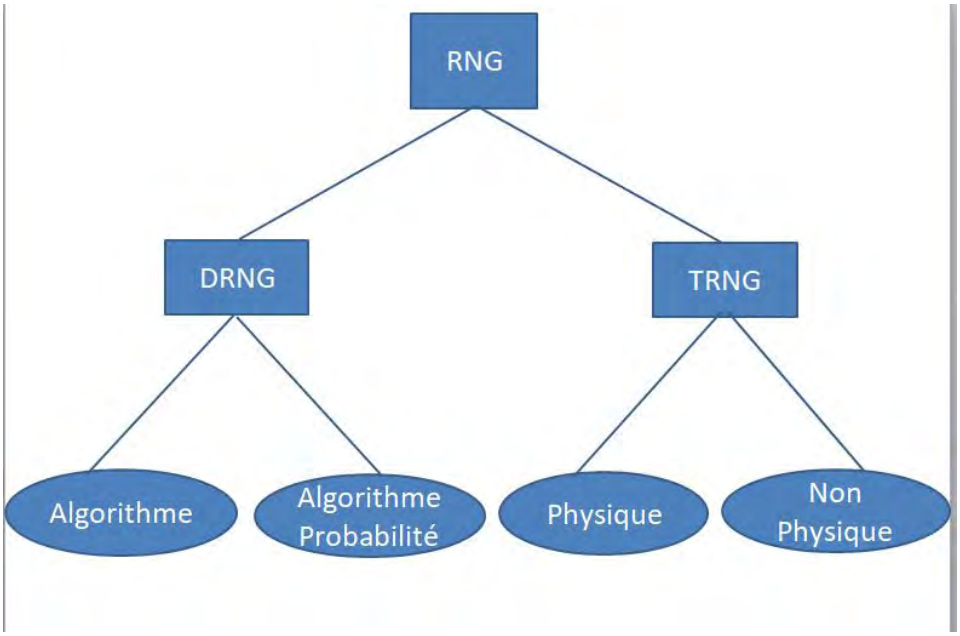


FIGURE 2.5 – Structure des RNG

Après analyse de ce schéma ci-dessus nous nous sommes posés la question suivante :
Existe t-il des générateurs cryptographiquement sûrs ?
Et à partir de cette question nous allons introduire les générateurs de nombres cryptographiquement sûrs.

2.3.3 Générateurs Cryptographiques Sûrs

2.3.3.1 Définition

Un générateur aléatoire est dit cryptographiquement sûr si sa sécurité se repose sur un protocole (**problème**) mathématique connu difficile (**NP-Complet**).
Un générateur cryptographiquement sûr a également une preuve de sécurité bien définie.
Parmi ces protocoles mathématiques nous pouvons en citer quelques uns :
★ le Décodage par Syndrome
★ le Problème de la Factorisation
★ le Problème du résidu quadratique multi-varié etc.....
Cependant pour faire la preuve de sécurité illustrée ci-dessous nous nous baserons sur le problème de la Factorisation.

2.3.3.2 Exemple

Comme exemple nous avons prit le générateur de RSA et en dessous on note le résultat du test effectué sur ce générateur.

```
 RESTART: C:\Users\user-pc\Desktop\Memoire Kancou\Kancou Implementation\Générate
ur_de_RSA.py
>>> euclideétendu(13,60)
37
>>> GRSA(9,13,7,11,25)
37
[16, 58, 9, 37, 16, 58, 9, 37, 16, 58, 9, 37, 16, 58, 9, 37, 16, 58, 9, 37, 16,
58, 9, 37, 16]
>>> euclideétendu(5,288)
173
>>> GRSA(462739,5,17,19,50)
173
[186, 288, 67, 33, 203, 135, 186, 288, 67, 33, 203, 135, 186, 288, 67, 33, 203,
135, 186, 288, 67, 33, 203, 135, 186, 288, 67, 33, 203, 135, 186, 288, 67, 33, 2
03, 135, 186, 288, 67, 33, 203, 135, 186, 288, 67, 33, 203, 135, 186, 288]
>>> euclideétendu(37,520)
253
>>> GRSA(462739,37,53,11,100)
253
[501, 293, 365, 228, 490, 205, 519, 195, 468, 381, 420, 492, 501, 293, 365, 228,
 490, 205, 519, 195, 468, 381, 420, 492, 501, 293, 365, 228, 490, 205, 519, 195,
 468, 381, 420, 492, 501, 293, 365, 228, 490, 205, 519, 195, 468, 381, 420, 492,
 501, 293, 365, 228, 490, 205, 519, 195, 468, 381, 420, 492, 501, 293, 365, 228,
 490, 205, 519, 195, 468, 381, 420, 492, 501, 293, 365, 228, 490, 205, 519, 195,
 468, 381, 420, 492, 501, 293, 365, 228, 490, 205, 519, 195, 468, 381, 420, 492,
 501, 293, 365, 228]
>>>
```

Après analyse du test effectué , nous avons pu conclu que plus les nombres (e, p, q) sont grands plus la période l'est.

Et pour terminer ce chapitre nous avons proposé une preuve de sécurité pour les générateurs pseudo-aléatoires.

2.3.4 Preuve de Sécurité

Pour faire cette preuve nous allons d'abord définir la théorie de la complexité qui est un élément central de la notion de fonction à sens unique, ensuite nous donnerons le principe et enfin nous terminerons avec la preuve de sécurité accompagnée d'un exemple plus détaillé.

2.3.4.1 Définition de la Théorie de la Complexité des Algorithmes

C'est le domaine des mathématiques et plus précisément de l'informatique théorique qui étudie formellement la quantité des ressources temporelles mais aussi spatiales (en mémoire) dont a besoin un algorithme pour résoudre un problème algorithmique.
Cette théorie est un élément central de la notion de fonction à sens unique car elle donne un sens mathématique à la notion floue de difficulté à trouver un antécédent.

2.3.4.2 Principe

Soit

$$f : \{0,1\}^* \longrightarrow \{0,1\}^*$$

une fonction asymptotique à sens unique c'est-à-dire pour toute image donnée de f il est difficile de lui trouver un antécédent en un temps polynomial.
De plus
$\{0,1\}^* = \bigcup_{n\in\mathbb{N}}\{0,1\}^n$ = l'ensemble des mots de toute longueur constitués de 0 et de 1 muni de l'opération associative que constitue la concaténation. Il est également appelé l'ensemble des mots de tailles variables.
Soit aussi

$$\begin{array}{rcccc} f_i & : & D_i & \longrightarrow & \{0,1\}^* \\ & & x & \longmapsto & f(x) \end{array}$$

une famille de fonction à sens unique où
$I = \{i/i = |(a)_2|, a \in \{0,1\}^*\}$ est l'ensemble des indices ;
$D_i = \{x = (p,q)/|p.q| = i\}$: est l'ensemble des couples d'entiers premiers dont le produit comprend i chiffres binaires.
Ce principe est basé sur les trois algorithmes suivants :

Algorithmes

- Etant donné A un algorithme efficace de la famille des fonctions à sens unique qui à partir d'un indice $i \in I$ génére un élément $x \in D_i$.
  Cet algorithme est composé de deux étapes :
  S_1 : Pour tout $n \in \mathbb{N}$, il existe un algorithme probabiliste efficace S_1 qui prend en paramètre n et génére un élément i.
  S_2 : Pour tout $x \in D_i$, il existe un algorithme probabiliste efficace S_2 qui à partir de l'indice $i \in I$ génére l'élément x.
- Soit B un algorithme efficace qui à partir de tout élément $i \in I$ et de tout élément $x \in D_i$, calcule la valeur $f_i(x)$.
- Soit C un algorithme efficace qui inverse les fonctions f_i sur une entrée de longueur i . Cependant la probabilité de trouver un antécédent de taille i de $y = f_i(x)$ est négligeable.

Le schéma donné ci-dessous, nous permet de mieux comprendre le principe de la preuve.

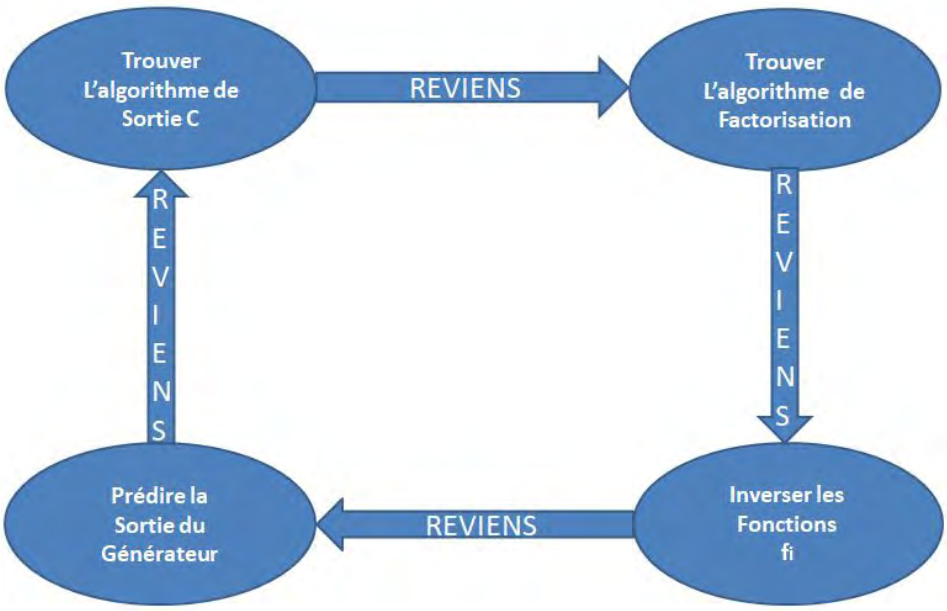


FIGURE 2.6 – Principe de la Preuve de sécurité

2.3.4.3 Preuve

Soit **y** le produit de deux nombres premiers **p** et **q** , **i** un indice appartenant à **I**.
Supposons qu'il existe un algorithme de complexité $\tau(h_n(x))$ nous permettant d'inverser les fonctions f_i.
Autrement dit nous pouvons retrouver l'antécédent **x** de l'image **y** en un temps polynômial :

$$f_i^{-1}(y) = x = (p, q) \in D_i$$

Ce qui entraine l'existence d'un autre algorithme de complexité $\tau(h_m(x))$ nous permettant de factoriser l'entier

$$y = p \cdot q$$

Donc nous pouvons dire que retrouver les entiers p et q en un temps polynômial est **bien faisable**.
Ce qui est absurde car la **factorisation** est un problème **NP-Heard**.
D'où par conclusion nous pouvons dire que la connaissance de $x = (p, q) \in D_i$ est **quasi-dur**.

Exemple 2.3.4. Soit n un entier naturel
On a $n = 27702 \in \mathbb{N}$
Appliquons à n l'algorithme S_1 on a donc :
$S_1(n) = S_1(27702) = 16 = i$ appelé paramètre de sécurité
De plus
$n = 27702 = (0110110000110110)_2$ et $i = 16 \in I$
Soient p et q deux nombres premiers tel que $x = (p, q) \in D_i$.
On a
$p = 149 = (10010101)_2$ et $p = 167 = (10100111)_2$
Donc
$x = (p, q) = (1001010110100111)_2$
Soit aussi S_2 un algorithme efficace qui prend en entrée i et génère un nombre x de même longueur binaire que n.
On a donc
$S_2(i) = x = (p, q) = 38311$
Finalement on a :
$y = f_i(x) = f(x) = f_i((p, q)) = p * q = 149 * 167 = 24883$

★ Après calcul de y nous pouvons dire que même si le statisticien connait le couple (y, i) il lui sera difficile de retrouver x parce qu'il a besoin de connaitre les deux nombres premiers qui l'on produit.
Or la factorisation est un problème difficile à résoudre mathématiquement donc on peut dire qu'inverser la fonction f_i est quasi impossible. D'òu la difficulté de retrouver x.

Conclusion

Les générateurs pseudo-aléatoires possèdent de très bonnes propriétés statistiques, mais la plupart d'entre eux ne répondent pas à l'attente des applications de transmission. Ces générateurs

ne peuvent également pas générer de séquence à l'abri de toute analyse statistique.
Mais on note une exception vis-à-vis des générateurs pseudo-aléatoires à base de registres filtres et pour illustrer cela nous avons décider de parler d'eux dans le chapitre qui suit.

Chapitre 3

Générateurs Pseudo-Aléatoires à Base De Registres Filtrés

Introduction

la plupart des générateurs pseudo-aléatoires sont construits en utilisant les registres filtrés.
En effet ces registres sont faciles à implémenter mais également économique dans leur mise en oeuvre matériel.Raisons pour laquelle on les utilise dans les chiffrements par flot mais aussi par bloc.
Pour mieux illustrer cette partie du mémoire nous :
•manipulerons et étudierons les propriétés des LFSR.
•mais également mettrons au point des algorithmes pour retrouver le plus petit LFSR générant une suite de bits donnée.

3.1 Régistre à Décalage à Rétroaction Linéaire

Définition 3.1.1. Un registre à décalage à rétroaction linéaire est un dispositif permettant d'engendrer une séquence de r bits $(s_i, ..., s_{i+r-1})$ satisfaisant à la relation de récurrence linéaire suivante :

$$a_{n+r} = c_0 a_{n+r-1} + c_1 a_{n+r-2} + ... + c_{r-1} a_n$$

où $(c_0, c_1, ..., c_{r-1}) \in \mathbb{F}_q$ sont les coefficients de connexion du LFSR.

Cependant pour mieux illustrer ces propos nous allons introduit tout juste après l'architecture de fonctionnement d'un LFSR :

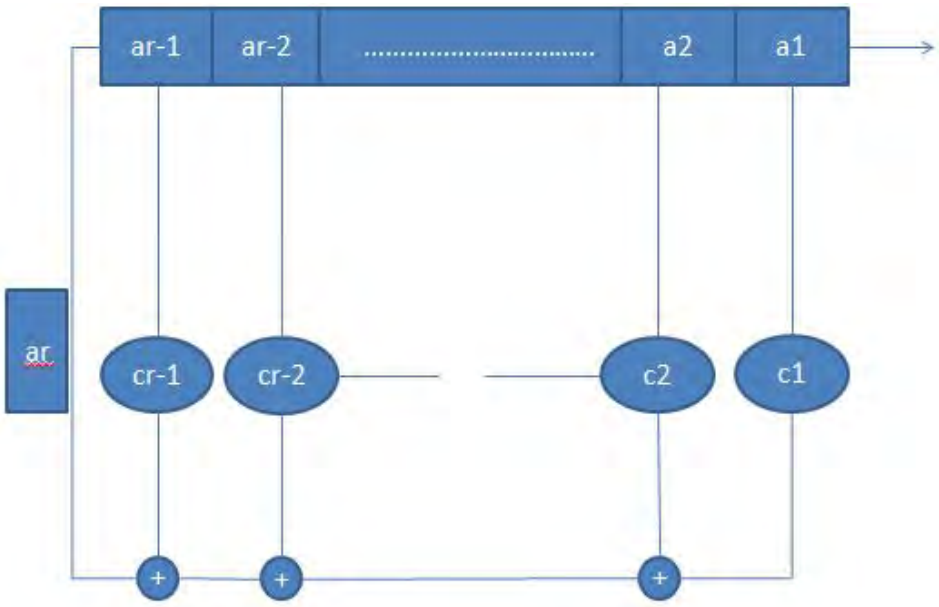


FIGURE 3.1 – Fonctionnement d'un LFSR

Propriété 3.1.1. Les propriétés fondamentales d'un LFSR sont les suivantes :

- être bien adapté à une configuration matérielle
- être capable de produire de grandes périodes de séquences binaires
- mais également des séquences qui ont de bonnes propriétés statistiques
- Et grâce à sa nature, un LFSR peut être facilement analysé en utilisant des modèles mathématiques.

Principe 3.1.1. Le principe de base d'un LFSR de $2bits$ à base de XOR initialisée avec 01 est le suivant :

- d'abord on rempli le premier étage par le résultat (bit s_2) d'une fonction de rétroaction linéaire qui prend en compte l'état d'un ou de plusieurs étages.
- Ensuite ce bit (s_2) est placé dans la cellule de gauche du premier registre.
- Après les autres bits sont décalés vers la droite et le bit s_0 va constituer la sortie du registre.
- Et au troisiéme top d'horloge l'état du registre est identique à son état initial. On dit alors que le LFSR est de période 3.

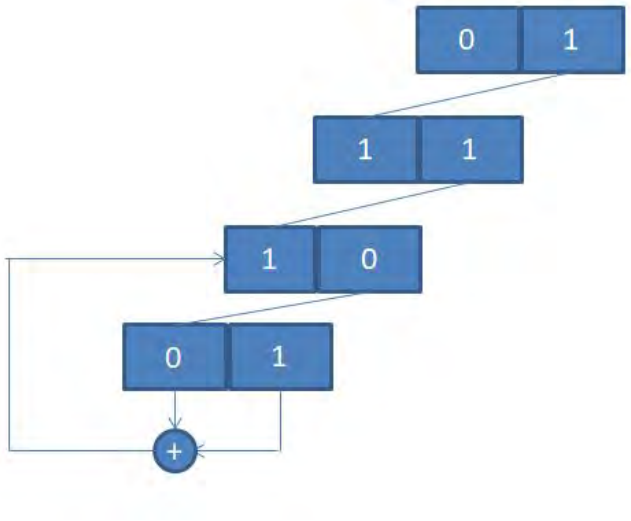


FIGURE 3.2 – Principe du LFSR

3.1.1 Séquences Générées par un LFSR

Définition 3.1.2. Soient $a_0, a_1, ..., a_{r-1}$ une séquence de bits. On dit que les (a_i) sont générés par un LFSR de taille r s'il suit la relation de récurrence linéaire suivante :

$$a_{n+r} = \sum_{i=o}^{r-1} c_i a_{n+i}$$

Proposition 3.1.1. *Toute séquence binaire à récurrence linéaire homogène d'ordre r est dite ultimement périodique si* $\forall n \geq n_0$ *sa plus petite période T est inférieure ou égale à* $2^r - 1$.
De plus si $\forall n \geq 0$ *sa période* $T = 2^r - 1$ *, alors la séquence est dite périodique.*

Exemple 3.1.1. Soient deux suites récurrentes linéaires d'ordre 4 initialisées par $(a_0, a_1, a_2, a_3) = (0, 0, 0, 1)$.

$$a_{n+4} = a_{n+3} + a_{n+2}$$

$$a_{n+4} = a_{n+1} + a_n$$

• Pour la séquence1 on a :

$$S_1 = \underline{00}011011011.....\forall n \geq 2$$

De plus la période $T = 3 < (2^4 - 1)$ donc cette suite est ultimement périodique.
• Pour la séquence2 on a :

$$S_2 = \underline{0}0010011010111\underline{0}0010011010111.....\forall n \geq 0$$

De plus la période $T = 15 = (2^4 - 1)$; donc cette suite est périodique.
$*$ Puisque la sécurité des LFSR dépend de la complexité linéaire, nous allons maintenant nous intéresser à la plus petite relation de récurrence permettant de reproduire la séquence $(a_n) = (a_0, a_1, ..., a_{r-1})$ et pour ce faire associons à cette dernière la série génératrice suivante :

$$a(X) = \sum_{n=o}^{\infty} a_n X^n$$

Cette approche qui a été introduite en 1959 par Niezereiter nous a permis de décrire le développement en série formelle de la séquence (a_n) sous forme d'une fraction rationnelle et ceci grâce à l'intervention du polynôme de rétroaction ci-dessous :

$$f(X) = 1 + c_1 X + c_2 X^2 + c_3 X^3 + ... + c_r X^r$$

Pour ce faire introduisons d'abord le théorème suivant :
Mais avant tout qu'est ce qu'une série formelle

3.1.1.1 Série Formelle

Définition 3.1.3. Soit A un anneau commutatif intègre.
On appelle série formelle(ou série génératrice) sur A toute expression symbolique :
$a_0 + a_1 X + a_2 X^2 + a_3 X^3 + ...$ où les $a_i \in A$, $\forall i \in \mathbb{N}$.
Le symbole X est appelé l'indéterminée.
On note une série formelle en X par :

$$S(X) = (a_n) = \sum_{i \in \mathbb{N}} a_i X^i = a_0 + a_1 X + a_2 X^2 + a_3 X^3 + ...$$

Exemple 3.1.2. $S(x) = 1 + 5x + x^2 + 7x^3 + ...$ série formelle sur
$(\mathbb{Z}, +, \times)$
$S(x) = 1 + \frac{1}{4}x + \frac{3}{5}x^2 + ...$ série formelle sur $(\mathbb{Q}, +, \times)$
$S(x) = 1 + x + x^2 + ...$ série formelle sur $(\mathbb{F}_2, +, \times)$

Remarque. Une série formelle n'est pas une fonction mais simplement une expression liée à une suite d'éléments (a_i).
De plus x n'est pas une variable et a_0 est toujours différent de zéro $(a_0) \neq 0$
Maintenant nous pouvons introduire le théorème :

Théoreme 3.1.2. *Soient $(a_n) = (a_0, a_1, ..., a_{r-1})$ une séquence de r bits ; elle est produite par un LFSR de polynôme de rétroaction $f(X) = 1 + c_1X + c_2X^2 + c_3X^3 + ... + c_rX^r$ si et seulement si son développement en série formelle $a(X) = \sum_{n=o}^{\infty} a_nX^n$ s'écrit de la manière suivante :*

$$a(X) = \frac{g(X)}{f(X)}$$

où f et g sont des polynômes de $\mathbb{F}_2[X]$ et $deg(g) < deg(f)$
De plus $g(X) = \sum_{i=1}^{r} X^i \sum_{j=0}^{i-1} c_{i-j}a_j$ est déterminé par l'état initial du registre.

