Générateurs Uniformes

$\mathcal{S}$, espace d'états fini;
$f : \mathcal{S} \to \mathcal{S}$, fonction de transition;

s_0, germe (état initial);
$s_n = f(s_{n-1})$

Générateurs Uniformes

$\mathcal{S}$, espace d'états fini;
$f : \mathcal{S} \to \mathcal{S}$, fonction de transition;
$\mathcal{U}$, espace des valeurs de sortie;
$g : \mathcal{S} \to \mathcal{U}$, fonction de sortie.

s_0, germe (état initial);
$s_n = f(s_{n-1})$

$u_n = g(s_n)$

Générateurs Uniformes

$\mathcal{S}$, espace d'états fini;
$f : \mathcal{S} \to \mathcal{S}$, fonction de transition;
$\mathcal{U}$, espace des valeurs de sortie;
$g : \mathcal{S} \to \mathcal{U}$, fonction de sortie.

s_0, germe (état initial);
$s_n = f(s_{n-1})$

$u_n = g(s_n)$

s_0

Générateurs Uniformes

$\mathcal{S}$, espace d'états fini;
$f : \mathcal{S} \to \mathcal{S}$, fonction de transition;
$\mathcal{U}$, espace des valeurs de sortie;
$g : \mathcal{S} \to \mathcal{U}$, fonction de sortie.

s_0, germe (état initial);
$s_n = f(s_{n-1})$

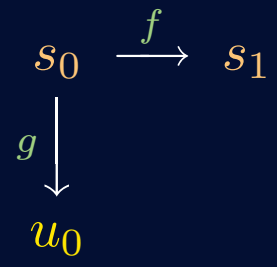
$u_n = g(s_n)$

Générateurs Uniformes

$\mathcal{S}$, espace d'états fini;
$f : \mathcal{S} \to \mathcal{S}$, fonction de transition;
$\mathcal{U}$, espace des valeurs de sortie;
$g : \mathcal{S} \to \mathcal{U}$, fonction de sortie.

s_0, germe (état initial);
$s_n = f(s_{n-1})$

$u_n = g(s_n)$

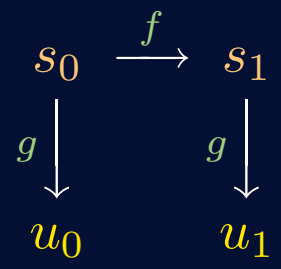


Générateurs Uniformes

$\mathcal{S}$, espace d'états fini;
$f : \mathcal{S} \to \mathcal{S}$, fonction de transition;
$\mathcal{U}$, espace des valeurs de sortie;
$g : \mathcal{S} \to \mathcal{U}$, fonction de sortie.

s_0, germe (état initial);
$s_n = f(s_{n-1})$

$u_n = g(s_n)$

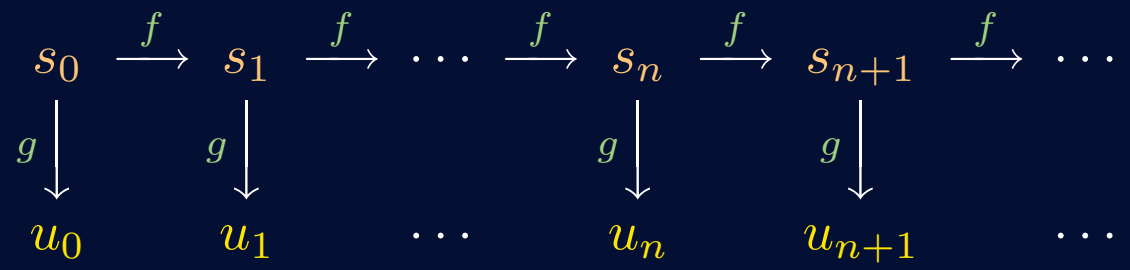


Générateurs Uniformes

$\mathcal{S}$, espace d'états fini;
$f : \mathcal{S} \to \mathcal{S}$, fonction de transition;
$\mathcal{U}$, espace des valeurs de sortie;
$g : \mathcal{S} \to \mathcal{U}$, fonction de sortie.

s_0, germe (état initial);
$s_n = f(s_{n-1})$

$u_n = g(s_n)$

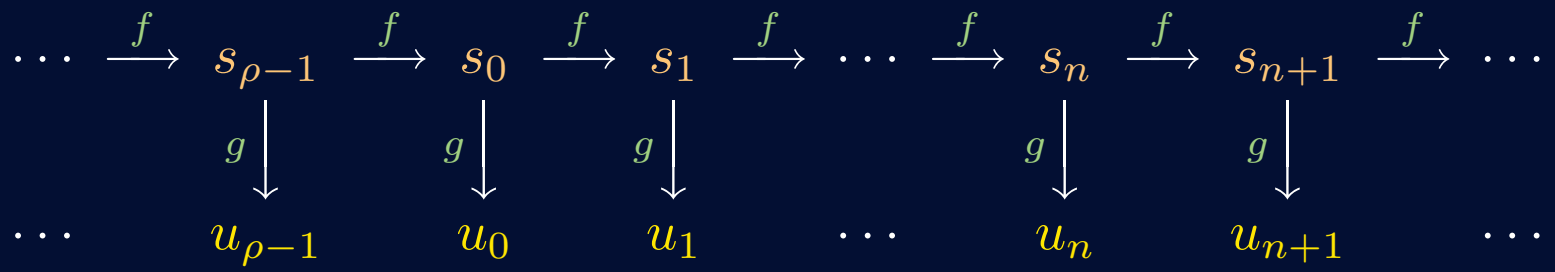


Générateurs Uniformes

$\mathcal{S}$, espace d'états fini;
$f : \mathcal{S} \to \mathcal{S}$, fonction de transition;
$\mathcal{U}$, espace des valeurs de sortie;
$g : \mathcal{S} \to \mathcal{U}$, fonction de sortie.

s_0, germe (état initial);
$s_n = f(s_{n-1})$

$u_n = g(s_n)$

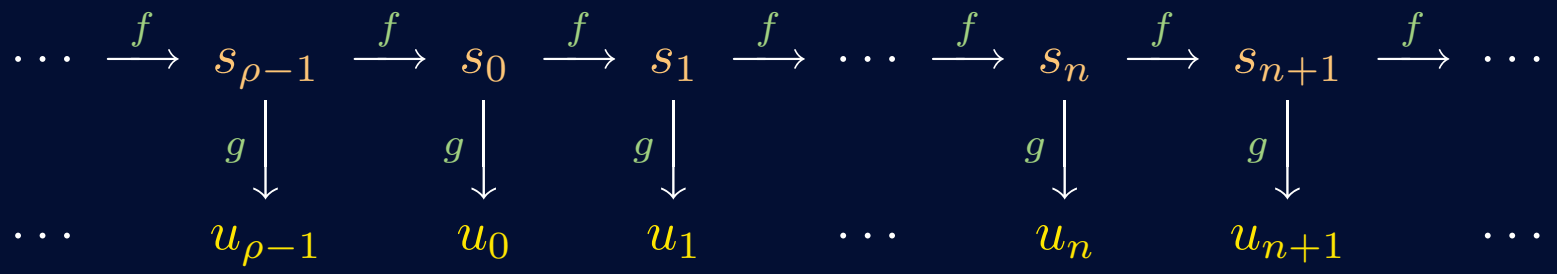


Générateurs Uniformes

$\mathcal{S}$, espace d'états fini;
$f : \mathcal{S} \to \mathcal{S}$, fonction de transition;
$\mathcal{U}$, espace des valeurs de sortie;
$g : \mathcal{S} \to \mathcal{U}$, fonction de sortie.

s_0, germe (état initial);
$s_n = f(s_{n-1})$

$u_n = g(s_n)$



Période: $\rho \leq \mathrm{card}(\mathcal{S})$. $s_{n+\rho} = s_n \quad \forall n \geq \tau$.

On supposera que $\tau = 0$.

Objectif: en observant seulement $(u_0, u_1, \ldots)$, il doit être difficile de distinguer cette suite d'une suite de v.a. i.i.d. uniformes sur $\mathcal{U}$.

Objectif: en observant seulement $(u_0, u_1, \ldots)$, il doit être difficile de distinguer cette suite d'une suite de v.a. i.i.d. uniformes sur $\mathcal{U}$.

La loi uniforme sur $[0,1]^t$:
Choisir s_0 au hasard correspond à choisir un point au hasard dans
$\Psi_t = \{\mathbf{u} = (u_0, \ldots, u_{t-1}) = (g(s_0), \ldots, g(s_{t-1})),\ s_0 \in \mathcal{S}\}$,
qui peut être interprété comme espace échantillonnnal, approximation de $[0,1]^t$.

Objectif: en observant seulement $(u_0, u_1, \ldots)$, il doit être difficile de distinguer cette suite d'une suite de v.a. i.i.d. uniformes sur $\mathcal{U}$.

La loi uniforme sur $[0,1]^t$:
Choisir s_0 au hasard correspond à choisir un point au hasard dans
$\Psi_t = \{\mathbf{u} = (u_0, \ldots, u_{t-1}) = (g(s_0), \ldots, g(s_{t-1})),\ s_0 \in \mathcal{S}\}$,
qui peut être interprété comme espace échantillonnnal, approximation de $[0,1]^t$.

Critère: Ψ_t doit recouvrir $[0,1]^t$ très uniformément pour "tout" t.

Objectif: en observant seulement $(u_0, u_1, \ldots)$, il doit être difficile de distinguer cette suite d'une suite de v.a. i.i.d. uniformes sur $\mathcal{U}$.

La loi uniforme sur $[0,1]^t$:
Choisir s_0 au hasard correspond à choisir un point au hasard dans
$\Psi_t = \{\mathbf{u} = (u_0, \ldots, u_{t-1}) = (g(s_0), \ldots, g(s_{t-1})),\ s_0 \in \mathcal{S}\}$,
qui peut être interprété comme espace échantillonnnal, approximation de $[0,1]^t$.

Critère: Ψ_t doit recouvrir $[0,1]^t$ très uniformément pour "tout" t.

Généralisation: mesurer l'uniformité de $\Psi_I = \{(u_{i_1}, \ldots, u_{i_t}) \mid s_0 \in \mathcal{S}\}$
pour une classe choisie d'ensembles d'indices de forme $I = \{i_1, i_2, \cdots, i_t\}$.

GVAs à sous-suites multiples

Dans un contexte de programmation par objets, on veut pouvoir instancier des GVAs à volonté et les faire évoluer en parallèle.