3.1.1.2 Preuve du Théorème

Soit $(a_n) = (a_0, a_1, ..., a_{r-1})$ la séquence produite par un LFSR .
$\Rightarrow$ Supposons que son polynôme de rétroaction est $f(X) = 1 + c_1X + c_2X^2 + c_3X^3 + ... + c_rX^r$ et montrons que sa série génératrice s'écrit sous cette forme :

$$a(X) = \frac{g(X)}{f(X)}$$

• Soit $a(X) = \sum_{n=o}^{\infty} a_nX^n$
avec $(a_n) = a_0, a_1, ...$ la séquence de sortie générée par un LFSR
Or $a_n = \sum_{i=1}^{r} c_ia_{n-i}$
On a donc :
$a(X) = \sum_{n=o}^{\infty} \sum_{i=1}^{r} c_ia_{n-i}X^n = \sum_{i=1}^{r} c_iX^i \sum_{n=0}^{\infty} a_{n-i}X^{n-i}$
$= \sum_{i=1}^{r} c_iX^i \{ \sum_{n=i}^{\infty} a_{n-i}X^{n-i} + \sum_{n=o}^{i-1} a_{n-i}X^{n-i}\}$
Pour la somme $\sum_{n=i}^{\infty} a_{n-i}X^{n-i}$ faisons un changement de variable c'est-à-dire $n' = n - i$
On a donc
$\sum_{n=i}^{\infty} a_{n-i}X^{n-i} = \sum_{n'=0}^{\infty} a_{n'}X^{n'} = \sum_{n=0}^{\infty} a_nX^n$
En remplaçant on a :
$a(X) = \sum_{i=1}^{r} c_iX^i \{ \sum_{n=0}^{\infty} a_nX^n + \sum_{n=o}^{i-1} a_{n-i}X^{n-i}\} = \sum_{i=1}^{r} c_iX^i \{a(X) + \sum_{n=o}^{i-1} a_{n-i}X^{n-i}\}$
$a(X) - \sum_{i=1}^{r} c_iX^i \{a(X)\} = \sum_{i=1}^{r} c_iX^i \sum_{n=o}^{i-1} a_{n-i}X^{n-i}$
$a(X)\{1 - \sum_{i=1}^{r} c_iX^i\} = \sum_{i=1}^{r} c_iX^i \sum_{n=o}^{i-1} a_{n-i}X^{n-i}$
Puisque $f \in \mathbb{F}_2[X]$ on a donc $\{1 - \sum_{i=1}^{r} c_iX^i\} = \{1 + \sum_{i=1}^{r} c_iX^i\} = f(X)$
Donc
$a(X)f(X) = \sum_{i=1}^{r} c_iX^i \sum_{n=o}^{i-1} a_{n-i}X^{n-i} = \sum_{n=o}^{i-1} X^n \sum_{i=1}^{r} c_ia_{n-i}$
En posant $g(X) = \sum_{n=o}^{i-1} X^n \sum_{i=1}^{r} c_ia_{n-i}$

On a donc
$a(X)f(X) = g(X)$
D'où

$$a(X) = \frac{g(X)}{f(X)}$$

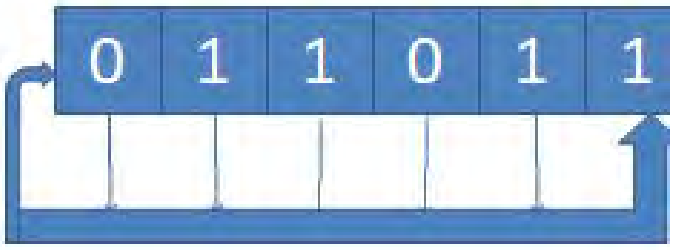
$\Leftarrow$ Supposons que le développement en série formelle de la suite $(a_n)_{n\geq 0}$ s'écrit sous cette forme : $a(X) = \frac{g(X)}{f(X)}$ avec $pgcd(g(X), f(X)) \neq 1$ et montrons l'existence de son polynôme de rétroaction.
Puisque $pgcd(g(X), f(X)) \neq 1$ nous pouvons donc introduire les 4 points suivants :

- Soit (a_n) une séquence binaire à rétroaction linéaire d'ordre r dont l'état initial est non nul.
- Son polynôme de rétroaction minimal est l'unique polynôme unitaire f_0 de $\mathbb{F}_2[X]$ tel qu'il existe $g_0 \in \mathbb{F}_2[X]$ avec $deg(g_0) < deg(f_0)$ et $pgcd(g_0, f_0) = 1$ vérifiant :

$$a(X) = \frac{g_0(X)}{f_0(X)}$$

- Ce polynôme de rétroaction minimal f_0 est le polynôme de plus bas degré parmi les polynômes de rétroaction de tous les LFSRs possibles qui peuvent générés la séquence (a_n).
- Et ce plus bas degré de f_0 également appelé complexité linéaire du LFSR est la longueur du plus petit LFSR permettant d'engendrer la séquence (a_n).

Exemple 3.1.3. Soit (a_n) la séquence produite par le LFSR suivant :



Son polynôme de rétroaction est :

$$f(X) = 1 + X + X^2 + X^3 + X^4 + X^5$$

Soit g le polynôme déterminé par l'état initial du registre :

$$g(X) = \sum_{n=o}^{4} X^n \sum_{i=o}^{n} c_{n-i} a_i$$

On a ainsi :

$g_0(X) = X^0 \cdot c_0 \cdot a_0 = 1$

$g_1(X) = X^1(c_1 \cdot a_0 + c_0 \cdot a_1) = X^1(1+1) = 0$

$g_2(X) = X^2(c_2 \cdot a_0 + c_1 \cdot a_1 + c_0 \cdot a_2) = X^2(1+1+0) = 0$

$g_3(X) = X^3(c_3 \cdot a_0 + c_2 \cdot a_1 + c_1 \cdot a_2 + c_0 \cdot a_3) = X^3(1+1+0+1) = X^3$

$g_4(X) = X^4(c_4 \cdot a_0 + c_3 \cdot a_1 + c_2 \cdot a_2 + c_1 \cdot a_3 + c_0 \cdot a_4) = X^4(1+1+0+1+1) = 0$

Finalement on a :

$$g(X) = X^3 + 1$$

Or la série génératrice de $(a_n)_{n\geq 0}$ est :

$$a(X) = \frac{g(X)}{f(X)}$$

Donc on a :

$$a(X) = \frac{X^3+1}{1+X+X^2+X^3+X^4+X^5} = \frac{X^3+1}{(X^3+1)\cdot(1+X+X^2)} = \frac{1}{1+X+X^2} = \frac{1}{f_0(X)}$$

$\Rightarrow f_0(X) = 1+X+X^2$ donc la séquence (a_n) est produite par un LFSR de longueur $(2^2-1) = 3$ et de complexité linéaire égale à 2.

Remarque. Si on connait la séquence on connait le polynôme de rétroaction. Cependant si ce polynôme est irréductible , unitaire et de période égale à $2^L - 1$ avec L la complexité linéaire dans un corps donné $\mathbb{F}_q$, il devient automatiquement le polynome minimal.Dans le cas contraire le polynôme minimal sera l'un de ses diviseurs.

$*$ Donc nous pouvons conclure que d'après le théorème énoncé ci-dessus il existe une bijection entre l'ensemble des suites d'ordre r et l'ensemble des séries formelles et pour mieux illustrer cela nous avons proposé tout juste après un schéma comme guise d'exemple.

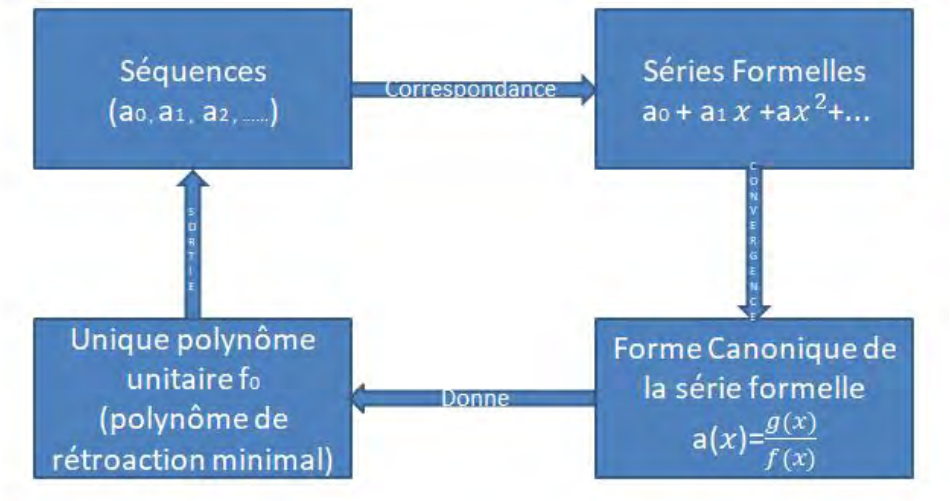


FIGURE 3.3 – Bijection entre les séries formelles et les suites récurrentes linéaires

3.2 LFSR et Cryptographie

Un LFSR n'est pas cryptographiquement sûr parce que les progrès faites récemment dans le domaine de la cryptanalyse notamment les attaques dites algébriques proposées dernièrement mettent toutefois en évidence des faiblesses inhérentes à de nombreux générateurs de ce type. En effet

- si les coefficients du polynôme de rétroaction sont publics, il suffit à un attaquant qui connait L bits consécutifs de la suite d'appliquer la relation de récurrence pour retrouver tous les bits suivants.
- dans le cas contraire où les coefficients de rétroaction sont secrets, l'algorithme de Berlekamp-Massey permet de les reconstituer ainsi que l'état initial à partir de $2L$ bits de suite chiffrante.

3.2.1 Attaque dite Algèbrique

3.2.1.1 Méthode par résolution du système linéaire

Une méthode plus efficace pour calculer la complexité linéaire d'une suite $A = (a_0, a_1, a_2, ..., a_{r-1})$ est de remarquer qu'un LFSR de longueur r génère la suite A si et seulement si ces coefficients $(c_0, c_1, c_2, ..., c_{r-1})$ vérifient un système de $n - r$ équations linéaires définies sur $\mathbb{F}_q$ par :

$$\left\{\begin{array}{rcl} c_0a_{r-1} + c_1a_{r-2} + ... + c_{r-1}a_0 & = & a_r \\ c_0a_r + c_1a_{r-1} + ... + c_{r-1}a_1 & = & a_{r+1} \\ c_0a_{r+1} + c_1a_r + ... + c_{r-1}a_2 & = & a_{r+2} \\ \cdots & & \\ \cdots & & \\ \cdots & & \\ c_0a_{n-r-2} + c_1a_{n-r-3} + ... + c_{r-1}a_{n-1} & = & a_{n-r-1} \end{array}\right\}$$

Afin de résoudre un tel système à r inconnues dans $\mathbb{F}_q$, une méthode est de l'exprimer sous la forme matricielle

$$M \cdot c = a$$

avec M(matrice) , c(les coefficients) et $a = (a_r, a_{r+1}, a_{r+2}, ..., a_{n-r-1})$ la séquence
Chaque solution de ce système nous donnera un LFSR et toute absence de solutions nous permettra d'affirmer que la complexité linéaire est supérieure à r.

Exemple 3.2.1. Soit $T = (1,1,1,0,1,1,0,1)$ une séquence , pour $l = 4$
on a l'équation matricielle suivante :

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Après avoir appliqué la méthode du pivot de Gauss on a

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Ce qui implique que :

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

finalement

$$\left\{ \begin{array}{rcl} c_0 & = & 1 \\ c_1 + c_3 & = & 0 \\ c_1 + c_2 & = & 0 \end{array} \right\} \Longrightarrow \left\{ \begin{array}{rcl} c_0 & = & 1 \\ c_1 & = & 1 \\ c_2 & = & 1 \\ c_3 & = & 1 \end{array} \right\}$$

En remplaçant dans la formule de la matrice compagnon A on a :

$$\begin{bmatrix} 0 & 0 & 0 & c_0 \\ 1 & 0 & 0 & c_1 \\ 0 & 1 & 0 & c_2 \\ 0 & 0 & 1 & c_3 \end{bmatrix} \quad = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Maintenant calculons l'inverse de la matrice A puisqu'elle est inversible.
D'après la formule :

$$A^{-1} = \frac{1}{detA} \cdot {}^{T}comA$$

on a :
$detA = -1 = 1 \neq 0$ dans $\mathbb{F}_2$

De plus

$$\text{ComA} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \Longrightarrow 1 \times {}^{T}comA = A^{-1} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Pour la vérification on a

$$AA^{-1} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = I_4$$

Maintenant essayons de retrouver les premiers termes de la suite :

On a donc :

$$(1110)A^{-1} = (1110) \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} = (\mathbf{1}111)$$

$$(1111)A^{-1} = (1111) \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} = (\mathbf{0}111)$$

$$(0111)A^{-1} = (0111) \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} = (\mathbf{1}011)$$

$$(1011)A^{-1} = (1011) \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} = (\mathbf{1}101)$$

$$(1101)A^{-1} = (1101) \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} = (1110)$$

On remarque que le dernier résultat est identique aux 4 premiers bits utilisés dans la démonstration. Donc pour déterminer l'état initial du LFSR correspondant on utilisera les chiffres écrient en gras des 4 premiers résultats. Ainsi la sortie sera :

$$S = 1011110111101111$$

Cependant retrouvez l'état initial revient à retrouver le LFSR correspondant.

3.2.2 Attaque par Berlekamp-Massey

En 1969, J.Massey a montré que l'algorithme proposé par Berlekamp pour le décodage des codes BCH pouvait être adapté pour retrouver le polynôme de rétroaction d'un LFSR à partir uniquement des $2L$ premiers bits de la séquence produite s. D'où le fameux nom de Berlekamp-Massey.

Définition 3.2.1. La complexité linéaire d'une suite est la dimension de l'espace vectoriel engendré par la suite et l'ensemble de ses décalées. Elle est aussi notée :
$L = k(a) = dim(vect((a), D.(a), D^2(a), ...)$ où D est l'opérateur décalage et $D^i.(a)$ est la i^{eme} décalée de (a).

Principe 3.2.1. Cet algorithme retourne pour $N = 2L$ bits le polynôme de rétroaction f du LFSR de départ.
Ce LFSR génére les N premiers bits de la suite et est de complexité linéaire L.

Entrée : $s_0, s_1, ..., s_{n-1}$ une séquence de longueur n
Initialisation :
$g_j(x)$; $h_j(x)$; m_j / $g_0(x) = 1$; $h_0(x) = x$; $m_0 = 0$
Pour **i** $= 0$.....**do** :
$* \; g_{j+1}(x) = g_j(x) - b_j \cdot h_j(x)$
Avec b_j le coefficient de x^j dans $G(x) \cdot g_j(x)$
Dans $\mathbb{F}_2$ on a $b_j^{-1} = b_j = 1$ si $b_j \neq 0$

$$* \; h_{j+1} = \begin{cases} b_j^{-1} \cdot x \cdot g_j(x) & \text{si } b_j \neq 0 \text{ et } m_j \geqslant 0 \\ x \cdot h_j(x) \end{cases}$$

Et :

$$* \; m_{j+1} = \begin{cases} -m_j & \text{si } b_j \neq 0 \text{ et } m_j \geqslant 0 \\ m_j + 1 & \text{sinon} \end{cases}$$

Si on définit $m(x)$ comme étant le polynôme minimal de la séquence alors on a :

$$m(x) = x^k \cdot g_{2k}(\frac{1}{x})$$

Exemple 3.2.2. Soit $S = 1101$ une séquence

$G(x) = 1 + x + x^3$ sa série génératrice
On a donc :

j	$g_j(x)$	$h_j(x)$	m_j	b_j
0	1	x	0	1
1	$1+x$	x	0	0
2	$1+x$	x^2	1	1
3	$1+x+x^2$	$x+x^2$	-1	0

Ainsi pour $k = 2$ on a

$$m(x) = x^2 \cdot g_4(\frac{1}{x}) = x^2 \cdot (\frac{1}{x^2} + \frac{1}{x} + 1) = (1 + x + x^2)$$

Finalement $m(x) = 1 + x + x^2$
Donc son LFSR correspondant est : $U_{n+2} = U_{n+1} + U_n$

★ Les registres à décalage à rétroaction linéaires sont des dispositifs extrêmement rapides et d'implémentation peu coûteuse , et les séquences qu'ils engendrent ont de bonnes propriétés statistiques.
Mais pour contourner l'effet du problème de la linéarité de ces LFSR trois méthodes ont été mis sur table :

- Associer une fonction non linéaire aux sorties de plusieurs LFSR(Registre combiné ou filtré)
- Ou utiliser une fonction de filtrage non linéaire (1-résiliente) basée sur le contenu d'un seul LFSR
- Ou bien utiliser plusieurs LFSR en parallèle qui peuvent provenir d'un autre LFSR(k-résilientes)

Ainsi pour une explication plus détaillée on prendra les registres combinés ou filtrés comme guise d'exemple.