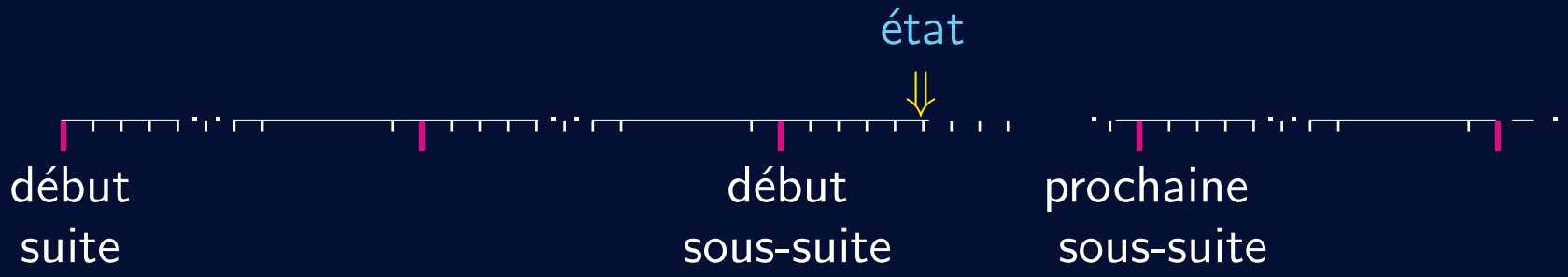
GVAs à sous-suites multiples

Dans un contexte de programmation par objets, on veut pouvoir instancier des GVAs à volonté et les faire évoluer en parallèle.
Utile aussi de pouvoir partitionner ces suites (ou “streams”) en sous-suites.

GVAs à sous-suites multiples

Dans un contexte de programmation par objets, on veut pouvoir instancier des GVAs à volonté et les faire évoluer en parallèle.
Utile aussi de pouvoir partitionner ces suites (ou “streams”) en sous-suites.

Une instance:



Interface Java

```
public interface RandomStream {
   public void resetStartStream ();
```

Réinitialise la suite à son état initial.

```
   public void resetStartSubstream ();
```

Réinitialise la suite au début de sa sous-suite courante.

```
   public void resetNextSubstream ();
```

Réinitialise la suite au début de sa prochaine sous-suite.

```
   public double nextDouble ();
```

Retourne une v.a. $U(0,1)$ de cette suite et avance d'un pas.

```
   public int nextInt (int i, int j);
```

Retourne une v.a. uniforme sur $\{i, i+1, \ldots, j-1\}$.

```
}
```

Générateur récursif multiple (MRG)

$$x_n = (a_1 x_{n-1} + \cdots + a_k x_{n-k}) \bmod m, \qquad u_n = x_n/m.$$

Générateur récursif multiple (MRG)

$$x_n = (a_1 x_{n-1} + \cdots + a_k x_{n-k}) \bmod m, \qquad u_n = x_n/m.$$

En pratique, on prendra plutôt $u_n = (x_n + 1)/(m + 1)$,
ou encore $u_n = x_n/(m + 1)$ si $x_n > 0$ et $u_n = m/(m + 1)$ sinon.
Mais la structure reste essentiellement la même.

Générateur récursif multiple (MRG)

$$x_n = (a_1 x_{n-1} + \cdots + a_k x_{n-k}) \bmod m, \qquad u_n = x_n/m.$$

En pratique, on prendra plutôt $u_n = (x_n + 1)/(m + 1)$,
ou encore $u_n = x_n/(m + 1)$ si $x_n > 0$ et $u_n = m/(m + 1)$ sinon.
Mais la structure reste essentiellement la même.

Si $k = 1$: générateur à congruence linéaire (GCL) classique.

Générateur récursif multiple (MRG)

$$x_n = (a_1 x_{n-1} + \cdots + a_k x_{n-k}) \bmod m, \qquad u_n = x_n/m.$$

En pratique, on prendra plutôt $u_n = (x_n + 1)/(m + 1)$,
ou encore $u_n = x_n/(m + 1)$ si $x_n > 0$ et $u_n = m/(m + 1)$ sinon.
Mais la structure reste essentiellement la même.

Si $k = 1$: générateur à congruence linéaire (GCL) classique.

État à l'étape n: $s_n = \mathbf{x}_n = (x_{n-k+1}, \ldots, x_n)^{\mathsf{t}}$.
Espace d'états: $\mathbb{Z}_m^k = \{0, 1, \ldots, m-1\}^k$, de cardinalité m^k.

$$\mathbf{x}_n = \mathbf{A}\mathbf{x}_{n-1} \bmod m = \begin{pmatrix} 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ a_k & a_{k-1} & \cdots & a_1 \end{pmatrix} \mathbf{x}_{n-1} \bmod m.$$

Période max. $\rho = m^k - 1$, possible si m est premier.

Polynôme caractéristique:

$$P(z) = z^k - a_1 z^{k-1} - \cdots - a_k = -\sum_{j=0}^{k} a_j z^{k-j},$$

où $a_0 = -1$.

Polynôme caractéristique:

$$P(z) = z^k - a_1 z^{k-1} - \cdots - a_k = -\sum_{j=0}^{k} a_j z^{k-j},$$

où $a_0 = -1$.