3.3 Régistres Combinés ou Filtrés

Même lorsqu'on choisit de manière appropriée le polynôme de rétroaction du registre, la complexité linéaire de la suite produite reste généralement trop faible pour se prémunir des attaques comme celle de l'algorithme de berlekamp-Massey et afin de surmonter cet obstacle et d'augmenter la taille de l'espace des clefs, nous allons utiliser k LFSRs en parallèle et combinés leurs sorties par une fonction booléenne.

3.3.1 Fonction Booléenne

Définition 3.3.1. Soit $\mathbb{F}_{q^n}$ un corps à q^n éléments et $\mathbb{F}_{q^n}^{(r)}$ l'espace vectoriel de dimension r sur $\mathbb{F}_{q^n}$. On appelle fonction booléenne à r variables toute fonction ayant r entrées et une sortie dans $\mathbb{F}_{q^n}$. Autrement dit :

$$\begin{array}{rccc} f & : & \mathbb{F}_{q^n}^{(r)} & \longrightarrow & \mathbb{F}_{q^n} \\ & & (a_0, a_1, ..., a_{r-1}) & \longrightarrow & f(a_0, a_1, ..., a_{r-1}) = a_i \end{array}$$

3.3.2 Exemples de Régistres Combinés

3.3.2.1 Générateur de Geff

Définition 3.3.2. Le générateur de Geff est un générateur défini par 3 LFSRs dont les polynômes de rétroaction sont primitifs et de degrés L_1, L_2, L_3 deux-à-deux premiers entre eux.
Les sorties de ces registres sont réunis dans une même fonction booléenne non seulement équilibrée mais aussi avec un degré le plus élevé possible :
De plus la complexité linéaire de la suite produite par ce générateur est :

$$L = \sum L_i = L_1 + L_2 + L_3$$

et sa période est :

$$T = (2^{L_1} - 1) \cdot (2^{L_2} - 1) \cdot (2^{L_3} - 1) \lesssim 2^L$$

Principe 3.3.1. Soit k un entier naturel , f une fonction booléenne définie par :

$$\begin{array}{rccc} f & : & \mathbb{F}_2^k & \longrightarrow & \mathbb{F}_2 \\ & & (x_0, x_1, ..., x_{k-1}) & \longrightarrow & f(x_0, x_1, ..., x_{k-1}) = x_i \end{array}$$

L'architecture donné ci-dessous illustre mieux le principe abordé :

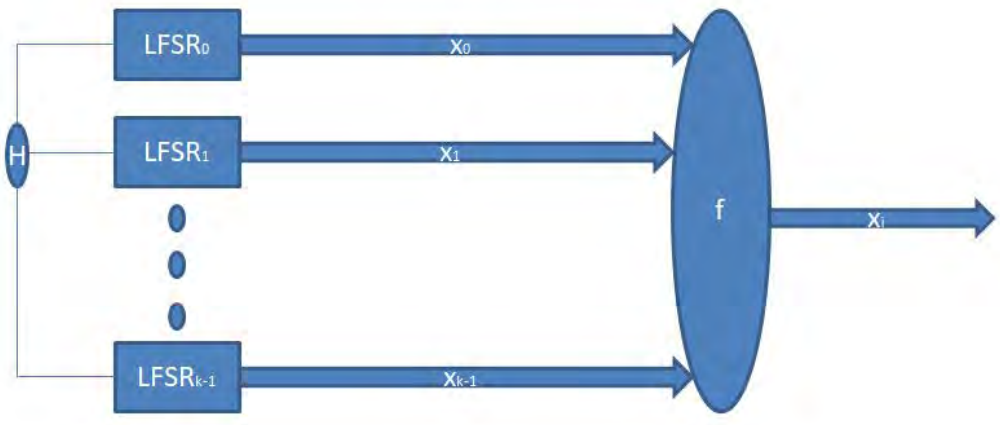


FIGURE 3.4 – Structure des Générateurs Combinés

Exemple 3.3.1. Soient :

$$U_{n+3} = U_{n+1} + U_n$$

$$V_{n+4} = V_{n+1} + V_n$$

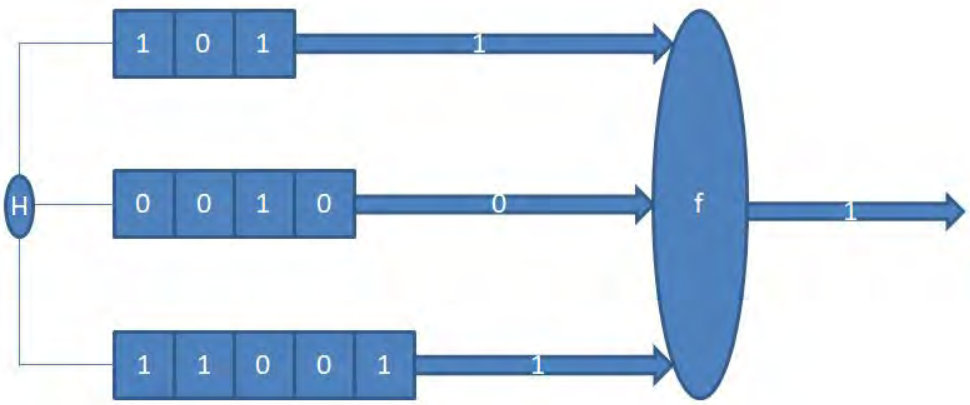
$$W_{n+5} = W_{n+3} + W_n$$

trois LFSRs de complexité linéaire 3 , 4 et 5 respectivement.

Et soit :

$$\begin{array}{rccc} f & : & \mathbb{F}_2^3 & \longrightarrow & \mathbb{F}_2 \\ & & (x_1, x_2, x_3) & \longrightarrow & f(x_1, x_2, x_3) = x_1x_2 + x_2x_3 + x_3 \end{array}$$

la fonction booléenne qui leur est associée.
On a donc après initialisation :



Ainsi $f(x_1, x_2, x_3) = f(1, 0, 1) = x_1x_2 + x_2x_3 + x_3 = 0 + 0 + 1 = 1$
D'où la sortie $X(t) = 1$ et la complexité $L = L_1L_2 + L_2L_3 + L_3 = 12 + 20 + 5 = 37$
Le générateur de Geff a deux avantages importants :
★ son rendement comporte une distribution moyenne égale à 0 et à 1
★ et pour décoder ce dispositif sans la connaissance de la clef, il serait nécessaire de résoudre un système d'équation non-linéaire dont la solution est quasi-impossible.
Mais ceci n'empêchera pas aux attaquants de mettre en évidence les faiblesses inhérentes à de nombreux générateurs de ce genre et ceux-ci grâce aux attaques par Corrélation.

3.3.3 Attaques par Corrélation

Définition 3.3.3. Cette attaque développée par T.Siegenthaler est de type :"diviser pour mieux régner" et consiste à retrouver l'initialisation de chacun des registres indépendamment des autres.

Principe 3.3.2. Pour cela on essaie toute les initialisations du premier registre jusqu'à ce que le nombre de coincidence entre la sortie $X(t)$ de la fonction booléenne et la suite $S(t)$ du premier régistre soit égale à leur probabilité de corrélation.
Cet algorithme s'arrête lorsqu'on est sûr que l'une de ces initialisations marches et on fait de même pour les deux autres régistres $V(t)$ et $W(t)$.

Exemple 3.3.2. Reconsidèrons le générateur de Geff avec :

$$f(x_1, x_2, x_3) = x_1x_2 + x_2x_3 + x_3 = x_2(x_1 + x_3) + x_3 = x_1x_2 + x_3(x_2 + 1)$$

On a donc les probabilités de sorties suivantes :
$P(X(t) = S(t)) = P(X_2(t) = 1) + P(X_3(t) = 0) \cdot P(X_2(t) = 1) = \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4}$
$P(X(t) = V(t)) = P(X_1(t) = 1) \cdot P(X_3(t) = 0) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$
$P(X(t) = W(t)) = P(X_2(t) = 0) = \frac{1}{2}$

Remarque. Cette attaque peut-être généralisée en regardant la corrélation de la sortie avec une somme de m sortie de LFSRs.
Donc pour s'en protéger, il faudrait prendre des fonctions de combinaisons garantissant qu'il n'y ait pas de corrélation. Parmi ces fonctions , nous pouvons citer les m-résilientes.
En effet ces fonctions d'ordre m sont non corrélées mais également équilibrées.

Conclusion

La plupart des générateurs pseudo-aléatoires sont construits en utilisant les registres à décalage à rétroaction linéaire.
Mais ces derniers on pour inconvénient de générer des suites linéaires et c'est pour cette raison que les PRNG sont construits en combinant à l'aide d'une fonction non linéaire plusieurs registres à décalage de tailles différentes.
Ainsi pour mesurer la qualité de ces nouveaux générateurs ,nous allons leur faire passer des tests statistiques .

Chapitre 4

Tests de Génération de Nombres Pseudo-aléatoires

Introduction

La nécessité de nombres pseudo-aléatoires dans de nombreuses applications cryptographiques à pousser une des sections du NIST a travaillé depuis longtemps à la description d'une battérie de 16 tests statistiques.
Mais en 2001 une nouvelle version à été mise à jour et celle-ci comporte 15 tests dont certains sont paramétrables et d'autre non.
Ces tests ont un aspect local mais le fait de les regrouper nous permettent de déterminer un aspect global qui est l'aléa.
Cependant la méthodologie de ces derniers est basée sur les critères de Golomb et le formalisme du test d'hypothèse.

4.1 Les Critères de Golomb et Test d'Hypothèse

4.1.1 Les Critères de Golomb

Ces critères ou postulats qui sont aux nombres de trois ont été proposés en 1982 par Golomb.
Cependant ces propriétés ne garantissent en aucun cas la sécurité cryptographique mais assure la proposition d'un bon aléa.

4.1.1.1 La propriété d'équilibre

Dans la période d'une séquence binaire périodique, le nombre de uns et le nombre de zéro doivent être égaux à une unité près.

4.1.1.2 La propriété des répétitions

Dans une période, les séries de la forme $(\mu, \alpha, \alpha, \alpha, ..., \alpha, \delta)$ doivent apparaitre suivant une bonne répartition avec $\mu \neq \alpha$ et $\mu \neq \delta$.
Autrement dit pour une séquence de période $q^L - 1$, les mots de taille $1 \leq k \leq L-2$ doivent apparaitre $(q-1)q^{L-k-2}$ fois.

4.1.1.3 Le théorème d'auto-Corrélation

Soit $C_a(\tau)$ la fonction d'auto-corrélation définie par :

$$C_a(\tau) = \sum_{i=o}^{L-1} (-1)^{s_i + s_{i+\tau}} = \begin{cases} L & \text{si } \tau = 0 \text{ mod n} \\ b & \text{si } \tau \neq 0 \text{ mod n} \end{cases}$$

Elle prend 2 valeurs suivant que $\tau = 0$ ou non.

Remarque. Si L est impaire alors le nombre d'apparition de mots de taille $1 \leq k \leq L-2$ de zéros et de uns diffère à une unité près.

Exemple 4.1.1. Soit $S = 10110001111100110100100000101011$ la séquence définie par le polynôme minimal $f(x) = x^5 + x^3 + 1$ et de période $2^5 - 1 = 31$.

Elle vérifie :
• la propriété d'équilibre puisqu'il y'a 16 uns et 15 zéros.
• la propriété de répétition puisque le nombre de mots de taille $1 \leq k \leq 5-2 = 3$ doivent apparaitre $(2-1) * 2^{5-k-2} = 2^{3-k}$ fois.
On a donc :
$\Rightarrow$ pour $k = 1$, $2^2 = 4$ apparitions de mots de taille 1 de zéros et de uns à une unité près (dont 4 séries de la forme (0) et 5 séries de la forme (1)).
$\Rightarrow$ pour $k = 3$, $2^0 = 1$ apparition de mots de taille 3 de zéros et de uns à une unité près (dont 1 série de la forme (000) et 0 série de la forme (111)).
$\Rightarrow$ pour $k = 2$, $2^1 = 2$ apparitions de mots de taille 2 de zéros et de uns à une unité près (dont 2 séries de la forme (00) et 3 séries de la forme (11)).
$\Rightarrow$ On a aussi une série de la forme (0000) et une série de la forme (11111)
• le théorème d'auto-Corrélation pour $\tau = 2$:

$$C_a(2) = \begin{cases} 5 & \text{si } \tau = 0 \text{ mod n} \\ -1 & \text{si } \tau = 2 \text{ mod n} \end{cases}$$

D'où cette séquence vérifie les 3 postulats de Golomb.

4.1.1.4 Architecture des postulats

L'architecture proposé ci-dessous nous résume le passage de toute séquence pseudo-aléatoire dans les trois critères de Golomb. .

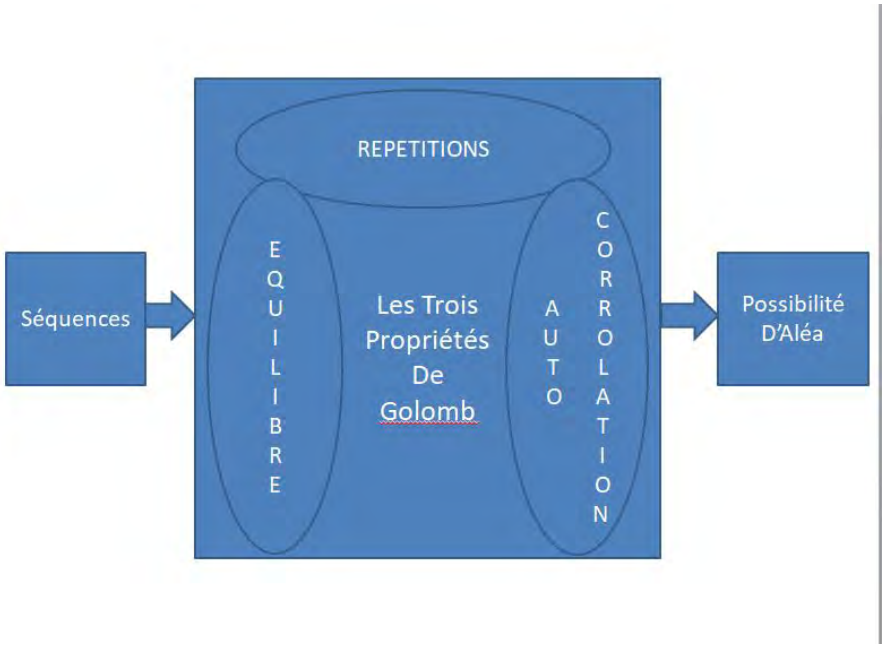


FIGURE 4.1 – Critères De Golomb

4.1.2 Le Test d'hypothèse

Le test d'hypothèse est un test composé de l'hypothèse nulle (H_0) et de l'hypothèse alternative inverse (H_a).
Il ne permet pas de déterminer si (H_0) est fondamentalement vraie ou non, mais plutôt de voir si (H_0) est une hypothèse cohérente avec les données observées.
Puisque nous travaillons avec les suites pseudo-aléatoires, des erreurs peuvent alors se produire.

4.1.2.1 Les Types d'Erreurs

On distingue deux types d'erreurs :

- l'erreur de 1^{ier} espèce (ou d'ordre 1) noté $\alpha \in [0,001; 0,01]$.
  Ici l'hypothèse (H_0) est accepter alors qu'elle est fausse .
  Autrement dit on accepte une suite produite par un générateur de bonne qualité alors que celui-ci n'est pas aléatoire.

- l'erreur de 2^{nd} espèce (ou d'ordre 2) noté β.

  Pour ce cas l'hypothèse (H_0) est rejeter alors qu'elle est vraie.

  Autrement dit on rejete une suite issue d'un générateur non aléatoire alors que celle-ci a les qualités nécessaires pour valider l'hypothèse nulle.

4.1.2.2 Tableau

Le tableau suivant résume tout sur les deux types d'erreurs énoncés.

		Conclusion (Décision du test)	Conclusion (Décision du test)
		H_0 acceptée	H_0 rejetée
Réalité	H_0 est vraie	Pas d'erreur	Erreur Type2
Réalité	H_0 est fausse	Erreur Type1	Pas d'erreur

Cependant le choix de α est important car il permet de déterminer le nombre de suites à tester.Donc plus le nombre et la taille des séquences sont importants et plus le temps d'exécution des tests l'est aussi.
Ce qui nous permettra d'introduire la notion de valeur critique.

4.1.2.3 La Valeur Critique

La valeur critique ou p-value est la probabilité qu'une suite produite par un générateur parfait ait l'air moins aléatoire que la suite examinée par le test. Cette valeur dépend de la forme de l'hypothèse alternative (H_a). Si la p-value $\geq \alpha$ alors l'hypothèse nulle est acceptée et dans le cas contraire elle est rejetée.
Cependant la validation de l'hypothèse nulle est basée sur une p-value $> \alpha$ mais également sur l'uniformité réparties dans l'intervalle $[\alpha, 1]$ des valeurs des p-values $> \alpha$.
Voici la répartition en 2 parties des 15 tests du NIST :

Tests non Paramétrés (8)	**Tests Paramétrés** (7)
Le test de fréquence Monobit	Le test de fréquence par Bloc
Le test de Rang d'une Matrice Binaire	Le test de Run par Bloc
Le test de Run	Le test de Somme Cumulative
Le test Universel de Maurer's	Le test de Complexité Linéaire
Le test de TFDS	Le test de CMAC
Le test de CMSC	Le test de Série
Le test Approximatif d'entropie	Le test d'Excursions Aléatoires
	Le test de VEA

4.2 Tests de Génération de Nombres Pseudo-Aléatoires

4.2.1 Tests non Paramétrés

4.2.1.1 Test de fréquence Monobit

Définition 4.2.1. Le test de fréquence Monobit est un test qui détermine si le nombre de uns et de zéro dans une séquence est approximativement le même que celui prévu pour une séquence véritablement aléatoire. Pour ce faire on commence toujours par convertir les uns et les zéro de la séquence d'entrée en valeur de 1 et -1 grâce à la formule $x_i = (2u_i - 1)$.

Le principe qui suit nous montre que les séquences aléatoires passent avec succés ce test.

Principe 4.2.1. Soit U_n une suite de taille n dans la base $\{-1, 1\}$ et $U = \sum_{i=0}^{n} U_i$.
Soit S_{obs} la valeur absolue de la somme définie par :

$$S_{obs} = \frac{\mid S_n \mid}{\sqrt{n}}$$

Alors la p-value définie par :

$$p - value = erfc(\frac{S_{obs}}{\sqrt{2}})$$

est supérieure à 0.01.

Ainsi pour mieux appuier cela nous avons proposé un algorithme généralisant le principe abordé.

Algorithm 1 Test de fréquence Monobit

Require: La séquence de bits (ϵ) est convertie en valeurs de -1 et 1

Require: $n \geq 100$

Ensure: P-value

if $n < 100$ **then**

Le test ne peut pas être effectué

else

for i from 0 to n **do**

$sum \longleftarrow sum + X_i$

end for

$\mid S_n \mid \longleftarrow \mid sum \mid$

$S_{obs} \longleftarrow \frac{|S_n|}{\sqrt{n}}$

$p - value \longleftarrow erfc(\frac{S_{obs}}{\sqrt{2}})$

end if

Conclusion $P - value \geq 0.01$

Cependant pour une bonne visibilité des choses nous avons proposé ce tableau en guise d'exemple.

n	séquence	seuil	paramètre	p-value	Etat
100	1100100100001111110110101010001000 0100001011010001100001000110100110 001001100011001100010100010111000	$\alpha = 0.1$ ou 0.05 ou 0.01	non	0.2578	passe
150	1000111111110010000100101100101001 10100111000111111101101000100110000011 1000100000111000100110101001001 0 1011100111001010011111011 1000 1001101011110101001101001 11111	$\alpha = 0.1$ ou 0.05 ou 0.01	non	0.4884	passe

Après analyse des faits nous avons conclu qu'avec les trois seuils proposés, les deux séquences passent avec succés le test de fréquence Monobit.

4.2.1.2 Test de Rang d'une Matrice Binaire

Définition 4.2.2. Le test sur le rang de la matrice binaire est un test qui permet de calculer le rang des sous-matrices disjointes issues de la séquence entière.
Ce test à pour but de vérifier la dépendance linéaire entre les sous-chaînes de longueurs fixes de la séquence d'origine.
Cependant le choix ne se porte pas sur la longueur du bloc, mais sur celle des lignes.
Le principe qui suit nous montre que les séquences aléatoires passent avec succés ce test.

Principe 4.2.2. Soit U_n une suite de longueur n et $\lambda^2(obs)$ la mesure de la qualité de proportion définie par :

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

où F_M est le nombre de matrices de rang complet M ; F_{M-1} le nombre de matrices de rang $M-1$ et $(N - F_M - F_{M-1})$ le nombre de matrices restantes.
Alors la p-value définie par :

$$p - value = igamc(1, \frac{\lambda^2(obs)}{2})$$

est supérieure à 0.01.

Ainsi pour mieux appuier cela nous avons proposé un algorithme généralisant le principe abordé.

Algorithm 2 Test de Rang d'une Matrice Binaire

Require: La séquence de bits (ϵ)

Require: $n \geq 38MQ$

Require: M=Q=32

Ensure: P-value

if $n < 38MQ$ **then**

Le test ne peut pas être effectué

else

$N \longleftarrow \lfloor \frac{n}{MQ} \rfloor$

Calculer F_M , F_{M-1} and $N - F_M - F_{M-1}$

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

$p - value \longleftarrow igamc(1, \frac{\lambda^2(obs)}{2})$

end if

Conclusion $P - value \geq 0.01$

Cependant pour une bonne visibilité des choses nous avons proposé ce tableau en guise d'exemple.

n	séquence	seuil	paramètre	p-value	Etat
100	11001001000011111101101010100010001000010110100011000010001101001100100011000110011000101000010111000	$\alpha = 0.1$ ou 0.05 ou 0.01	$M = 5$	0.1210	passe
150	10001111111100100001001011001010011010011100011111110110100001001100001111000100100101101010010010101100111100101001111101110001111000	$\alpha = 0.1$ ou 0.05 ou 0.01	$M = 5$	0.4084	passe

Après analyse des faits nous avons conclu qu'avec les trois seuils proposés, les deux séquences passent avec succés le test de Rang d'une Matrice binaire.

4.2.1.3 Test de Run

Définition 4.2.3. Le test de Run est un test qui permet de déterminer le nombre total de séquence de bits identiques ininterrompue dans une séquence entière.
Ce test permet aussi de déterminer si le nombre de run de uns et de zèro de différentes longueurs est approximativement le même que celui prévu pour une séquence aléatoire.
Pour ce faire, on calcul d'abord avant le test la proportion π de uns de la séquence d'entrée $\pi = \sum_i \frac{\epsilon_i}{n}$ et l'inégalité $\mid \pi - \frac{1}{2} \mid \geq \frac{2}{\sqrt{n}} = \tau$.
Le principe qui suit nous montre que les séquences passent avec succés ce test.

Principe 4.2.3. Soit U_n une suite de taille n et $V_n(obs)$ le nombre total de runs définie par :

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

où $r(k) = \begin{cases} 0 & \text{si } U_k = U_{k+1} \\ 1 & \text{sinon} \end{cases}$
Alors la p-value définie par :

$$p - value = erfc(\frac{\mid V_n(obs) - 2n\pi(1-\pi) \mid}{2\sqrt{2n\pi(1-\pi)}})$$

est supérieure à 0.01.

Ainsi pour mieux appuier cela nous avons proposé un algorithme généralisant le principe abordé.

Algorithm 3 Test de Run

Require: La séquence de bits (ϵ)

Require: $n \geq 100$

Require: $\tau \longleftarrow \frac{2}{\sqrt{n}}$

Ensure: P-value

  if $n < 100$ **then**

    Le test ne peut pas être effectué

  else

    if $\mid \pi - \frac{1}{2} \mid \geq \tau$ **then**

      Le test de Run n'a pas besoin d'être effectué

    else

      for j from 1 to n **do**

        $\pi \longleftarrow \pi + \frac{\epsilon_j}{n}$

      end for

      for k from 1 to $n-1$ **do**

        $V_n(obs) \longleftarrow V_n(obs) + (r(k) + 1)$

      end for

      Calculer $p - value = erfc(\frac{|V_n(obs) - 2n\pi(1-\pi)|}{2\sqrt{2n\pi(1-\pi)}})$

    end if

  end if

  Conclusion $P - value \geq 0.01$

Cependant pour une bonne visibilité des choses nous avons proposé ce tableau en guise d'exemple.

n	séquence	seuil	paramètre	p-value	Etat
100	110010010000111111011010101000100 0100001011010001100001000110100110 001001100011001100010100010111000	$\alpha = 0.1$ ou 0.05 ou 0.01	non	0.5007	passe
150	1000111111110010000100101100101001 101001110001111111011010001001100011 10001000001110001001101010010010 10111001110010100111110111000 100110101111010100110100111111	$\alpha = 0.1$ ou 0.05 ou 0.01	non	0.6835	passe

Après analyse des faits nous avons conclu qu'avec les trois seuils proposés, les deux séquences passent avec succés le test de Run.

4.2.1.4 Test Universel de Maurer

Définition 4.2.4. Le test de Maurer est un test qui permet de déterminer le nombre de bits entre deux modèles identiques.
Le but de ce test est de détecter si la séquence peut être compressée de manière significative sans perte d'information et toute séquence nettement compressible est considérée comme non aléatoire.
Le principe qui suit nous montre que les séquences aléatoires passent avec succés ce test.

Principe 4.2.4. Soit U_n une suite de taille n ; L la longueur de chaque bloc et Q le nombre de blocs dans la séquence d'initialisation.
Soit K le nombre de bloc dans la séquence de test définie par :

$$K = \lfloor \frac{n}{L} \rfloor - Q$$

f_n la somme des distances de $\log_2$ entre les modèles de L-bits :

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

Et σ la variable définie par : $\sigma = c\sqrt{(\frac{variance(L)}{K})}$ où $c = 0.7 - \frac{0.8}{L} + (4 + \frac{32}{L}) \cdot \frac{K^{\frac{-3}{L}}}{15}$
Alors la p-value :

$$p - value = erfc(|\ \frac{f_n - expectedValue(L)}{\sqrt{2}\sigma}\ |)$$

est supérieure à 0.01.

Ainsi pour mieux appuier cela nous avons proposé un algorithme généralisant le principe abordé.

Algorithm 4 Test Universel de Maurer

Require: La séquence de bits (ϵ)

Require: $n \geq (Q+K)L$

Require: $Q \longleftarrow 10 \cdot 2^L$

Require: $6 \leq L \leq 16$

Ensure: P-value

if $n < (Q+K)L$ **then**

Le test ne peut pas être effectué

else

$c \longleftarrow (0.7 - \frac{0.8}{L} + (4 + \frac{32}{L}) \cdot \frac{K^{\frac{-3}{L}}}{15})$

$\sigma \longleftarrow c\sqrt{(\frac{variance(L)}{K})}$

$K = \lfloor \frac{n}{L} \rfloor - Q \approx 1000 \cdot 2^L$

for i from $(Q+1)$ to $(Q+K)$ **do**

$Somm \longleftarrow Somm + log_2(i - T_j)$

end for

$f_n \longleftarrow \frac{1}{K} \cdot Somm$

Calculer ExpectedValue

$p - value \longleftarrow erfc(| \frac{f_n - expectedValue(L)}{\sqrt{2}\sigma} |)$

end if

Conclusion $P - value \geq 0.01$

Cependant pour une bonne visibilité des choses nous avons proposé ce tableau en guise d'exemple.

n	séquence	seuil	paramètre	p-value	Etat
100	11001001000011111101101010100010001000010110100011000010001101001100010011000110011000101000101110000	$\alpha = 0.1$ ou 0.05 ou 0.01	$L = 6$	$5.7629\ 10^{-5}$	ne passe pas
150	10001111111100100001001011001010011010011100011111110110100010011000011100010000011100010011010100100101011100111001010011111011100010011010111101010011010011111	$\alpha = 0.1$ ou 0.05 ou 0.01	$L = 6$	0.0019	ne passe pas

Après analyse des faits nous avons conclu qu'avec les trois seuils proposés aucune des deux séquences ne passent le test Universel Statistique de Maurer.

4.2.1.5 Test de Transformée de Fourier Discrète (Spectrale)

Définition 4.2.5. Le test de transformée de fourier discrète (spectrale) est un test qui permet de déterminer les hauteurs maximales de la séquence.
Le but de ce test consiste à détecter les caractéristiques de périodicité de la séquence d'origine. Autrement dit ce test permet de vérifier si le nombre de pics (N_1) dépassant le seuil $(T = \sqrt{(\log \frac{1}{0.05}) \cdot n})$ de 95% est significativement différent de 5%.
Le principe suivant nous montre que les séquences aléatoires passent avec succés ce test.

Principe 4.2.5. Soit X une suite de taille n dans la base {-1,1}.
M le module de S′ définie par :
$M = \mid S' \mid$ où S′ sous-chaîne de S=DFT(X) constituée des $\frac{n}{2}$ premiers termes. Et d la différence normalisée entre le nombre de fréquence observé et celui attendu des composants dépassant le seuil de 95%. Cette différence est définie par :

$$d = \frac{(N_1 - N_0)}{\sqrt{\frac{n(0.95)(0.05)}{4}}}$$

où $N_0 = \frac{(0.95)n}{2}$ nombre de pics attendu inférieur à T de 95%
Alors la p-value :

$$p - value = erfc(\frac{\mid d \mid}{\sqrt{2}})$$

est supérieure à 0.01.

Ainsi pour mieux appuier cela nous avons proposé un algorithme généralisant le principe abordé.

Algorithm 5 Test de Transformée de Fourier Discrète (Spectrale)

Require: La séquence de bits (ϵ) est convertie en valeurs de -1 et 1

Require: $n \geq 1000$

Ensure: P-value

if $n < 1000$ **then**

  Le test ne peut pas être effectué

else

  $T \longleftarrow \sqrt{(\log \frac{1}{0.05})n}$

  Calculer $N_1 < T$

  $N_0 \longleftarrow \frac{(0.95)n}{2}$

  $d \longleftarrow \frac{(N_1 - N_0)}{\sqrt{\frac{n(0.95)(0.05)}{4}}}$

  $p - value \longleftarrow erfc(\frac{|d|}{\sqrt{2}})$

end if

Conclusion $P - value \geq 0.01$

Cependant pour une bonne visibilité des choses nous avons proposé ce tableau en guise d'exemple.

n	séquence	seuil	paramètre	p-value	Etat
100	11001001000011111101101010101000100 01000010110100011000010001101001100 00100110001100110001010001011100 0	$\alpha = 0.1$ ou 0.05 ou 0.01	non	0.1687	passe
150	100011111111001000010010110010100 1 1010011100011111110110100010011000011 10001000001110001001101010010010 10111001110010100111110111000 10011010111101010011010011111 1	$\alpha = 0.1$ ou 0.05 ou 0.01	non	0.3489	passe

Après analyse des faits nous avons conclu qu'avec les trois seuils proposés ,les deux séquences passent avec succés le test de transformée de fourier discrète (spectrale).

4.2.1.6 Test de Correspondance des Modèles Sans Chevauchement

Définition 4.2.6. Le test de correspondance des modèles sans chevauchement est un test qui permet de déterminer le nombre d'occurences des modèles pré-spécifiées.
Le but de ce test consiste à détecter les générateurs qui produisent trop d'occurences pour un modèle non périodique donné.
Ce test utilise une fenêtre à m bits pour rechercher le motif spécifique (B) correspondant.
Si B n'est pas trouvé, la fenêtre glisse d'une position de bit et dans le cas contraire la fenêtre est réinitialisée sur le bit après le motif trouvé et la recherche reprend.
Le principe suivant nous montre que les séquences aléatoires passent avec succés ce test.

Principe 4.2.6. Soit U_n une suite de taille n ; M la longueur de chaque bloc ; $\mu = \frac{(M-m+1)}{2^m}$ la moyenne et
$\sigma^2 = M(\frac{1}{2^m} - \frac{2m-1}{2^{2m}})$ la variance.
Soit $\lambda^2(obs)$ la mesure de la corrélation entre le nombre de hits observé dans le modèle et celui de correspondance attendu dans l'hypothèse d'aléa :

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

où W_j est le nombre de fois que B apparait dans le bloc j
Alors la p-value :

$$p - value = igamc(\frac{N}{2}, \frac{\lambda^2(obs)}{2})$$

est supérieure à 0.01.

Ainsi pour mieux appuier cela nous avons proposé un algorithme généralisant le principe abordé.

Algorithm 6 Test de Correspondance des Modèles Sans Chevauchement

Require: La séquence de bits (ϵ)

Require: $n \geq 2^{20}$

Require: m et M

Require: Une chaine binaire B de taille m

Ensure: P-value

if $n < 2^{20}$ **then**

  Le test ne peut pas être effectué

else

  $\mu \longleftarrow \frac{(M-m+1)}{2^m}$

  $\sigma^2 \longleftarrow M(\frac{1}{2^m} - \frac{2m-1}{2^{2m}})$

  for j from 1 to N **do**

    Calculer W_j

    $\lambda^2(obs) \longleftarrow \lambda^2(obs) + \frac{(W_j-\mu)^2}{\sigma^2}$

  end for

  $p - value \longleftarrow igamc(\frac{N}{2}, \frac{\lambda^2(obs)}{2})$

end if

Conclusion $P - value \geq 0.01$

Cependant pour une bonne visibilité des choses nous avons proposé ce tableau en guise d'exemple.

n	séquence	seuil	paramètre	p-value	Etat
100	11001001000011111101101010100010 01000010110100011000010001101001 10 0010011000110011000101000101110 00	$\alpha = 0.1$ ou 0.05 ou 0.01	$M = 25$	0.6426	passe
150	1000111111110010000100101100101001 10100111000111111101101000100110000 11 100010000011100010011010100 10010 10111001110010100111110111000 1001101011110101001101001 11111	$\alpha = 0.1$ ou 0.05 ou 0.01	$M = 25$	$7.2933\ 10^{-7}$	ne passe pas

Après analyse des faits nous avons conclu qu'avec les trois seuils proposés, la séquence de 150 bits ne passe pas le test de correspondance des modèles sans chevauchement.

4.2.1.7 Test Approximatif d'Entropie

Définition 4.2.7. Comme le test de série, le test approximatif d'entropie permet lui aussi de déterminer la fréquence de tous les modèles possibles de m-bits qui recouvrent en partie la séquence entière.
Le but de ce test est de comparer la fréquence superposée de deux longueurs consécutives (m et m+1) par rapport au résultat attendu pour une séquence aléatoire.

Le principe suivant nous montre que les séquences aléatoires passent avec succés ce test.

Principe 4.2.7. Soit U_n une suite de taille n ; i la valeur de m-bit , $j = \log_2(i)$,
$\pi_i = C_j^m$ et $C_m^i = \frac{\sharp i}{n}$.
Soient $\varphi^{(m)} = \sum_{i=0}^{2^{m-1}} \pi_i \log \pi_i$
et $\lambda^2(obs)$ la mesure de l'efficacité de la valeur observée de ApEn(m).
Cette mesure est définie par :

$$\lambda^2 = 2n[\log 2 - ApEn(m)] avec ApEn(m) = \varphi^{(m)} - \varphi^{(m+1)}$$

Alors la p-value :

$$p - value = igamc(2^{m-1}, \frac{\lambda^2}{2})$$

est supérieure à 0.01.

Ainsi pour mieux appuier cela nous avons proposé un algorithme généralisant le principe abordé.

Algorithm 7 Test Approximatif d'Entropie

Require: la séquence de bits (ϵ)

Require: $n \geq 100$

Require: $m < \lfloor \log_2(n) \rfloor - 5$

Ensure: P-value

if $n < 100$ **then**

Le test ne peut pas être effectué

else

for i from 0 to $2^m - 1$ **do**

$C_m^i \longleftarrow \frac{\sharp i}{n}$

$\pi_i \longleftarrow C_{\log_2 i}^m$

$\varphi^{(m)} \longleftarrow \varphi^{(m)} + \pi_i \log \pi_i$

end for

$ApEn(m) \longleftarrow \varphi^{(m)} - \varphi^{(m+1)}$

$\lambda^2 \longleftarrow 2n[\log 2 - ApEn(m)]$

$p - value \longleftarrow igamc(2^{m-1}, \frac{\lambda^2}{2})$

end if

Conclusion $P - value \geq 0.01$

Cependant pour une bonne visibilité des choses nous avons proposé ce tableau en guise d'exemple.

n	séquence	seuil	paramètre	p-value	Etat
100	11001001000011111101101010100010 010000010110100011000010001101001100 0010011000110011000101000101110000	$\alpha = 0.1$ ou 0.05 ou 0.01	$M = 2$	0.1602	passe
150	100011111111001000010010110010100 1 1010011100011111110110100010011000011 1000100001110001001101010010010 1011100111001010011111011100 0 1001101011110101001101001111111	0.1	$M = 2$	0.0637	ne passe pas
150	1000111111110010000100101100101001 1010011100011111110110100010011000011 1000100001110001001101010010010 10111001110010100111110111000 1001101011110101001101001111111	$\alpha = 0.05$ ou 0.01	$M = 2$	0.0637	passe

Après analyse des faits nous avons conclu qu'avec le seuil de 10% la séquence de 150 bits ne

passe pas le test approximatif d'entropie.

4.2.2 Tests Paramétrés

4.2.2.1 Test de fréquence par bloc

Définition 4.2.8. Le test de fréquence par bloc est un test qui permet de déterminer si la fréquence de uns qui se trouvent dans les blocs de M-bits est approximativement $\frac{M}{2}$.
Cependant si $M = 1$ ce test dégénère pour le test de fréquence Monobit.
Le principe suivant nous montre que les séquences aléatoires passent avec succés ce test.

Principe 4.2.8. Soit U_n une suite paramètrée par M ;
π_j la proportion de 1 dans chaque bloc de M-bits définie par :

$$\pi_j = \sum_{i=1}^{M} \frac{U_{(j-1)\cdot M+i}}{M}$$

et $\lambda^2(obs)$ la mesure de la qualité de proportion définie par :

$$\lambda^2(obs) = 4 \cdot M \cdot \sum_{j=1}^{N} (\pi_j - \frac{1}{2})^2$$

Alors la p-value donnée par :

$$p - value = igamc(\frac{N}{2}, \frac{\lambda^2(obs)}{2})$$

où igamc est la fonction gamma incomplète est supérieure à 0.01.

Ainsi pour mieux appuier cela nous avons proposé un algorithme généralisant le principe abordé.