Il est utile de représenter cette récurrence dans trois espaces différents:

(i) l'espace $\mathbb{Z}_m^k$ des vecteurs ayant k coordonnées dans $\mathbb{Z}_m = \{0, \ldots, m-1\}$,

(ii) l'espace $\mathbb{Z}_m[z]/(P)$ des polynômes de degré $< k$ à coefficients dans $\mathbb{Z}_m$ (i.e., les polynômes réduits modulo $P(z)$),

(iii) l'espace $\mathcal{L}(P)$ des séries formelles de Laurent de la forme $\tilde{s}(z) = \sum_{j=1}^{\infty} c_j z^{-j}$, où les coefficients c_j sont dans $\mathbb{Z}_m$ et satisfont $c_j = (a_1 c_{j-1} + \cdots + a_k c_{j-k}) \bmod m$ pour tout $j > k$.

Bijections entre ces espaces.

À $\mathbf{x}_n = (x_{n-k+1}, \ldots, x_n)^{\mathsf{t}}$ on associe la série

$$\tilde{s}_n(z) = \sum_{j=1}^{\infty} x_{n-k+j} z^{-j}$$

où $x_{n+1}, x_{n+2}, \ldots$ sont déterminés par la récurrence du MRG. Cette série est la fonction génératrice de $\{x_{n-k+j},\, j \geq 1\}$. On associe aussi de polynôme

$$p_n(z) = P(z)\tilde{s}_n(z) \bmod m,$$

vu comme un cas particulier de série formelle.

Proposition. Ces applications $\mathbf{x}_n \to \tilde{s}_n(z) \to p_n(z)$ sont des bijections entre $\mathbb{Z}_m^k$, $\mathcal{L}(P)$ et $\mathbb{Z}_m[z]/(P)$. De plus, l'application correspondante $\mathbf{x}_n \to p_n(z)$ satisfait

$$p_n(z) = \sum_{j=1}^{k} c_{n,j} z^{k-j}$$

où

$$\begin{pmatrix} c_{n,1} \\ c_{n,2} \\ \vdots \\ c_{n,k} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ -a_1 & 1 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ -a_{k-1} & \dots & -a_1 & 1 \end{pmatrix} \begin{pmatrix} x_{n-k+1} \\ x_{n-k+2} \\ \vdots \\ x_n \end{pmatrix} \mod m.$$

En multipliant $\tilde{s}_{n-1}(z)$ par z, on obtient

$$z\tilde{s}_{n-1}(z) = z\sum_{j=1}^{\infty} x_{n-k-1+j}z^{-j} = \sum_{j=0}^{\infty} x_{n-k+j}z^{-j} = x_{n-k} + \tilde{s}_n(z).$$

En enlevant le terme x_{n-k} on obtient $\tilde{s}_n(z)$. I.e.,

$$\tilde{s}_n(z) = z\tilde{s}_{n-1}(z) \bmod 1,$$

où " $\bmod 1$" veut dire que l'on enlève la partie polynômiale.

En multipliant $\tilde{s}_{n-1}(z)$ par z, on obtient

$$z\tilde{s}_{n-1}(z) = z\sum_{j=1}^{\infty} x_{n-k-1+j}z^{-j} = \sum_{j=0}^{\infty} x_{n-k+j}z^{-j} = x_{n-k} + \tilde{s}_n(z).$$

En enlevant le terme x_{n-k} on obtient $\tilde{s}_n(z)$. I.e.,

$$\tilde{s}_n(z) = z\tilde{s}_{n-1}(z) \bmod 1,$$

où " $\bmod 1$" veut dire que l'on enlève la partie polynômiale.

En multipliant par $P(z)$, on obtient

$$p_n(z) = zp_{n-1}(z) \bmod P(z).$$

On a ainsi un LCG dans un espace de polynômes, avec modulo $P(z)$ et multiplicateur z.

Le MRG est de pleine période $m^k - 1$ ssi cette récurrence l'est. Si on prend $p_0(z) = 1$ comme état initial, on voit que la période maximale est atteinte ssi le plus petit n tel que $z^n \bmod P(z) = 1$ est $n = m^k - 1$.

Le MRG est de pleine période $m^k - 1$ ssi cette récurrence l'est. Si on prend $p_0(z) = 1$ comme état initial, on voit que la période maximale est atteinte ssi le plus petit n tel que $z^n \bmod P(z) = 1$ est $n = m^k - 1$.

Un polynôme $P(z)$ qui satisfait cette condition s'appelle un polynôme primitif modulo m. Un tel polynôme ne peut exister que si m est premier, auquel cas $\mathbb{Z}_m$ devient le corps fini $\mathbb{F}_m$.

Le MRG est de pleine période $m^k - 1$ ssi cette récurrence l'est. Si on prend $p_0(z) = 1$ comme état initial, on voit que la période maximale est atteinte ssi le plus petit n tel que $z^n \bmod P(z) = 1$ est $n = m^k - 1$.

Un polynôme $P(z)$ qui satisfait cette condition s'appelle un polynôme primitif modulo m. Un tel polynôme ne peut exister que si m est premier, auquel cas $\mathbb{Z}_m$ devient le corps fini $\mathbb{F}_m$.

Pour $k > 1$, un tel polynôme doit avoir au moins deux coefficients non nuls, dont a_k.

Le MRG est de pleine période $m^k - 1$ ssi cette récurrence l'est. Si on prend $p_0(z) = 1$ comme état initial, on voit que la période maximale est atteinte ssi le plus petit n tel que $z^n \bmod P(z) = 1$ est $n = m^k - 1$.