Algorithm 8 Test de Fréquence par bloc

Require: La séquence de bits (ϵ)
Require: $n \geq 100$ et $n \geq MN$
Require: $M \geq 20$ et $M > 0.01n$
Require: $N < 100$
Ensure: P-value
  if $n < 100$ **then**
    Le test ne peut pas être effectué
  else
    for j from 1 to N **do**
      for i from 1 to M **do**
        $sum \longleftarrow sum + \frac{\epsilon_{(j-1)\cdot M+i}}{M}$
        $\pi_j \longleftarrow sum$
        $obs \longleftarrow obs + (\pi_j - \frac{1}{2})^2$
      end for
    end for
    $\lambda^2(obs) \longleftarrow 4 \cdot M \cdot \cdot obs$
    $p - value \longleftarrow igamc(\frac{N}{2}, \frac{\lambda^2(obs)}{2})$
  end if
  Conclusion $P - value \geq 0.01$

Cependant pour une bonne visibilité des choses nous avons proposé ce tableau en guise d'exemple.

n	séquence	seuil	paramètre	p-value	Etat
100	110010010000111111011010101000100 0100001011010001100001000110100110 001001100011001100010100010111000	$\alpha = 0.1$ ou 0.05 ou 0.01	$M = 9$	0.2597	passe
150	1000111111110010000100101100101001 1010011100011111110110100010011000011 10001000001110001001101010010010 10111001110010100111110111000 100110101111010100110100111111	$\alpha = 0.1$ ou 0.05 ou 0.01	$M = 50$	0.4374	passe

Après analyse des faits nous avons conclu qu'avec les trois seuils proposés, les deux séquences passent avec succés le test de fréquence par bloc.

4.2.2.2 Test du plus long Run de Un par bloc

Définition 4.2.9. le test of longest run of ones est un test basé sur la plus longue série de uns contenus dans les blocs de M-bits.
Le but de ce test est de déterminer si la longueur de la plus longue série d'unités dans la séquence à tester est cohérente avec celle attendue dans une séquence aléatoire.
Le principe suivant montre que les séquences aléatoires passent avec succés le test of longest run of ones.

Principe 4.2.9. Soit U_n une suite de longueur n et $\lambda^2(obs)$ la mesure de la qualité de proportion définie par :

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

où π_i et v_i sont respectivement les proportions et fréquences proposées par le NIST ; K et N des valeurs déterminées à partir de celle de M.
Alors la p-value définie par :

$$p - value = igamc(\frac{K}{2}, \frac{\lambda^2(obs)}{2})$$

est supérieure à 0.01.

Ainsi pour mieux appuier cela nous avons proposé un algorithme généralisant le principe abordé.

Algorithm 9 Test du plus long Run de Un par bloc

Require: La séquence de bits (ϵ)

Require: $n \geq 128$

Require: M, K, and N

Ensure: P-value

if $n < 128$ **then**

Le test ne peut pas être effectué

else

Calculer le sous block maximale de "1"

Calculer les fréquences v_i pour $i = (0, ..., 6)$

for i from 0 to K **do**

$\lambda^2(obs) \longleftarrow \lambda^2(obs) + \frac{(v_i - N\pi_i)^2}{N\pi_i}$

end for

$p - value \longleftarrow igamc(\frac{K}{2}, \frac{\lambda^2(obs)}{2})$

end if

Conclusion $P - value \geq 0.01$

Cependant pour une bonne visibilité des choses nous avons proposé ce tableau en guise d'exemple.

n	séquence	seuil	paramètre	p-value	Etat
100	11001001000011111101101010100010001000010110100011000010001101001100100011000110010000101000111000	$\alpha = 0.1$ 0.05	$M = 8$	0.0343	ne passe pas
100	11001001000011111101101010100010001000010110100011000010001101001100100011000110010000101000111000	0.01	$M = 8$	0.0343	passe
150	10001111111100100001001011001010011010011100011111110110100001001100001110001001101010010010010	$\alpha = 0.1$ ou 0.05 ou 0.01	$M = 8$	0.5958	passe

Après analyse des faits nous avons conclu qu'avec les seuils de 10% et 5% la séquence de 100 bits ne passe pas le test de Run par bloc.

4.2.2.3 Test de Somme Cumulative

Définition 4.2.10. Le test de la somme cumulée est un test qui permet de déterminer la distance maximale atteinte par une séquence ϵ codée sur la base $\{-1, 1\}$ grâce à la formule $X_i = 2\epsilon_i - 1$. Ce test nous indique si la somme cumulée des séquences partielles est trop grande ou trop petite par rapport à celle attendue pour une séquence aléatoire.
Le principe qui suit nous montre que ces séquences aléatoires passent avec succés ce test.

Principe 4.2.10. Soit X_n une suite dans la base $\{-1, 1\}$ et S_k la somme cumulée définie par :
$S_k = S_{k-1} + X_k$ pour le mode 1 et
$S_k = S_{k-1} + X_{n-k+1}$ pour le mode 2
Alors,
pour tout $z = \max_{1 \leq k \leq n} \mid S_k \mid$ on a la p-value définie :

$$p - value = 1 - \sum_{k=\frac{(\frac{-n}{z}+1)}{4}}^{\frac{(\frac{n}{z}-1)}{4}} (\Phi(\frac{(4k+1)z}{\sqrt{n}}) - \Phi(\frac{(4k-1)z}{\sqrt{n}})) + \sum_{k=\frac{(\frac{-n}{z}-3)}{4}}^{\frac{(\frac{n}{z}-1)}{4}} (\Phi(\frac{(4k+3)z}{\sqrt{n}}) - \Phi(\frac{(4k+1)z}{\sqrt{n}}))$$

est supérieure à 0.01.

Ainsi pour mieux appuier cela nous avons proposé un algorithme généralisant le principe abordé.

Algorithm 10 Test de Somme Cumulative

Require: La séquence de bits (ϵ) est convertie en valeurs de -1 et 1

Require: $n \geq 100$

Require: Mode=0(sens normal) || Mode=1(sens inverse)

Ensure: P-value

if $n < 100$ **then**

Le test ne peut pas être effectué

else

$z \longleftarrow \max_{1 \leq k \leq n} \mid S_k \mid$

for k from $\lfloor \frac{(\frac{-n}{z}+1)}{4} \rfloor$ to $\lfloor \frac{(\frac{n}{z}-1)}{4} \rfloor$ **do**

$som1 \longleftarrow som1 + (\Phi(\frac{(4k+1)z}{\sqrt{n}}) - \Phi(\frac{(4k-1)z}{\sqrt{n}}))$

end for

for k from $\lfloor \frac{(\frac{-n}{z}-3)}{4} \rfloor$ to $\lfloor \frac{(\frac{n}{z}-1)}{4} \rfloor$ **do**

$som2 \longleftarrow som2 + (\Phi(\frac{(4k+3)z}{\sqrt{n}}) - \Phi(\frac{(4k+1)z}{\sqrt{n}}))$

end for

$p - value \longleftarrow 1 - som1 + som2$

end if

Conclusion $P - value \geq 0.01$

Cependant pour une bonne visibilité des choses nous avons proposé ce tableau en guise d'exemple.

n	séquence	seuil	paramètre	p-value	Etat
100	11001001000011111101101010100010 01000010110100011000010001101001 10 00100110001100110001010001011 1000	$\alpha = 0.1$ ou 0.05 ou 0.01	non	0.2191	passe
150	10001111111100100001001011001010 01 10100111000111111101101000100110000 11 10001000011100010011010100100 10 10111001110010100111110111000 10011010111101010011010011111 1	$\alpha = 0.1$ ou 0.05 ou 0.01	non	0.6543	passe

Après analyse des faits nous avons conclu qu'avec les trois seuils proposés, les deux séquences passent avec succés le test de somme Cumulative.

4.2.2.4 Test de Complexité Linéaire

Définition 4.2.11. Le test de complexité linéaire est un test qui permet de calculer la longueur du plus petit LFSR permettant de générer les blocs de séquences données.
Cette longueur L_n appelée complexité linéaire est obtenue grâce à l'algorithme de Berlekamp-Massey.
Le principe suivant nous montre que les séquences aléatoires passent avec succés ce test.

Principe 4.2.11. Soit U_n une suite de taille n et M la longueur en bits d'un bloc.
Soit μ la moyenne de ces complexités définie par :

$$\mu = \frac{M}{2} + \frac{(9 + (-1)^{M+1})}{36} - \frac{(\frac{M}{3} + \frac{2}{9})}{2^M}$$

T_i la variable obtenue en combinant μ, M et L_i :

$$L_i = (-1)^M \cdot (L_i - \mu) + \frac{2}{9}$$

Et $\lambda^2(obs)$ le mesure du nombre d'occurences observées définies par :

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

Alors la p-value :

$$p - value = igamc(\frac{K}{2}, \frac{\lambda^2(obs)}{2})$$

est supérieure à 0.01.
Ainsi pour mieux appuier cela nous avons proposé un algorithme généralisant le principe abordé.

Algorithm 11 Test de Complexité Linéaire

Require: La séquence de bits (ϵ)

Require: $n \geq 10^6$

Require: $500 \leq M \leq 5000$ et $N \geq 200$

Ensure: P-value

if $n < 10^6$ **then**

Le test ne peut pas être effectué

else

Calculer les fréquences v_i pour $i = (0, ..., 6)$

for i from 0 to K **do**

$\lambda^2(obs) \longleftarrow \lambda^2(obs) + \frac{(v_i - N\pi_i)^2}{N\pi_i}$

end for

$p - value \longleftarrow igamc(\frac{K}{2}, \frac{\lambda^2(obs)}{2})$

end if

Conclusion $P - value \geq 0.01$

Cependant pour une bonne visibilité des choses nous avons proposé ce tableau en guise d'exemple.

n	séquence	seuil	paramètre	p-value	Etat
100	11001001000011111101101010100010 01000010110100011000010001101001 10 00100110001100110001010001011 1000	$\alpha = 0.1$ ou 0.05 ou 0.01	$M = 3$	0.3656	passe
150	10001111111100100001001011001010 01 10100111000111111101101000100110000 11 10001000011100010011010100100 10 10111001110010100111110111000 10011010111101010011010011111 1	$\alpha = 0.1$ ou 0.05 ou 0.01	$M = 3$ $M = 3$	0.0744 0.0744	ne passe pas passe passe

Après analyse des faits nous avons conclu qu'avec le seuil de 10% la séquence de 100 bits ne passe pas le test de Complexité Linéaire.

4.2.2.5 Test de Correspondance des Modèles Avec Chevauchement

Définition 4.2.12. Le test de correspondance des modéles avec chevauchement est un test qui permet de déterminer le nombre d'occurences spécifiées des cordes.
Ce test est utilisé de la même manière que le test de correspondance des modèles sans chevauchement.La seule diffèrence est que lorsque le motif (B) est trouvé, la fenêtre ne glisse qu'un bit avant de reprendre la recherche.

Le principe introduit ci-dessous nous montre que certaines séquences aléatoires passent avec succés ce test.

Principe 4.2.12. Soit U_n une suite de taille n ; M la longueur de chaque bloc ;
$\lambda = \frac{(M-m+1)}{2^m}$ et $\eta = \frac{\lambda}{2}$.
Soit $\lambda^2(obs)$ la mesure de la corrélation définie par :

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

Où $v_i(i = 0, ...5)$ est le nombre d'occurences de B dans chaque bloc tel que v_0 est incrémenté lorsqu'il n'ya aucune occurence de B dans un sous-chaîne, v_1 est incrémenté pour une occurence de B, ... et v_5 est incrémenté pour 5 ou plus d'occurences de B.
Alors la p-value :

$$p - value = igamc(\frac{K}{2}, \frac{\lambda^2(obs)}{2})$$

est supérieure à 0.01.

Ainsi pour mieux appuier cela nous avons proposé un algorithme généralisant le principe abordé.

Algorithm 12 Test de Correspondance des Modèles Avec Chevauchement

Require: La séquence de bits (ϵ)

Require: $n \geq MN$

Require: M et $m \approx \log_2(M)$

Require: Une chaine binaire B de taille m

Ensure: P-value

if $n < 2^{20}$ **then**

Le test ne peut pas être effectué

else

$\lambda \longleftarrow \frac{(M-m+1)}{2^m}$

$K \approx 2\lambda$

if $K \neq 5$ **then**

Recalculer les valeurs de π_i

end if

if $N \cdot min(\pi_i) > 5$ **then**

for i from 0 to 5 **do**

Calculer les fréquences v_i

$\lambda^2(obs) \longleftarrow \lambda^2(obs) + \frac{(v_i - N\pi_i)^2}{N\pi_i}$

end for

$p - value \longleftarrow igamc(\frac{5}{2}, \frac{\lambda^2(obs)}{2})$

end if

end if

Conclusion $P - value \geq 0.01$

Cependant pour une bonne visibilité des choses nous avons proposé ce tableau en guise d'exemple.

n	séquence	seuil	paramètre	p-value	Etat
100	11001001000011111101101010100010 01000010110100011000010001101001 10 00100110001100110001010001011 1000	$\alpha = 0.1$ ou 0.05 ou 0.01	$M = 25$	0.7354	passe
150	10001111111100100001001011001010 01 10100111000111111101101000100110000 11 10001000011100010011010100100 10 10111001110010100111110111000 10011010111101010011010011111 1	$\alpha = 0.1$ ou 0.05 ou 0.01	$M = 25$	0.3647	passe

Après analyse des faits nous avons conclu qu'avec les trois seuils proposés les deux séquences

passent avec succés le test de correspondance des modèles avec chevauchement.

4.2.2.6 Test de Série

Définition 4.2.13. Le test de série est un test qui permet de déterminer la fréquence de tous les modèles possibles de m-bits recouvrant en partie l'ensemble de la séquence.
Le but de ce test est de déterminer si le nombre d'occurences des 2^m motifs de m-bits qui se chevauchent sont approximativement les mêmes que ceux attendue pour une séquence aléatoire.De plus ces motifs sont uniformes.
Cependant si $m = 1$, le test de série est équivalent au test de fréquence.

Le principe qui suit nous montre que les séquences aléatoires passent avec succés ce test.

Principe 4.2.13. Soit U_n une suite de taille n ; ψ_m^2 , ψ_{m-1}^2 et ψ_{m-2}^2 des variables définies par :
$\psi_m^2 = \frac{2^m}{n} \sum_{i_1 \ldots i_m} v_{i_1 \ldots i_m}^2 - n$
$\psi_{m-1}^2 = \frac{2^{m-1}}{n} \sum_{i_1 \ldots i_{m-1}} v_{i_1 \ldots i_{m-1}}^2 - n$
$\psi_{m-2}^2 = \frac{2^{m-2}}{n} \sum_{i_1 \ldots i_{m-2}} v_{i_1 \ldots i_{m-2}}^2 - n$
Soit $\bigtriangledown\psi_m^2(obs)$et $\bigtriangledown^2\psi_m^2(obs)$les mesures de concordance entre les fréquences observées des modèles de m bits et celles attendues.
Ces mesures sont définies respectivement par :
$\bigtriangledown\psi_m^2 = \psi_m^2 - \psi_{m-1}^2$
$\bigtriangledown^2\psi_m^2 = \psi_m^2 - 2\psi_{m-1}^2 + \psi_{m-2}^2$
Alors les p-values :

$$p - value1 = igamc(2^{m-2}, \frac{\bigtriangledown\psi_m^2}{2})$$

$$p - value2 = igamc(2^{m-3}, \frac{\bigtriangledown^2\psi_m^2}{2})$$

sont supérieures à 0.01.

Ainsi pour mieux appuier cela nous avons proposé un algorithme généralisant le principe abordé.

Algorithm 13 Test de Série

Require: La séquence de bits (ϵ)

Require: $n \geq 10^6$

Require: $m < \lfloor \log_2(n) \rfloor - 2$

Ensure: P-value

if $n < 10^6$ **then**

Le test ne peut pas être effectué

else

Calculer la fréquence de tous les blocs de m-bits (v_{i_m})

Calculer la fréquence de tous les blocs de (m-1)-bits $(v_{i_{m-1}})$

Calculer la fréquence de tous les blocs de (m-2)-bits $(v_{i_{m-2}})$

for i_m from 0 to $(2^m - 1)$ **do**

$sum1 \longleftarrow sum1 + \frac{2^m}{n} v_{i_m}^2$

end for

$\psi_m^2 \longleftarrow sum1 - n$

for i_{m-1} from 0 to $(2^{m-1} - 1)$ **do**

$sum2 \longleftarrow sum2 + \frac{(2^{m-1})}{n} v_{i_{m-1}}^2$

end for

$\psi_{m-1}^2 \longleftarrow sum2 - n$

for i_{m-2} from 0 to $(2^{m-2} - 1)$ **do**

$sum3 \longleftarrow sum3 + \frac{(2^{m-2})}{n} v_{i_{m-2}}^2$

end for

$\psi_{m-2}^2 \longleftarrow sum3 - n$

$\bigtriangledown \psi_m^2 \longleftarrow \psi_m^2 - \psi_{m-1}^2$

$\bigtriangledown^2 \psi_m^2 \longleftarrow \psi_m^2 - 2\psi_{m-1}^2 + \psi_{m-2}^2$

$p - value1 \longleftarrow igamc(2^{m-2}, \frac{\bigtriangledown \psi_m^2}{2})$

$p - value2 \longleftarrow igamc(2^{m-3}, \frac{\bigtriangledown^2 \psi_m^2}{2})$

end if

Conclusion $P - value1 \geq 0.01$ and $P - value2 \geq 0.01$

Cependant pour une bonne visibilité des choses nous avons proposé ce tableau en guise d'exemple.

n	séquence	seuil	paramètre	p-value	Etat
100	11001001000011111101101010100010001000010110100011000010001101001100100011001100010100010111000	$\alpha = 0.1$ ou 0.05 ou 0.01	$M = 3$	$p_1 = 0.2052$ $p_2 = 0.2018$	passe
150	100011111111001000010010110010100110001111111011010000100110000111000100110101001001010011100011111111011010000100110000111000100110101001001010111001100101001111101110010111101010011010011111	$\alpha = 0.1$	$M = 3$	$p_1 = 0.0896$ $p_2 = 0.0292$	ne passe pas
150	100011111111001000010010110010100110001111111011010000100110000111000100110101001001010011100011111111011010000100110000111000100110101001001010111001100101001111101110010111101010011010011111	0.01	$M = 3$	$p_1 = 0.0896$ $p_2 = 0.0292$	passe passe
150	100011111111001000010010110010100110001111111011010000100110000111000100110101001001010011100011111111011010000100110000111000100110101001001010111001100101001111101110010111101010011010011111	0.05	$M = 3$	$p_1 = 0.0896$ $p_2 = 0.0292$	passe ne passe pas

Aprés analyse des faits nous avons conclu, qu'avec les seuils de 10% et 5% la séquence de 150 bits ne passent pas le test de Série.

4.2.2.7 Test d'Excursions Aléatoires

Définition 4.2.14. Le test d'excursions aléatoires est un test qui permet de déterminer si le nombre de visites dans un état donné au cours d'un cycle dans une marche aléatoire cumulée dépasse ce à quoi on pourrait s'attendre d'une séquence aléatoire.
Ainsi une marche aléatoire cumulée est définie comme étant la dérivée des sommes partielles après le transfert de la séquence (0,1) à la séquence appropriées (-1,1).
Ce test est en réalité une série de huits tests et conclusions dont une pour chacun des états suivants : -4,-3,-2,-1 et 1,2,3,4.

Le principe illustré ci-dessous nous montre que les séquences aléatoires passent avec succés ce test.

Principe 4.2.14. Soit U_n une suite de taille n dans la base $\{-1, 1\}$; $S = \{S_i\}$ l'ensemble des sommes partielles, et $v_k(x)$ le nombre total de cycles dans lesquels l'état x apparait exactement $k = (0, ..., 5)$ fois parmi tous les cycles.
Soient $J = \sum_{k=0}^{5} v_k(x)$ le nombre total de passages par zéro dans $S' = \{0, S_i, 0\}$
et $\lambda^2(obs)$ la mesure de la qualité du nombre de visites d'état observée et celui attendue au cours d'un cycle.
Cette mesure est définie par :

$$\lambda^2(obs) = \sum_{k=0}^{5} \frac{(v_k(x) - J\pi_k(x))}{J\pi_k(x)}$$

Alors la p-value :

$$p - value = igamc(\frac{5}{2}, \frac{\lambda^2(obs)}{2})$$

est supérieure à 0.01 .

Ainsi pour mieux appuier cela nous avons proposé un algorithme généralisant le principe abordé.

Algorithm 14 Test d'Excursions Aléatoires

Require: La séquence de bits (ϵ) est convertie en des valeurs de -1 et 1
Require: $n \geq 10^6$
Require: $I = \{-4, -3, -2, -1, 1, 2, 3, 4\}$
Require: $x \in I$
Ensure: P-value
  if $n < 10^6$ **then**
    Le test ne peut pas être effectué
  else
    for k from 0 to 5 **do**
      Calculer les fréquences v_k
      $J \longleftarrow J + v_k(x)$
    end for
    for j from 0 to 5 **do**
      $\lambda^2(obs) \longleftarrow \lambda^2(obs) + \frac{(v_j(x) - J\pi_j(x))}{J\pi_j(x)}$
    end for
    $p - value \longleftarrow igamc(\frac{5}{2}, \frac{\lambda^2(obs)}{2})$
  end if
  Conclusion $P - value \geq 0.01$

Cependant pour une bonne visibilité des choses nous avons proposé ce tableau en guise d'exemple.

n	séquence	seuil	paramètre	p-value	Etat
100	1100100100001111110110101010001000 0100001011010001100001000110100110 0010011000110011000101000101110000	$\alpha = 0.1$ ou 0.05 ou 0.01	non	0.2977	passe
150	10001111111100100001001011001010 01 101001110001111111011010001001100 0011 10001000011100010011010100100 10 101110011100101001111101110 00 10011010111101010011010011 1111	$\alpha = 0.1$ ou 0.05 ou 0.01	non	0.1460	passe

Après analyse des faits nous avons conclu qu'avec les trois seuils proposés, les deux séquences passent avec succés le test d'excursions aléatoires .

4.2.2.8 Test de Variante d'Excursion Aléatoire

Définition 4.2.15. Le test de variante d'excursion aléatoire est un test qui permet de déterminer le nombre total de visites d'un état particulier au cours d'une marche aléatoire cumulative.
Le but de ce test est de détecter les écarts par rapport à la distribution du nombre de visites d'une marche aléatoire dans un certain état.
Ce test est en réalité une série de dix-huits tests et conclusions dont une pour chacun des états suivants :-9,-8,...,-1 et 1,2,...,9.

Le principe qui suit nous montre que les séquences aléatoires passent avec succés ce test.

Principe 4.2.15. Soit U_n une suite de taille n dans la base $\{-1,1\}$; J le nombre total de passages par zéro dans S' et ξ le nombre total de visites de l'état donné pendant toute la marche al éatoire.
Soient $\xi(x)$ le nombre total d'occurences(d'apparitions) de cet état x sur tous les J-cycles.
Alors la p-value :

$$p - value = erfc(\frac{\mid \xi(x) - J \mid}{\sqrt{2J(4 \mid x \mid -2)}})$$

est supérieure à 0.01.

Ainsi pour mieux appuier cela nous avons proposé un algorithme généralisant le principe abordé.

Algorithm 15 Test de Variante d'Excursion Aléatoire

Require: La séquence de bits (ϵ) est convertie en des valeurs de -1 et 1
Require: $n \geq 10^6$
Require: $L = \{-9, -8, -7, -6, -5, -4, -3, -2, -1, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
Require: $x \in L$
Ensure: P-value
  if $n < 10^6$ **then**
    Le test ne peut pas être effectué
  else
    Calculer J
    Calculer $\xi(x)$
    $p - value \longleftarrow erfc(\frac{|\xi(x) - J|}{\sqrt{2J(4|x|-2)}})$
  end if
  Conclusion $P - value \geq 0.01$

Cependant pour une bonne visibilité des choses nous avons proposé ce tableau en guise d'exemple.

n	séquence	seuil	paramètre	p-value	Etat
100	11001001000011111101101010100010 0100001011010001100001000110100110 00100110001100110001010001011100 0	$\alpha = 0.1$ ou 0.5 ou 0.01	non	1	passe
150	1000111111110010000100101100101001 101001110001111111011010001001100001 1 10001000001110001001101010010010 10111001110010100111110111000 10011010111101010011010011111 1	$\alpha = 0.1$ ou 0.05 ou 0.01	non	0.7940	passe

Après analyse des faits nous avons conclu qu'avec les trois seuils proposés, les deux séquences passent avec succés le test de variante d'excursions aléatoires .

Conclusion

Les suites récurrentes linéaires ne passent pas la plupart des tests du NIST à cause de la période et de la linéarité de leurs séquences. Mais après analyse des faits nous avons conclu que toute séquence qui passent les trois propriétés de Golomb à la possibilité de passer ces tests .

Chapitre 5

Simulation

Introduction

Dans cette partie du mémoire nous avons choisi de faire la simulation sur 6 tests parmi les 15 tests proposés par le NIST.

5.1 Test de fréquence Monobit

Pour la simulation de ce test nous avons fait une capture d'écran des résultats de l'implémentation proposé mais également une courbe de l'évolution des p-values en fonction de la longueur n.

5.1.1 Implémentation et Courbe

Comme nous l'avons énoncé en haut, nous allons commencé par l'implémentation pour enfin terminer avec la courbe.

5.1.1.1 Implémentation

```
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0
, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0
, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
1, 0, 0]
>>> polynome2(250,1,13)
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0,
0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0
, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1,
1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
1, 1, 1]
```

```
>>> fonction_booléenne(250,1,10,13,17)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
[1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1,
0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0
, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0,
0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0
, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0,
1, 1, 0]
```

```
>>> changebase(250,1,10,13,17)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
[1, -1, -1, -1, -1, 1, 1, 1, 1, -1, 1, 1, 1, -1, 1, -1, 1, 1, -1, 1, -1, 1, 1, -1, 1, 1, 1, 1, 1, -1, -1, -1, 1, -1, -1, 1, 1, -1, -1, 1, -1, -1, -1
, 1, -1, 1, 1, 1, 1, -1, -1, 1, 1, 1, -1, 1, -1, 1, -1, 1, -1, 1, 1, 1, 1, -1, 1, -1, 1, -1, 1, -1, -1, -1, 1, -1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, 1, 1,
-1, -1, -1, -1, -1, -1, 1, -1, 1, 1, 1, 1, 1, -1, 1, 1, -1, 1, -1, -1, 1, -1, -1, 1, 1, -1, 1, 1, 1, -1, -1, 1, -1, -1, -1, 1, 1, 1, 1, 1, -1, 1, -1, -1,
1, 1, 1, 1, 1, 1, 1, 1, -1, -1, 1, -1, -1, 1, -1, 1, -1, 1, -1, -1, -1, -1, 1, 1, 1, 1, -1, -1, -1, 1, 1, -1, -1, 1, -1, -1, -1, -1, -1, 1, -1, -1,
1, -1, 1, -1, 1, -1, -1, 1, 1, 1, 1, 1, 1, -1, -1, -1, 1, -1, -1, -1, -1, -1, 1, 1, -1, -1, 1, -1, -1, -1, 1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, 1, -1, 1, 1, 1, -1, -1, 1, -1, -1, -1, 1, 1, -1, 1, -1, -1, -1, -1, -1, 1, 1, -1, -1, 1, -1, 1, 1, 1, -1, 1, -1, -1, -1, -1, 1, 1, -1]
```

```
>>> somme(250,1,10,13,17)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
-10
```

```
>>> observation(250,1,10,13,17)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
0.6324555320336759
```

```
>>> p_value(250,1,10,13,17)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
0.6547208460185769
```

Et pour terminer voici la courbe du test de fréquence Monobit :

5.1.1.2 Courbe

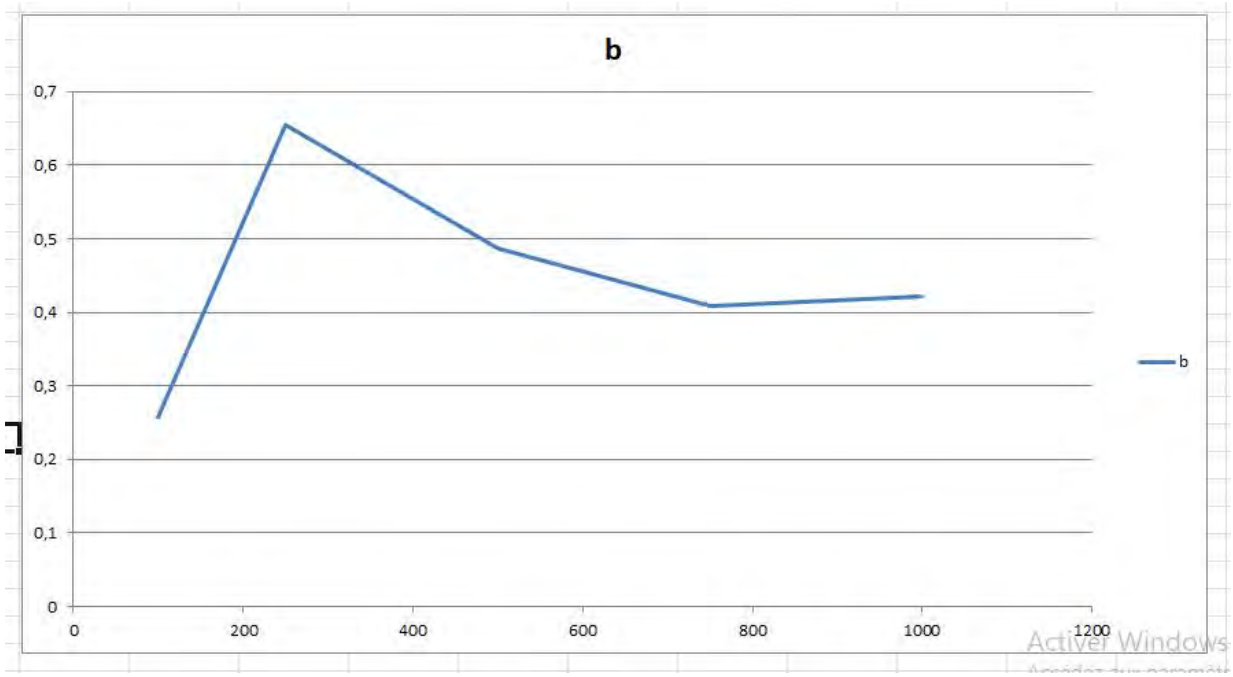


FIGURE 5.1 – Courbe du test de Fréquence Monobit

5.1.1.3 Interprétation

Après analyse de la courbe , nous avons pu constater qu'à partir de n=250 la p-value diminue en fonction de l'augmentation de la taille des séquences.

5.2 Test de fréquence par Bloc

Pour la simulation de celui-ci nous avons aussi fait une capture d'écran des résultats de l'implémentation proposé mais également une courbe de l'évolution des p-values en fonction de la longueur n.

5.2.1 Implémentation et Courbe

Comme nous l'avons fait en haut, nous allons débuté avec l'implémentation pour enfin terminer avec la courbe.

5.2.1.1 Implémentation

```
>>> polynome1(250,1,10)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0
, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0,
0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0
, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
1, 0, 0]
>>> polynome2(250,1,13)
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1,
0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0
, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1,
1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1,
1, 0, 1]
>>> polynome3(250,1,17)
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0,
0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0
, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1,
0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0
, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
1, 1, 1]
```

```
>>> fonction_booléenne(250,1,10,13,17)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1,
0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0
, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0,
0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0
, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0,
1, 1, 0]
```

```
>>> somme1(250,1,10,13,17,75)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
250
1
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1,
0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1]
2
[0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1,
0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0]
3
[1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0,
1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1]
[[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1,
0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1], [0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1,
0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0
, 0, 0], [1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0,
1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1]]
```

```
>>> somme(250,1,10,13,17,75)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
```

```
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
250
1
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1,
0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1]
2
[0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1,
0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0]
3
[1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0,
1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1]
[0.5599999999999995, 0.5066666666666663, 0.39999999999999974]
>>> observation(250,1,10,13,17,75)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
```

```
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
250
1
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1,
0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1]
2
[0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1,
0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0]
3
[1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0,
1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1]
[0.5599999999999995, 0.5066666666666663, 0.39999999999999974]
4.093333333333329
>>> p_value(250,1,10,13,17,75)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
```

```
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
250
1
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1,
0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1]
2
[0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1,
0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0]
3
[1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0,
1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1]
[0.5599999999999995, 0.5066666666666663, 0.39999999999999974]
3.0
mpf('0.2229397657598951')
```

Et pour terminer voici la courbe du test de fréquence par Bloc :

5.2.1.2 Courbe

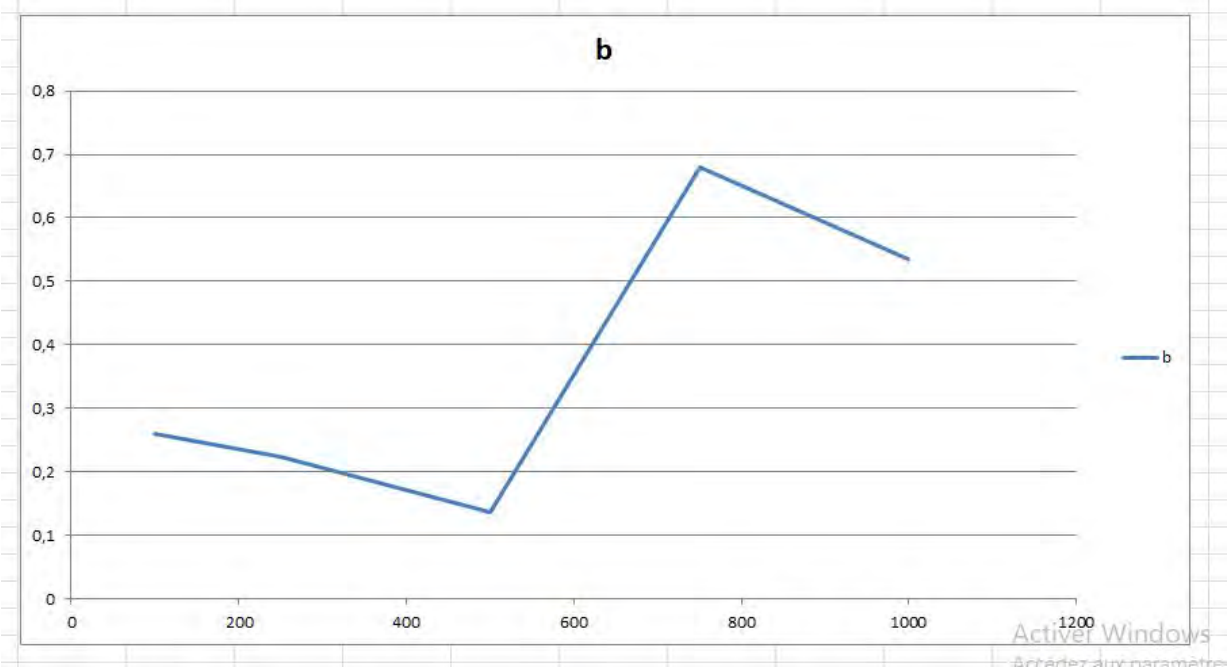


FIGURE 5.2 – Courbe du test de Fréquence par Bloc

5.2.1.3 Interprétation

Après analyse de la courbe , nous avons pu constater que la p-value ne dépend pas de l'augmentation de la taille des séquences ; tant tôt elle augmente , tant tôt elle diminue

5.3 Test de Run

Pour la simulation de ce test la procédure est la même .

5.3.1 Implémentation et Courbe

Comme les autres tests, nous allons débuté par les résultats de l'implémentation et enfin terminer par la courbe.