Un polynôme $P(z)$ qui satisfait cette condition s'appelle un polynôme primitif modulo m. Un tel polynôme ne peut exister que si m est premier, auquel cas $\mathbb{Z}_m$ devient le corps fini $\mathbb{F}_m$.

Pour $k > 1$, un tel polynôme doit avoir au moins deux coefficients non nuls, dont a_k.
Ainsi, la récurrence la plus économique a la forme:

$$x_n = (a_r x_{n-r} + a_k x_{n-k}) \bmod m.$$

Pour vérifier la condition de période max., on ne veut pas calculer explicitement $z^n \bmod P(z) = 1$ pour $n = 1, \ldots, m^k - 1$.

Pour vérifier la condition de période max., on ne veut pas calculer explicitement $z^n \bmod P(z) = 1$ pour $n = 1, \ldots, m^k - 1$.

Les conditions suivantes sont plus pratiques:

Proposition. (Alanen et Knuth, 1964).
Soit $r = (m^k - 1)/(m - 1)$. Le polynôme $P(z)$ est primitif modulo m ssi les trois conditions suivantes sont satisfaites:

(i) $((-1)^{k+1} a_k)^{(m-1)/q} \bmod m \neq 1$ pour tout q facteur premier de $m - 1$;

(ii) $z^r \bmod (P(z), m) = (-1)^{k+1} a_k \bmod m$;

(iii) $(z^{r/q} \bmod (P(z), m))$ est de degré positif pour chaque facteur premier q de r, $1 < q < r$.

Si m est premier et $k = 1$, alors $r = 1$ et ces conditions se réduisent à

$$a_1^{(m-1)/q} \bmod m \neq 1 \text{ pour chaque facteur premier } q \text{ de } m - 1.$$

Un tel a_1 s'appelle un élément primitif modulo m.

Pour chercher des polynômes primitifs, pour $k > 1$, on cherche d'abord un a_k qui satisfait (i), puis on cherche au hasard pour des vecteurs de coefficients $(a_1, \ldots, a_{k-1})$ parmi ceux qui satisfont aussi certaines contraintes pour l'implantation.

Pour chercher des polynômes primitifs, pour $k > 1$, on cherche d'abord un a_k qui satisfait (i), puis on cherche au hasard pour des vecteurs de coefficients $(a_1, \ldots, a_{k-1})$ parmi ceux qui satisfont aussi certaines contraintes pour l'implantation.

La proportion des $m^k - 1$ polynômes qui sont primitifs pour k et m donnés est

$$\frac{1}{k} \prod_{j=1}^{J} \frac{p_j - 1}{p_j}$$

où $p_1, \ldots, p_J$ sont les facteurs premiers distincts de $m^k - 1$.

Pour chercher des polynômes primitifs, pour $k > 1$, on cherche d'abord un a_k qui satisfait (i), puis on cherche au hasard pour des vecteurs de coefficients $(a_1, \ldots, a_{k-1})$ parmi ceux qui satisfont aussi certaines contraintes pour l'implantation.

La proportion des $m^k - 1$ polynômes qui sont primitifs pour k et m donnés est

$$\frac{1}{k} \prod_{j=1}^{J} \frac{p_j - 1}{p_j}$$

où $p_1, \ldots, p_J$ sont les facteurs premiers distincts de $m^k - 1$.

Exemple. Soient $m = 2^{31} - 1$ et $k = 1$. Alors $m^k - 1 = m - 1 = 2^{31} - 2 = 2 \times 3^2 \times 7 \times 11 \times 31 \times 151 \times 331$ et cette proportion est $= (1/2)(2/3)(6/7)(10/11)(30/31)(150/151)(330/331) \approx 0.248943$.

Pour chercher des polynômes primitifs, pour $k > 1$, on cherche d'abord un a_k qui satisfait (i), puis on cherche au hasard pour des vecteurs de coefficients $(a_1, \ldots, a_{k-1})$ parmi ceux qui satisfont aussi certaines contraintes pour l'implantation.

La proportion des $m^k - 1$ polynômes qui sont primitifs pour k et m donnés est

$$\frac{1}{k} \prod_{j=1}^{J} \frac{p_j - 1}{p_j}$$

où $p_1, \ldots, p_J$ sont les facteurs premiers distincts de $m^k - 1$.

Exemple. Soient $m = 2^{31} - 1$ et $k = 1$. Alors $m^k - 1 = m - 1 = 2^{31} - 2 = 2 \times 3^2 \times 7 \times 11 \times 31 \times 151 \times 331$ et cette proportion est $= (1/2)(2/3)(6/7)(10/11)(30/31)(150/151)(330/331) \approx 0.248943$.
Si $m = 2^{31} - 1$ et $k = 2$, alors $m^k - 1 = (m-1)(m+1) = (2^{31} - 2)2^{31}$. Les p_j sont les mêmes et la proportion devient ≈ 0.124471.

On a $m^k - 1 = 2hr$ où $h = (m-1)/2$ et $r = (m^k - 1)/(m-1)$.
Il peut être très difficile de factoriser r.
Idée: choisir h et r premiers.

On a $m^k - 1 = 2hr$ où $h = (m-1)/2$ et $r = (m^k - 1)/(m-1)$.
Il peut être très difficile de factoriser r.
Idée: choisir h et r premiers.

La proportion de polynômes primitifs est alors approx.