5.3.1.1 Implémentation

```
>>> polynome1(250,1,10)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0
, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0
, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
1, 0, 0]
>>> polynome2(250,1,13)
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1,
0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0
, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1,
1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1,
1, 0, 1]
>>> polynome3(250,1,17)
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0,
0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0
, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1,
0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0
, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
1, 1, 1]
```

```
>>> fonction_booléenne(250,1,10,13,17)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1,
0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0
, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0,
0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0
, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0,
1, 1, 0]
>>> somm(250,1,10,13,17)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
```



```
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
0.480000000000000037
>>> inégalité(250,1,10,13,17)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
```

```
0000000011010
[0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
0.12649110640673517
le test de fréquence est executable
>>> run(250,1,10,13,17)
test de run
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
[[1], [0, 0, 0, 0], [1, 1, 1, 1], [0], [1, 1, 1], [0], [1], [0], [1, 1], [0], [1], [0], [1, 1], [0], [1, 1, 1, 1, 1], [0, 0, 0], [1], [0, 0], [1, 1]
, [0, 0], [1], [0, 0, 0], [1], [0], [1, 1, 1, 1], [0, 0], [1, 1, 1], [0], [1], [0], [1], [0], [1], [0], [1, 1, 1, 1], [0], [1], [0], [1], [0], [1],
[0, 0, 0], [1], [0, 0], [1], [0], [1], [0], [1], [0], [1, 1, 1], [0, 0, 0, 0, 0, 0, 0], [1], [0], [1, 1, 1, 1], [0], [1, 1], [0], [1], [0, 0], [1], [0,
0], [1, 1], [0], [1, 1, 1], [0, 0], [1], [0, 0, 0], [1, 1, 1, 1, 1], [0], [1], [0, 0], [1, 1, 1, 1, 1, 1, 1, 1, 1], [0, 0], [1], [0, 0], [1], [0], [1], [0
], [1], [0, 0, 0, 0], [1, 1, 1, 1], [0, 0, 0], [1, 1], [0, 0], [1], [0, 0, 0, 0, 0], [1], [0, 0], [1], [0], [1], [0], [1], [0], [1, 1, 1, 1, 1, 1
], [0, 0, 0], [1], [0, 0, 0, 0, 0, 0], [1, 1], [0, 0], [1], [0, 0], [1], [0, 0], [1], [0, 0, 0, 0, 0, 0, 0, 0, 0], [1], [0], [1, 1, 1], [0, 0], [1], [0
, 0, 0], [1, 1], [0], [1], [0, 0, 0, 0, 0], [1, 1], [0, 0], [1], [0], [1, 1, 1], [0], [1], [0, 0, 0, 0], [1, 1], [0]]
```

```
>>> observation(250,1,10,13,17)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
128
>>> p_value(250,1,10,13,17)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
```

```
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
0.685168655801806
```

Et pour terminer voici la courbe du test de Run :

5.3.1.2 Courbe

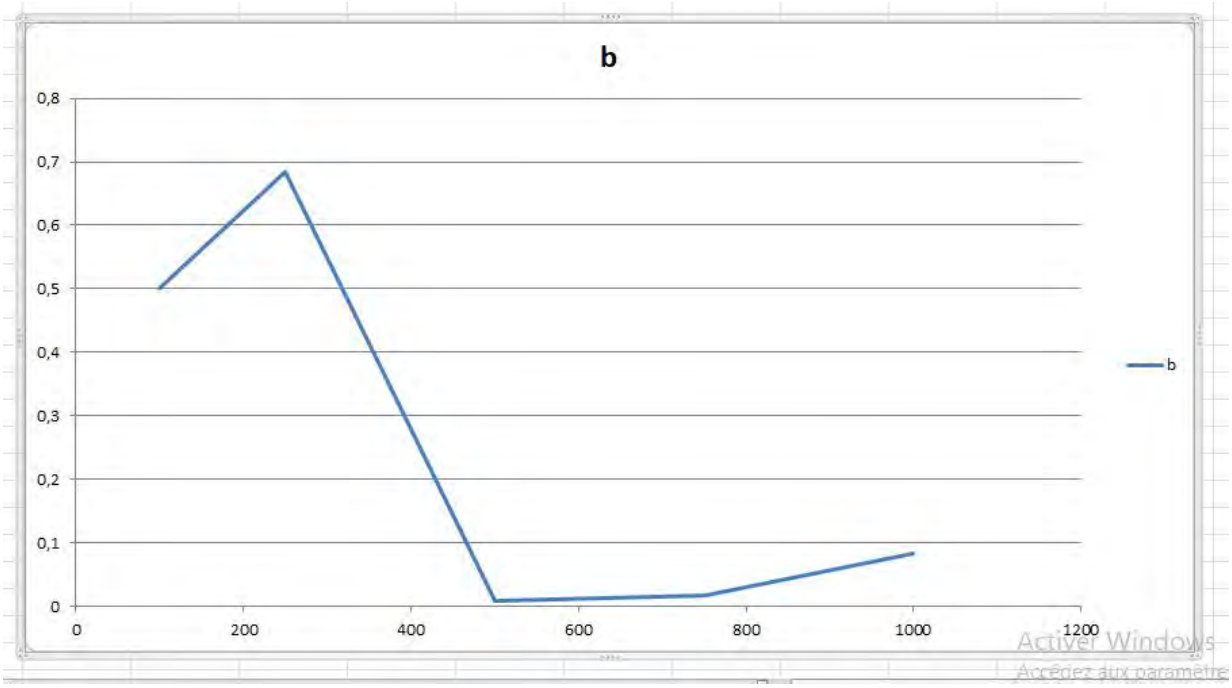


FIGURE 5.3 – Courbe du test de Run

5.3.1.3 Interprétation

Après analyse de la courbe , nous avons pu constater que sur l'intervalle [250; 750] la p-value diminue progressivement en fonction de l'augmentation de la taille des séquences.

5.4 Test de Run par Bloc

Pour la simulation , la procédure est la même.

5.4.1 Implémentation et Courbe

De même que les autres tests, nous allons débuté par les résultats de l'implémentation et enfin terminer par la courbe.

5.4.1.1 Implémentation

```
>>> polynome1(250,1,10)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0
, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0
, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
1, 0, 0]
>>> polynome2(250,1,13)
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1,
0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0
, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0,
1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1,
1, 0, 1]
>>> polynome3(250,1,17)
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
[1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0,
0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0
, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1,
0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0
, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1,
1, 1, 1]
```

```
>>> fonction_booléenne(250,1,10,13,17)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1,
0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0
, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0,
0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0
, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0,
1, 1, 0]
>>> sommebloc(250,1,10,13,17,75)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
```



```
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
3.0
[[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1,
0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1,
0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0
, 0, 0], [1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0,
1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1]]
>>> run(250,1,10,13,17,75)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
donner la condition initiale de la suite1
1100011010
```

```
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
3.0
[[[1], [0, 0, 0, 0], [1, 1, 1, 1], [0], [1, 1, 1], [0], [1], [0], [1, 1], [0], [1], [0], [1, 1], [0], [1, 1, 1, 1, 1], [0, 0, 0], [1], [0, 0], [1, 1
], [0, 0], [1], [0, 0, 0], [1], [0], [1, 1, 1, 1], [0, 0], [1, 1, 1], [0], [1], [0], [1], [0], [1], [0], [1, 1, 1, 1], [0], [1], [0], [1], [0], [1],
[0, 0, 0], [1]], [[0, 0], [1], [0], [1], [0], [1], [0], [1, 1, 1], [0, 0, 0, 0, 0, 0], [1], [0], [1, 1, 1, 1], [0], [1, 1], [0], [1], [0, 0], [1], [
0, 0], [1, 1], [0], [1, 1, 1], [0, 0], [1], [0, 0, 0], [1, 1, 1, 1], [0], [1], [0, 0], [1, 1, 1, 1, 1, 1, 1], [0, 0], [1], [0, 0], [1], [0], [1],
[0], [1], [0, 0, 0, 0]], [[1, 1, 1, 1], [0, 0, 0], [1, 1], [0, 0], [1], [0, 0, 0, 0, 0], [1], [0, 0], [1], [0], [1], [0], [1], [0, 0], [1, 1, 1, 1,
1, 1], [0, 0, 0], [1], [0, 0, 0, 0, 0], [1, 1], [0, 0], [1], [0, 0], [1], [0, 0], [1], [0, 0, 0, 0, 0, 0, 0, 0, 0], [1], [0], [1, 1, 1], [0, 0], [1]
, [0, 0, 0], [1, 1]]]
>>> run1(250,1,10,13,17,75)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
```



```
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
3.0
[[[[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1, 1]], [], [[1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [],
[[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1
]]]]
[[[[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1, 1]], [], [[1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [],
[[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1
]]], [[], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1]], [], [[1]], [], [[1]], [], [[1, 1]], [], [[1, 1,
1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1, 1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], []]
[[[[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1, 1]], [], [[1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [],
[[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1
]]], [[], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1]], [], [[1]], [], [[1]], [], [[1, 1]], [], [[1, 1,
1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1, 1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1, 1]], [],
[[1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1]], [], [[1]],
[], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]]]]
[[[1], [], [4], [], [3], [], [1], [], [2], [], [1], [], [2], [], [5], [], [1], [], [2], [], [1], [], [1], [], [4], [], [3], [], [1], [], [1], [], [1
], [], [4], [], [1], [], [1], [], [1], [], [1]], [[], [1], [], [1], [], [1], [], [3], [], [1], [], [4], [], [2], [], [1], [], [1], [], [2], [], [3],
[], [1], [], [4], [], [1], [], [8], [], [1], [], [1], [], [1], [], [1], []], [[4], [], [2], [], [1], [], [1], [], [1], [], [1], [], [1], [], [6], []
, [1], [], [2], [], [1], [], [1], [], [1], [], [1], [], [3], [], [1], [], [2]]]
>>> maximum1(250,1,10,13,17,75)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
```

```
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
3.0
[[[[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1, 1]], [], [[1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [],
[[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1
]]]]
[[[[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1, 1]], [], [[1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [],
[[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1
]]], [[], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1]], [], [[1]], [], [[1]], [], [[1, 1]], [], [[1, 1,
1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1, 1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], []]
[[[[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1, 1]], [], [[1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [],
[[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1
]]], [[], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1]], [], [[1]], [], [[1]], [], [[1, 1]], [], [[1, 1,
1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1, 1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1, 1]], [],
[], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]]]]
[[5], [8], [6]]
>>> maximum(250,1,10,13,17,75)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
```



```
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
3.0
[[[[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1, 1]], [], [[1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [],
[[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1
]]]]
[[[[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1, 1]], [], [[1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [],
[[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1
]]], [[], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1]], [], [[1]], [], [[1]], [], [[1, 1]], [], [[1, 1,
1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1, 1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1, 1]], [],
[], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]]]]
[5, 8, 6]
>>> fréquence(250,1,10,13,17,75)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
```

```
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
3.0
[[[[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1, 1]], [], [[1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [],
[[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1
]]]]
[[[[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1, 1]], [], [[1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [],
[[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1
]]], [[], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1]], [], [[1]], [], [[1]], [], [[1, 1]], [], [[1, 1, 1,
1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1, 1, 1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], []]
[[[[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1, 1]], [], [[1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [],
[[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1
]]], [[], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1]], [], [[1]], [], [[1]], [], [[1, 1]], [], [[1, 1,
1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1, 1, 1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], []], [[[1, 1, 1, 1]], [],
[[1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1]], [], [[1]],
[], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]]]
5
8
6
([], [], [], [5, 8, 6])
>>> sommefréquence(250,1,10,13,17,75)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
```



```
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
3.0
[[[[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1, 1]], [], [[1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [],
[[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1
]]]]
[[[[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1, 1]], [], [[1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [],
[[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1
]]], [[], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1]], [], [[1]], [], [[1]], [], [[1, 1]], [], [[1, 1,
1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1, 1, 1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], []]
[[[[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1, 1]], [], [[1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [],
[[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1
]]], [[], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1]], [], [[1]], [], [[1]], [], [[1, 1]], [], [[1, 1,
1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1, 1, 1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], []], [[[1, 1, 1, 1]], [],
[[1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1]], [], [[1]],
[], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]]]
5
8
6
[0, 0, 0, 19]
>>> observation(250,1,10,13,17,75)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
```

```
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
3.0
[[[[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1, 1]], [], [[1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [],
[[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1
]]]]
[[[[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1, 1]], [], [[1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [],
[[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1
]]], [[], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1]], [], [[1]], [], [[1]], [], [[1, 1]], [], [[1, 1,
1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1, 1, 1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], []]
[[[[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1, 1]], [], [[1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [],
[[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1, 1, 1, 1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1
]]], [[], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1]], [], [[1]], [], [[1]], [], [[1, 1]], [], [[1, 1,
1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1, 1, 1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1, 1]], [],
[[1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1]], [], [[1]], [], [[1]], [], [[1, 1]], [], [[1,
[], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]]]]
5
8
6
[0, 0, 0, 19]
250
3.0
[mpf('0.64439999999999997'), mpf('1.1016000000000001'), mpf('0.6915'), mpf('604.34027777777783')]
mpf('606.77777777777783')
>>> p_value(250,1,10,13,17,75)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
```



```
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
3.0
[[[[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1, 1]], [], [[1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [],
[[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1
]]]]
[[[[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]], [], [[1]], [], [[1, 1]], [], [[1, 1, 1, 1, 1]], [], [[1]], [], [[1, 1]], [],
[[1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1, 1, 1, 1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1
]]], [[], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1]], [], [[1]], [], [[1]], [], [[1, 1]], [], [[1, 1,
1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1, 1, 1, 1, 1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1, 1]], [],
[[1, 1]], [], [[1]], [], [[1]], [], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1, 1, 1]], [], [[1, 1]], [], [[1]], [], [[1]], [], [[1, 1]], [], [[1,
[], [[1]], [], [[1, 1, 1]], [], [[1]], [], [[1, 1]]]]
5
8
6
[0, 0, 0, 19]
250
3.0
[mpf('0.64439999999999997'), mpf('1.1016000000000001'), mpf('0.6915'), mpf('604.34027777777783')]
mpf('3.0310527772670228e-131')
>>>
```

Et pour terminer voici la courbe du test de Run par bloc :

5.4.1.2 Courbe

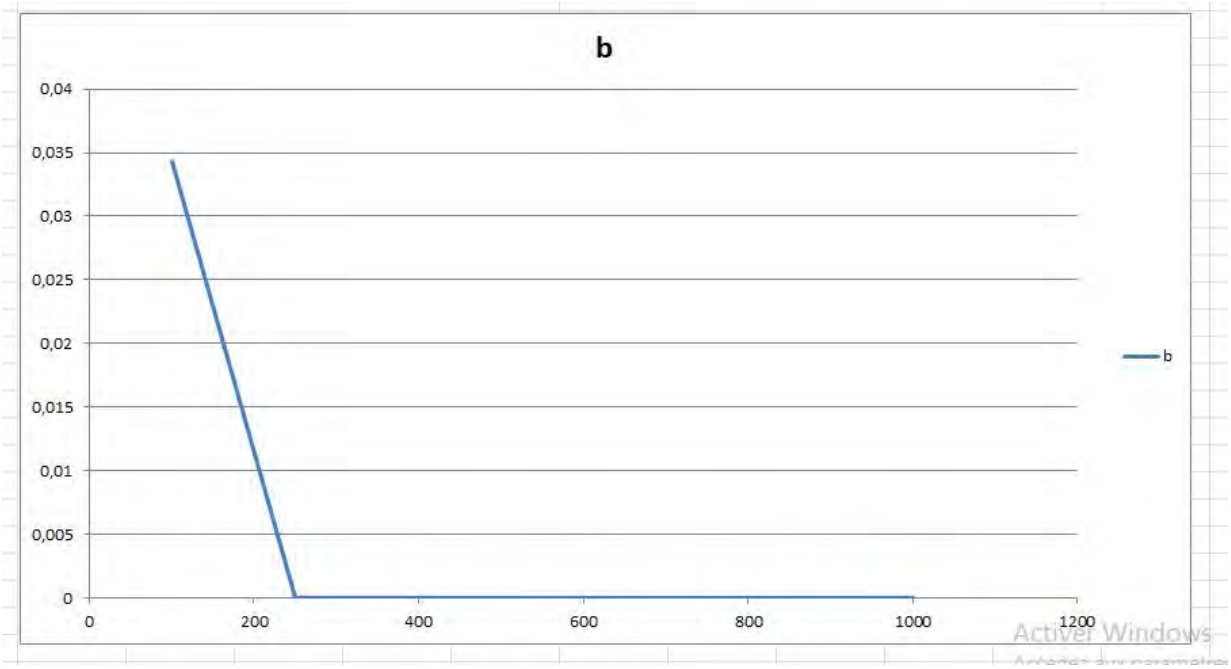


FIGURE 5.4 – Courbe du test de Run par Bloc

5.4.1.3 Interprétation

Après analyse de la courbe , nous avons pu constater qu'à partir de n=250 la valeur de la p-value tend vers 0 et ceci en fonction de l'augmentation de la taille des séquences.

5.5 Test de Rang d'une Matrice Binaire

Pour la simulation de ce test , la procédure est identique aux autres.

5.5.1 Implémentation et Courbe

Comme les autres tests, nous allons d'abord débuter par l'implémentation et enfin terminer par la courbe.

5.5.1.1 Implémentation

```
>>> polynome1(250,1,10)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0
, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0
, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
1, 0, 0]
>>> polynome2(250,1,13)
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1,
0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0
, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1,
1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
1, 0, 1]
>>> polynome3(250,1,17)
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0,
0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0
, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1,
0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0
, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1,
1, 1, 1]
```

```
>>> fonction_booléenne(250,1,10,13,17)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1,
0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0
, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0,
0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0
, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0,
1, 1, 0]
>>> sommebloc(250,1,10,13,17,10)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
```

```
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
[[1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1,
0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0
, 1], [1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0,
0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1
, 0, 0, 1]]
>>> matrice(250,1,10,13,17,10)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
```

```
250
[matrix([[1, 0, 0, 0, 0, 1, 1, 1, 1, 0],
        [1, 1, 1, 0, 1, 0, 1, 1, 0, 1],
        [0, 1, 1, 0, 1, 1, 1, 1, 1, 0],
        [0, 0, 1, 0, 0, 1, 1, 0, 0, 1],
        [0, 0, 0, 1, 0, 1, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 1, 0, 1, 0, 1],
        [0, 1, 1, 1, 1, 0, 1, 0, 1, 0],
        [1, 0, 0, 0, 1, 0, 0, 1, 0, 1],
        [0, 1, 0, 1, 1, 1, 0, 0, 0, 0],
        [0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1]]), matrix([[1, 0, 1, 0, 0, 1, 0, 0, 1, 1],
        [0, 1, 1, 1, 0, 0, 1, 0, 0, 0],
        [1, 1, 1, 1, 0, 1, 0, 0, 1, 1],
        [1, 1, 1, 1, 1, 1, 0, 0, 1, 0],
        [0, 1, 0, 1, 0, 1, 0, 0, 0, 0],
        [1, 1, 1, 1, 0, 0, 0, 1, 1, 0],
        [0, 1, 0, 0, 0, 0, 0, 1, 0, 0],
        [1, 0, 1, 0, 1, 0, 0, 1, 1, 1],
        [1, 1, 1, 0, 0, 0, 1, 0, 0, 0],
        [0, 0, 1, 1, 0, 0, 1, 0, 0, 1]])]
>>> rangmatrice(250,1,10,13,17,10)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
```

```
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
[10, 9]
>>> nbreRmatrice(250,1,10,13,17,10)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
2
(1, 1, 0)
>>> observation(250,1,10,13,17,10)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
```



```
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
2
0.5969529085872576
>>> p_value(250,1,10,13,17,10)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
```

Et pour terminer voici la courbe du test de Rang d'une Matrice :

5.5.1.2 Courbe

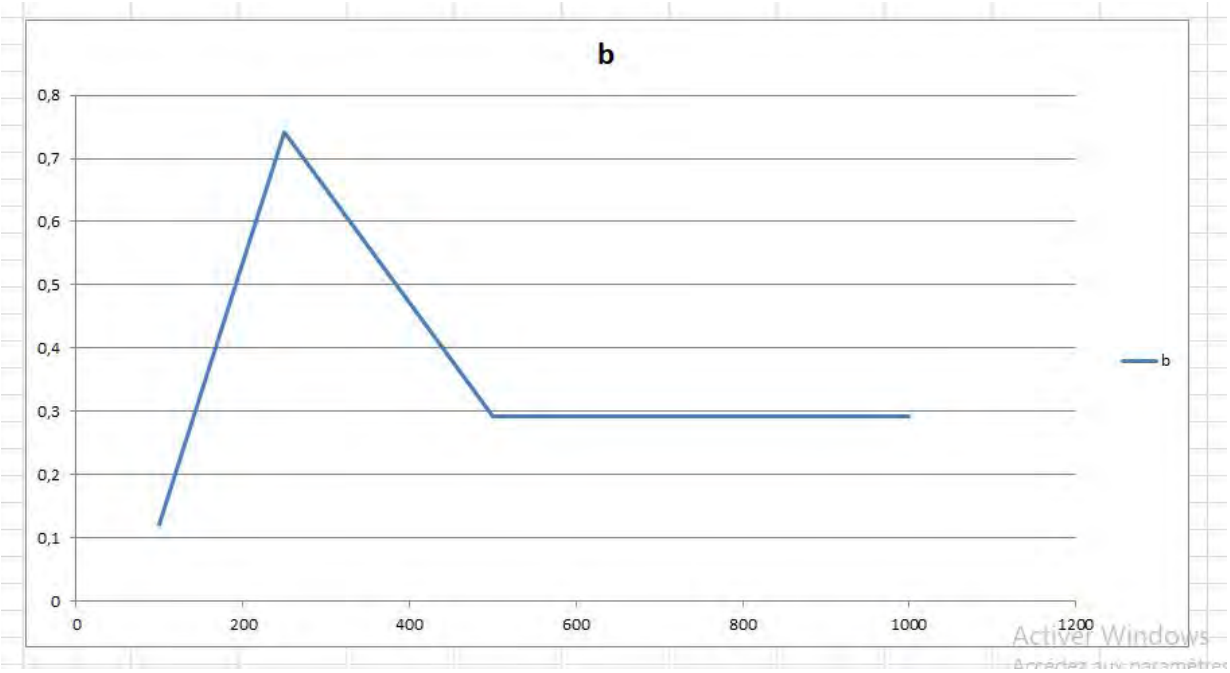


FIGURE 5.5 – Courbe du test de Rang d'une Matrice Binaire

5.5.1.3 Interprétation

Après analyse de la courbe , nous avons pu constater qu'à partir de n=250 la valeur de la p-value diminue jusqu'à se stabiliser à 0.29 et ceci en fonction de l'augmentation de la taille des séquences.

5.6 Test de Somme Cumulative

Pour la simulation du test de somme cumulative , la procédure est la même que les autres.

5.6.1 Implémentation et Courbe

Comme les autres tests, nous allons d'abord débuter par les résultats de l'implémentation et enfin terminer par la courbe.

5.6.1.1 Implémentation

```
>>> polynome1(250,1,10)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0
, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0
, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
1, 0, 0]
>>> polynome2(250,1,13)
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1,
0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0
, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1,
1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1,
1, 0, 1]
>>> polynome3(250,1,17)
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0,
0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0
, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1,
0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0
, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1,
1, 1, 1]
```

```
>>> fonction_booléenne(250,1,10,13,17)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1,
0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0
, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0,
0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0
, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0,
1, 1, 0]
>>>
```

```
>>> changebase(250,1,10,13,17)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
[1, -1, -1, -1, -1, 1, 1, 1, 1, -1, 1, 1, 1, -1, 1, -1, 1, 1, -1, 1, -1, 1, 1, -1, 1, 1, 1, 1, 1, -1, -1, -1, 1, -1, -1, 1, 1, -1, -1, 1, -1, -1, -1
, 1, -1, 1, 1, 1, 1, 1, -1, -1, 1, 1, 1, -1, 1, -1, 1, -1, 1, -1, 1, 1, 1, 1, 1, -1, 1, -1, 1, -1, 1, -1, -1, -1, 1, -1, -1, 1, -1, 1, -1, 1, 1, 1,
-1, -1, -1, -1, -1, -1, 1, -1, 1, 1, 1, 1, -1, 1, 1, -1, 1, -1, -1, 1, -1, -1, 1, 1, -1, 1, 1, 1, -1, -1, 1, -1, -1, -1, 1, 1, 1, 1, -1, 1, -1, -1,
1, 1, 1, 1, 1, 1, 1, 1, -1, -1, 1, -1, -1, 1, -1, 1, -1, 1, -1, -1, -1, -1, -1, 1, 1, 1, 1, -1, -1, -1, 1, 1, -1, -1, 1, -1, -1, -1, -1, -1, 1, -1, -1,
1, -1, 1, -1, 1, -1, -1, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, 1, -1, -1, -1, -1, -1, 1, 1, 1, -1, -1, 1, -1, 1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, 1, -1, 1, 1, 1, -1, -1, 1, -1, -1, -1, 1, 1, -1, 1, -1, -1, -1, -1, -1, 1, 1, -1, -1, 1, -1, 1, 1, 1, -1, 1, -1, -1, -1, -1, 1, 1, -1]
>>> sumCumulée(250,1,10,13,17)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
```



```
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
[1, 0, 1, 2, 3, 2, 1, 0, 1, 0, 1, 2, 3, 2, 3, 2, 3, 4, 3, 4, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8, 7, 6, 7, 6, 5, 6, 7, 6, 5, 6, 5, 4, 3, 4, 3, 4, 5, 6, 7,
6, 5, 6, 7, 8, 7, 8, 7, 8, 7, 8, 7, 8, 9, 10, 11, 10, 11, 10, 11, 10, 11, 10, 9, 8, 9, 8, 7, 8, 7, 8, 7, 8, 7, 8, 9, 10, 9, 8, 7, 6, 5, 4, 5, 4, 5,
6, 7, 8, 7, 8, 9, 8, 9, 8, 7, 8, 7, 6, 7, 8, 7, 8, 9, 10, 9, 8, 9, 8, 7, 6, 7, 8, 9, 10, 9, 10, 9, 8, 9, 10, 11, 12, 13, 14, 15, 16, 15, 14, 15, 14,
13, 14, 13, 14, 13, 14, 13, 12, 11, 10, 11, 12, 13, 14, 13, 12, 11, 12, 13, 12, 11, 12, 11, 10, 9, 8, 7, 8, 7, 6, 7, 6, 7, 6, 7, 6, 5, 6, 7, 8, 9, 1
0, 11, 10, 9, 8, 9, 8, 7, 6, 5, 4, 5, 6, 5, 4, 5, 4, 3, 4, 3, 2, 3, 2, 1, 0, 1, 2, 3, 4, 5, 6, 5, 6, 5, 4, 3, 4, 5, 4, 5, 6, 7, 6, 5, 6, 5, 6, 7, 8,
9, 10, 9, 8, 9, 10, 9, 10, 9, 8, 7, 8, 7, 8, 9, 10, 11, 10, 9, 10]
16
>>> p_value(250,1,10,13,17)
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
250
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
```

```
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
250
donner la condition initiale de la suite1
1100011010
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0]
donner le polynome minimal de la suite1
0000001001
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
donner la condition initiale de la suite2
0011100010110
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite2
0000000011010
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
donner la condition initiale de la suite3
10000111100010110
[1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
donner le polynome minimal de la suite3
00000000000001001
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
250
[1, 0, 1, 2, 3, 2, 1, 0, 1, 0, 1, 2, 3, 2, 3, 2, 3, 4, 3, 4, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8, 7, 6, 7, 6, 5, 6, 7, 6, 5, 6, 5, 4, 3, 4, 3, 4, 5, 6, 7,
6, 5, 6, 7, 8, 7, 8, 7, 8, 7, 8, 7, 8, 9, 10, 11, 10, 11, 10, 11, 10, 11, 10, 9, 8, 9, 8, 7, 8, 7, 8, 7, 8, 7, 8, 9, 10, 9, 8, 7, 6, 5, 4, 5, 4, 5,
6, 7, 8, 7, 8, 9, 8, 9, 8, 7, 8, 7, 6, 7, 8, 7, 8, 9, 10, 9, 8, 9, 8, 7, 6, 7, 8, 9, 10, 9, 10, 9, 8, 9, 10, 11, 12, 13, 14, 15, 16, 15, 14, 15, 14,
13, 14, 13, 14, 13, 14, 13, 12, 11, 10, 11, 12, 13, 14, 13, 12, 11, 12, 13, 12, 11, 12, 11, 10, 9, 8, 7, 8, 7, 6, 7, 6, 7, 6, 7, 6, 5, 6, 7, 8, 9, 1
0, 11, 10, 9, 8, 9, 8, 7, 6, 5, 4, 5, 6, 5, 4, 5, 4, 3, 4, 3, 2, 3, 2, 1, 0, 1, 2, 3, 4, 5, 6, 5, 6, 5, 4, 3, 4, 5, 4, 5, 6, 7, 6, 5, 6, 5, 6, 7, 8,
9, 10, 9, 8, 9, 10, 9, 10, 9, 8, 7, 8, 7, 8, 9, 10, 11, 10, 9, 10]
16
0.623144146628817
```


Et pour terminer voici la courbe du test de Somme Cumulative :

5.6.1.2 Courbe

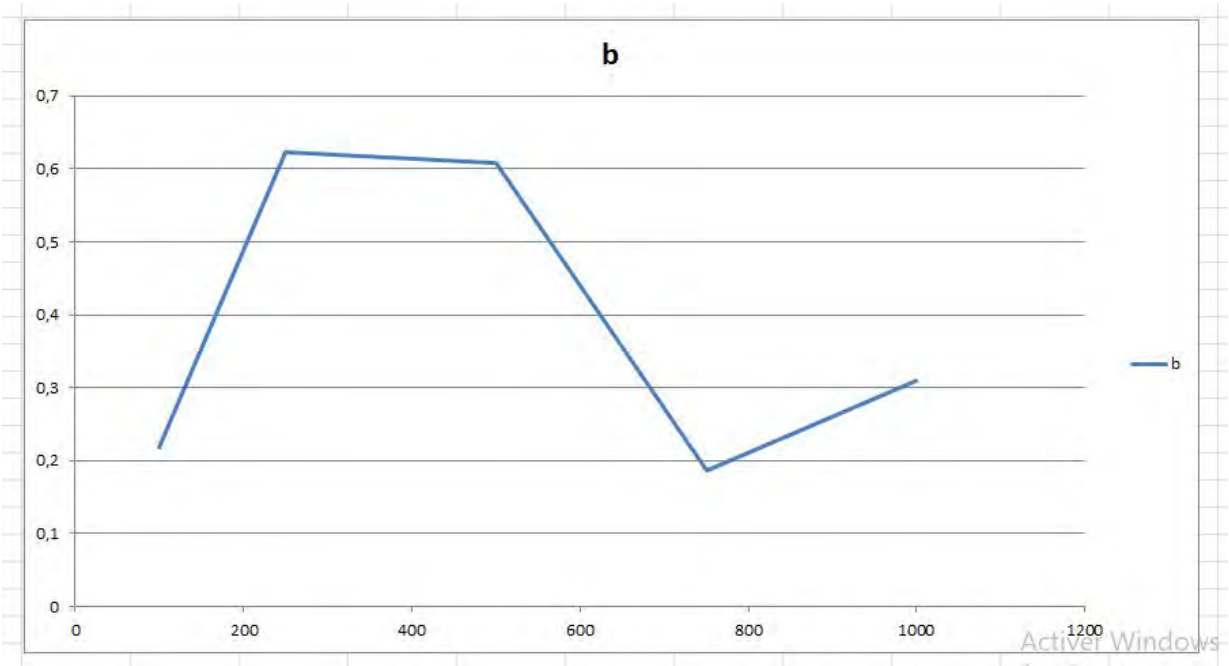


FIGURE 5.6 – Courbe du test de Somme Cumulative

5.6.1.3 Interprétation

Après analyse de la courbe , nous avons pu constater que sur l'intervalle [250; 750] la p-value diminue progressivement en fonction de l'augmentation de la taille des séquences.

Conclusion

La simulation faite sur les six tests du NIST, nous a permi de conclure que la diminution ou l'augmentation de la p-value ne dépend pas uniquement de la longueur de la séquence considérée.

Chapitre 6

Conclusion Générale et Perspectives

Conclusion Générale

Nous voici arrivé au terme de ce mémoire qui a consisté à travers les tests du NIST à vérifier l'aspect aléatoire des séquences.
En effet notre objectif était de proposer des méthodes qui serviraient à améliorer la qualité et la performance des générateurs pseudo-aléatoires et à travers ces méthodes il fallait aussi répondre à la problématique de ces générateurs.
Ce travail composé de chapitres, sections, sous-sections et paragraphes comme détaillé dans le développement à porter sur l'étude des tests du NIST pour les séquences pseudo-aléatoires.
Mais comme tout travail scientifique quelques imperfections ne manqueraient pas.Raison pour laquelle nous mettons à la disposition de toute personne capable d'enrechir , de compléter ou de perfectionner davantage ce document à l'occasion des prochains recherches avec le concours de notre vie de chercheurs.

Perspectives

✓ Mettre en Oeuvre une Plateforme en Python pour les Tests du NIST
✓ Redéfinir la valeur critique du NIST à l'ordre de $(1-\epsilon)\%$
✓ Reprendre la structure algorithmique des suites récurrentes linéaires sur un anneau
✓ La possibilité de mettre en place de nouveaux tests
✓ Proposition de nouveaux types de générateurs

BIBLIOGRAPHIE :

[1] O. Diankha,Suites Recurrentes Lineaires sur les corps finis : Théorie et Applications, Afrika Matematica, serie 3, Vol. 18, 2007, pp.46-60.

[2] O. Diankha, C. B. Deme., Linear recurrent sequence over finite field and their applications in cryptogrphy, JP Journal of Algebra, Number Theory and Application,Vol.22, Number 2, 2011, pp 205-221.

[3] O. Diankha and C. B. Deme, Connection between classification and the minimal polynomial of the multiplexed sequence,Journal of Mathematical Sciences (FJMS),2012.

[4] J. Berard. Fiche 1 - Nombres pseudo-aléatoires. ISTIL, 2005. disponible à http ://math. univ-lyon1.fr/˜jberard/genunif-www.pdf.

[5] U.M. Maurer. A universal statistical test for random bit generators. Journal of Cryptology, 5(2) : 89-105, 1992.

[6] D. H. Lehmer. Mathematical methods in large-scale computing units. Proceedings of a Second Symposium on Large-Scale Digital Calculating Machinery, 1949, pp. 141–146. Harvard University Press, Cambridge, Mass., 1951.

[7] A. Rukhin, J. Soto, J. Nechvatal, J. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo. A statistical test suite for random and pseudo random number generators for cryptographic applications, nist special publication 800-22. Online. Available at : http : //csrc.nist.gov/,2014.

[8] Boyan Valtchanov. Générateurs de suites binaires vraiment aléatoires : modélisation et implantation dans des cibles FPGA. PhD thesis, Université Jean Monnet de Saint-Etienne,2010.

[9] Michal Varchola and Milos Drutarovsky. New high entropy element for fpga based true random number generators. In Stefan Mangard and François-Xavier Standaert,editors, CHES, volume 6225 of Lecture Notes in Computer Science, pages 351–365. Springer, 2010.

[10] B. Valtchanov, V. Fischer, and A. Aubert. Enhanced trng based on the coherent sampling. In Signals, Circuits and Systems (SCS), 2009 3rd International Conference on, pages 1–6, Nov 2009.

[11] John Von Neumann. Various Techniques Used in Connection with Random Digits. J. Res. Nat. Bur. Stand., 12 :36–38, 1951.

[12] I. NIVEN : Formal Power Series, The American Mathematical Monthly, Vol.76, numero8 (Octo.1969), pp.871-889.

[13] J-P. RAMIS, A. WARUSFEL, A. CONNES : Mathématiques Tout-en-un pour la Licence, Niveau L2 : Cours complets avec applications et 760 exercices corrigés, Dunod, 2007.

[14] J-P. RAMIS, A. WARUSFEL : Cours de Mathématiques pures et appliquées – Algèbre et géométrie, Tome 1, De Boeck, 2010.

[15] M. Abramowitz and I. Stegun, Handbook of Mathematical Functions, Applied Mathematics Series. Vol. 55, Washington : National Bureau of Standards, 1964 ; reprinted 1968 by Dover

Publications, New York.
[16] T. Cormen, C. Leiserson, and R. Rivest, Introduction to Algorithms. Cambridge, MA : The MIT Press, 1990.
[17] Andrew Klapper : Feedback with Carry Shift Registers over Finite Fields (extended abstract). FSE 1994 : 170-178.
[18] A. Marjane, B. Allailou, Vectorial Conception of FCSR, In C. Carlet and A. Pott (Eds.) : SETA 2010, LNCS 6338, pp. 240-252, (2010).
[19] Gustafson et al., "A computer package for measuring strength of encryption algorithms," Journal of Computers and Security. Vol. 13, No. 8, 1994, pp. 687-697.
[20] J. L. MASSEY, Shift-register synthesis and BCH decoding, IEEE Transactions in Information Theory, Vol. IT-15, pp. 122-127, 1969.
[21] M. Goresky and A. Klapper, Fibonacci and galois representations of feedback-withcarry shift registers, IEEE Transactions on Information Theory, 48(11), pp.2826- 2836, (2002). [22] Andrew Klapper : Algebraic Feedback Shift Registers Based on Function Fields. SETA 2004 : 282-297.
[23] Andrew Klapper : Distributional properties of d-FCSR sequences. J. Complexity 20(2-3) : 305-317 (2004).
[24] A. Menezes, et al., Handbook of Applied Cryptography. CRC Press, Inc., 1997. See http ://www.c
[25] Mark Goresky, Andrew Klapper : Periodicity and Correlation Properties of d-FCSR Sequences. Des. Codes Cryptography 33(2) : 123-148 (2004).
[26] S.W. Golomb, Shift register sequences, Aegean Park Press, Laguna Hills, California, 1982.
[27] Guang Gong : Sequence Analysis, Lecture Notes for CO739x, Winter 1999.
[28]W. Press, S. Teukolsky, W. Vetterling, Numerical Recipes in C : The Art of Scientific Computing, 2nd Edition. Cambridge University Press, January 1993.
[29] G. Marsaglia, DIEHARD Statistical Tests : http ://www.stat.fsu.edu/pub/diehard/.
[30] T. Ritter,"Randomness Tests and Related Topics,http ://www.ciphersbyritter.com/RES/RANDTE
[31] American National Standards Institute : Financial Institution Key Management (Wholesale),American Bankers Association, ANSI X9.17 - 1985 (Reaffirmed 1991).
[32] R. Rolland, Institut de Mathématiques de Luminy, Campus de Luminy, Case 907,13288 MARSEILLE Cedex 9 •E-mail :robert.rolland@acrypta.fr Url : http ://www.acrypta.fr/ rolland

RESUME :

Puisque l'on ne peut pas prouver mathématiquement qu'un générateur de nombres est pseudo-aléatoire, nous avons préféré orienter nos recherches sur les batteries de tests statistiques pour vérifier l'aspect aléatoire des séquences.

Parmi ces batteries nous avons choisi de travailler avec celle du NIST dont la validation de l'hypothèse nulle est basée sur une p-value uniformément réparties dans l'intervalle $]0.01; 1]$.

Ces tests seront appliqués aux générateurs pseudo-aléatoires, mais particulièrement aux LFSRs dont la simplicité et la fiabilité à un temps réel nous a poussé à les utiliser pour la construction de séquences pseudo-aléatoires.

Néanmoins la sortie d'un tel générateur peut ne pas être utilisée dans de nombreuses applications cryptographiques.

En effet de part leur complexité les LFSRs peuvent masquer leur niveau de sécurité.

Ainsi pour faire face à ce problème nous avons décidé d'utiliser les tests du NIST pour déterminer si les LFSRs conviennent ou non à une application cryptographique.

Cependant aucune batterie de tests statistiques ne peut certifier la sécurité d'un générateur.

Mots Clés :

Générateurs pseudo-aléatoires ,Hypothèse nulle, Régistre à décalage à rétroaction linéaire , Séquences pseudo-aléatoires , Complexité ,Tests du NIST , Sécurité d'un générateur

ABSTRACT :

Since it can't be proven mathematically that a number generator is pseudo-random, we chose to focus our research on statistical test batteries to verify the randomness of the sequences.

Among these batteries we chose to work with the NIST battery whose null hypothesis validation is based on a p-value uniformly distributed in the interval $]0.01; 1]$.

These tests will be applied to pseudo-random generators, but particularly to LFSRs, whose simplicity and reliability in real time led us to use them for the construction of pseudo-random sequences.

However, the output of such a generator cannot be used in many cryptographic applications.

Indeed, due to their complexity, the LFSRs can mask their level of security.

To address this problem, we decided to use NIST tests to determine whether or not LFSRs are suitable for cryptographic applications.

However, there is no statistical test battery that can certify the security of a generator...

Keywords :

Pseudo-random generators , Null hypothesis, Linear feedback shift register , Pseudo-random sequences , Complexity , NIST tests , Generator safety