$(h-1)(r-1)/(2hrk) \approx 1/(2k)$ pour $m \gg 2$ et

$(2^k - 2)/(k(2^k - 1)) \approx 1/k$ pour $m = 2$ et $k \gg 1$.

$$m = 2^e$$

Facile de calculer le produit $ax \bmod m$.

$$m = 2^e$$

Facile de calculer le produit $ax \bmod m$. Mais:

$m = 2^e$

Facile de calculer le produit $ax \bmod m$. Mais:

Pour $k = 1$ et $e \geq 4$, on a $\rho \leq 2^{e-2}$;

$m = 2^e$

Facile de calculer le produit $ax \bmod m$. Mais:

Pour $k = 1$ et $e \geq 4$, on a $\rho \leq 2^{e-2}$;
Pour $k > 1$, on a $\rho \leq (2^k - 1)2^{e-1}$.

$m = 2^e$

Facile de calculer le produit $ax \bmod m$. Mais:

Pour $k = 1$ et $e \geq 4$, on a $\rho \leq 2^{e-2}$;
Pour $k > 1$, on a $\rho \leq (2^k - 1)2^{e-1}$.

Exemple. Si $k = 7$ et $m = 2^{31} - 1$, la période max. est $(2^{31} - 1)^7 - 1 \approx 2^{217}$.

$m = 2^e$

Facile de calculer le produit $ax \bmod m$. Mais:

Pour $k = 1$ et $e \geq 4$, on a $\rho \leq 2^{e-2}$;
Pour $k > 1$, on a $\rho \leq (2^k - 1)2^{e-1}$.

Exemple. Si $k = 7$ et $m = 2^{31} - 1$, la période max. est $(2^{31} - 1)^7 - 1 \approx 2^{217}$.
Mais pour $m = 2^{31}$ on a $\rho \leq (2^7 - 1)2^{31-1} < 2^{37}$, i.e. 2^{180} fois plus petit!

$m = 2^e$

Facile de calculer le produit $ax \bmod m$. Mais:

Pour $k = 1$ et $e \geq 4$, on a $\rho \leq 2^{e-2}$;
Pour $k > 1$, on a $\rho \leq (2^k - 1)2^{e-1}$.

Exemple. Si $k = 7$ et $m = 2^{31} - 1$, la période max. est $(2^{31} - 1)^7 - 1 \approx 2^{217}$.
Mais pour $m = 2^{31}$ on a $\rho \leq (2^7 - 1)2^{31-1} < 2^{37}$, i.e. 2^{180} fois plus petit!

Pour $k = 1$, la période de $x_n \bmod 2^i$ ne peut pas dépasser $\max(1, 2^{i-2})$.

$m = 2^e$

Facile de calculer le produit $ax \bmod m$. Mais:

Pour $k = 1$ et $e \geq 4$, on a $\rho \leq 2^{e-2}$;
Pour $k > 1$, on a $\rho \leq (2^k - 1)2^{e-1}$.

Exemple. Si $k = 7$ et $m = 2^{31} - 1$, la période max. est $(2^{31} - 1)^7 - 1 \approx 2^{217}$.
Mais pour $m = 2^{31}$ on a $\rho \leq (2^7 - 1)2^{31-1} < 2^{37}$, i.e. 2^{180} fois plus petit!

Pour $k = 1$, la période de $x_n \bmod 2^i$ ne peut pas dépasser $\max(1, 2^{i-2})$.
Pour $k > 1$, la période de $x_n \bmod 2^i$ ne peut pas dépasser $(2^k - 1)2^{i-1}$.

Exemple. Récurrence $x_n = 10205x_{n-1} \bmod 2^{15}$:

$$\begin{aligned}
x_0 &= 12345 = 011000000111001_2\\
x_1 &= 20533 = 101000000110101_2\\
x_2 &= 20673 = 101000011000001_2\\
x_3 &= 7581 = 001110110011101_2\\
x_4 &= 31625 = 111101110001001_2\\
x_5 &= 1093 = 000010001000101_2\\
x_6 &= 12945 = 011001010010001_2\\
x_7 &= 15917 = 011111000101101_2.
\end{aligned}$$

De tels générateurs ont quand même été populaires récemment.
Il ne faut pas les utiliser!

m	a	c	Source
2^{24}	1140671485	12820163	early MS VisualBasic
2^{31}	65539	0	RANDU
2^{31}	134775813	1	early Turbo Pascal
2^{31}	1103515245	12345	`rand()` in BSD ANSI C
2^{32}	69069	1	VAX/VMS systems
2^{32}	2147001325	715136305	BCLP language
2^{35}	5^{15}	7261067085	Knuth (1998)
2^{48}	68909602460261	0	Fishman (1990)
2^{48}	25214903917	11	Unix's `rand48()`
2^{48}	44485709377909	0	CRAY system
2^{59}	13^{13}	0	NAG Fortran/C library

$$x_n = (ax_{n-1} + c) \bmod m; \qquad u_n = x_n/m.$$

Sauter en avant. On peut écrire

$$\mathbf{x}_n = \mathbf{A}\mathbf{x}_{n-1} \bmod m = \begin{pmatrix} 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ a_k & a_{k-1} & \cdots & a_1 \end{pmatrix} \mathbf{x}_{n-1} \bmod m.$$

Sauter en avant. On peut écrire

$$\mathbf{x}_n = \mathbf{A}\mathbf{x}_{n-1} \bmod m = \begin{pmatrix} 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ a_k & a_{k-1} & \cdots & a_1 \end{pmatrix} \mathbf{x}_{n-1} \bmod m.$$

Ainsi

$$\mathbf{x}_{n+\nu} = \mathbf{A}^\nu \mathbf{x}_n \bmod m = (\mathbf{A}^\nu \bmod m)\mathbf{x}_n \bmod m.$$

Sauter en avant. On peut écrire

$$\mathbf{x}_n = \mathbf{A}\mathbf{x}_{n-1} \bmod m = \begin{pmatrix} 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ a_k & a_{k-1} & \cdots & a_1 \end{pmatrix} \mathbf{x}_{n-1} \bmod m.$$

Ainsi

$$\mathbf{x}_{n+\nu} = \mathbf{A}^\nu \mathbf{x}_n \bmod m = (\mathbf{A}^\nu \bmod m)\mathbf{x}_n \bmod m.$$

On peut précalculer $\mathbf{A}^\nu \bmod m$ via

$$\mathbf{A}^\nu \bmod m = \begin{cases} (\mathbf{A}^{\nu/2} \bmod m)(\mathbf{A}^{\nu/2} \bmod m) \bmod m & \text{si } \nu \text{ est pair;} \\ \mathbf{A}(\mathbf{A}^{\nu-1} \bmod m) \bmod m & \text{si } \nu \text{ est impair.} \end{cases}$$

Sauter en avant. On peut écrire

$$\mathbf{x}_n = \mathbf{A}\mathbf{x}_{n-1} \bmod m = \begin{pmatrix} 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ a_k & a_{k-1} & \cdots & a_1 \end{pmatrix} \mathbf{x}_{n-1} \bmod m.$$

Ainsi

$$\mathbf{x}_{n+\nu} = \mathbf{A}^\nu \mathbf{x}_n \bmod m = (\mathbf{A}^\nu \bmod m)\mathbf{x}_n \bmod m.$$

On peut précalculer $\mathbf{A}^\nu \bmod m$ via

$$\mathbf{A}^\nu \bmod m = \begin{cases} (\mathbf{A}^{\nu/2} \bmod m)(\mathbf{A}^{\nu/2} \bmod m) \bmod m & \text{si } \nu \text{ est pair;} \\ \mathbf{A}(\mathbf{A}^{\nu-1} \bmod m) \bmod m & \text{si } \nu \text{ est impair.} \end{cases}$$

Avec la représentation polynômiale,

$$p_{n+\nu}(z) = z^\nu p_n(z) \bmod P(z) = (z^\nu \bmod P(z))p_n(z) \bmod P(z),$$

où $z^\nu \bmod P(z)$ peut être précalculé.

Structure de réseau

Structure de Ψ_t:
$(x_0, \ldots, x_{k-1})$ peut être n'importe quel vecteur dans $\{0, 1, \ldots, m-1\}^k$.
Ensuite, $x_k, x_{k+1}, \ldots$ sont déterminés par la récurrence.

Structure de réseau

Structure de Ψ_t:
$(x_0, \ldots, x_{k-1})$ peut être n'importe quel vecteur dans $\{0, 1, \ldots, m-1\}^k$.
Ensuite, $x_k, x_{k+1}, \ldots$ sont déterminés par la récurrence.

Si $(x_0, \ldots, x_{k-1}) = (1, 0, \ldots, 0)$, alors on a

$$x_k = a_k, \qquad x_{k+1} = a_1 a_k \bmod m, \qquad x_{k+2} = (a_1^2 + a_2) a_k \bmod m, \ \ldots$$

Structure de réseau

Structure de Ψ_t:
$(x_0, \ldots, x_{k-1})$ peut être n'importe quel vecteur dans $\{0, 1, \ldots, m-1\}^k$.
Ensuite, $x_k, x_{k+1}, \ldots$ sont déterminés par la récurrence.

Si $(x_0, \ldots, x_{k-1}) = (1, 0, \ldots, 0)$, alors on a
$x_k = a_k, \qquad x_{k+1} = a_1 a_k \bmod m, \qquad x_{k+2} = (a_1^2 + a_2) a_k \bmod m, \ldots$
Si $(x_0, \ldots, x_{k-1}) = (0, 1, \ldots, 0)$, alors
$x_k = a_{k-1}, \qquad x_{k+1} = (a_1 a_{k-1} + a_k) \bmod m,$
$x_{k+2} = (a_1^2 a_{k-1} + a_1 a_k + a_2 a_{k-1}) \bmod m), \ldots$

Structure de réseau

Structure de Ψ_t:
$(x_0, \ldots, x_{k-1})$ peut être n'importe quel vecteur dans $\{0, 1, \ldots, m-1\}^k$.
Ensuite, $x_k, x_{k+1}, \ldots$ sont déterminés par la récurrence.

Si $(x_0, \ldots, x_{k-1}) = (1, 0, \ldots, 0)$, alors on a
$x_k = a_k, \qquad x_{k+1} = a_1 a_k \bmod m, \qquad x_{k+2} = (a_1^2 + a_2) a_k \bmod m, \ldots$
Si $(x_0, \ldots, x_{k-1}) = (0, 1, \ldots, 0)$, alors
$x_k = a_{k-1}, \qquad x_{k+1} = (a_1 a_{k-1} + a_k) \bmod m,$
$x_{k+2} = (a_1^2 a_{k-1} + a_1 a_k + a_2 a_{k-1}) \bmod m), \ldots$
$\vdots$
Si $(x_0, \ldots, x_{k-1}) = (0, \ldots, 0, 1)$, alors $x_k = a_1, \qquad x_{k+1} = (a_1^2 + a_2) \bmod m)$,
etc.

Structure de réseau

Structure de Ψ_t:
$(x_0, \ldots, x_{k-1})$ peut être n'importe quel vecteur dans $\{0, 1, \ldots, m-1\}^k$.
Ensuite, $x_k, x_{k+1}, \ldots$ sont déterminés par la récurrence.

Si $(x_0, \ldots, x_{k-1}) = (1, 0, \ldots, 0)$, alors on a
$x_k = a_k, \qquad x_{k+1} = a_1 a_k \bmod m, \qquad x_{k+2} = (a_1^2 + a_2) a_k \bmod m, \ldots$
Si $(x_0, \ldots, x_{k-1}) = (0, 1, \ldots, 0)$, alors
$x_k = a_{k-1}, \qquad x_{k+1} = (a_1 a_{k-1} + a_k) \bmod m,$
$x_{k+2} = (a_1^2 a_{k-1} + a_1 a_k + a_2 a_{k-1}) \bmod m), \ldots$
$\vdots$
Si $(x_0, \ldots, x_{k-1}) = (0, \ldots, 0, 1)$, alors $x_k = a_1, \qquad x_{k+1} = (a_1^2 + a_2) \bmod m)$,
etc.

Tout vecteur $(x_n, \ldots, x_{n+t-1})$ qui obéit à la récurrence, pour $t \geq k$, est une combinaison linéaire à coefficients entiers de ces k vecteurs de base.

Tout vecteur $(x_n, \ldots, x_{n+t-1})$ qui obéit à la récurrence, pour $t \geq k$, est une combinaison linéaire à coefficients entiers de ces k vecteurs de base.

Tout vecteur $(x_n, \ldots, x_{n+t-1})$ qui obéit à la récurrence, pour $t \geq k$, est une combinaison linéaire à coefficients entiers de ces k vecteurs de base.

Notons $x_{i,0}, x_{i,1}, x_{i,2}, \ldots$ la suite obtenue quand $(x_0, \ldots, x_{k-1}) = \mathbf{e}_i$.

Tout vecteur $(x_n, \ldots, x_{n+t-1})$ qui obéit à la récurrence, pour $t \geq k$, est une combinaison linéaire à coefficients entiers de ces k vecteurs de base.

Notons $x_{i,0}, x_{i,1}, x_{i,2}, \ldots$ la suite obtenue quand $(x_0, \ldots, x_{k-1}) = \mathbf{e}_i$.

Un état initial $(x_0, \ldots, x_{k-1}) = (z_1, \ldots, z_k)$ peut s'écrire comme $z_1\mathbf{e}_1 + \cdots + z_k\mathbf{e}_k$ et produit la suite $z_1(x_{1,0}, x_{1,1}, \ldots) + \cdots + z_k(x_{k,0}, x_{k,1}, \ldots) \bmod m$.

Tout vecteur $(x_n, \ldots, x_{n+t-1})$ qui obéit à la récurrence, pour $t \geq k$, est une combinaison linéaire à coefficients entiers de ces k vecteurs de base.

Notons $x_{i,0}, x_{i,1}, x_{i,2}, \ldots$ la suite obtenue quand $(x_0, \ldots, x_{k-1}) = \mathbf{e}_i$.

Un état initial $(x_0, \ldots, x_{k-1}) = (z_1, \ldots, z_k)$ peut s'écrire comme $z_1\mathbf{e}_1 + \cdots + z_k\mathbf{e}_k$ et produit la suite $z_1(x_{1,0}, x_{1,1}, \ldots) + \cdots + z_k(x_{k,0}, x_{k,1}, \ldots) \bmod m$.

Tout vecteur $(x_n, \ldots, x_{n+t-1})$ qui obéit à la récurrence, pour $t \geq k$, est une combinaison linéaire à coefficients entiers de ces k vecteurs de base.

Notons $x_{i,0}, x_{i,1}, x_{i,2}, \ldots$ la suite obtenue quand $(x_0, \ldots, x_{k-1}) = \mathbf{e}_i$.

Un état initial $(x_0, \ldots, x_{k-1}) = (z_1, \ldots, z_k)$ peut s'écrire comme $z_1\mathbf{e}_1 + \cdots + z_k\mathbf{e}_k$ et produit la suite $z_1(x_{1,0}, x_{1,1}, \ldots) + \cdots + z_k(x_{k,0}, x_{k,1}, \ldots) \bmod m$.

La réduction modulo m se fait en soustrayant des vecteurs $m\mathbf{e}_i$.

Tout vecteur $(x_n, \ldots, x_{n+t-1})$ qui obéit à la récurrence, pour $t \geq k$, est une combinaison linéaire à coefficients entiers de ces k vecteurs de base.

Notons $x_{i,0}, x_{i,1}, x_{i,2}, \ldots$ la suite obtenue quand $(x_0, \ldots, x_{k-1}) = \mathbf{e}_i$.

Un état initial $(x_0, \ldots, x_{k-1}) = (z_1, \ldots, z_k)$ peut s'écrire comme $z_1\mathbf{e}_1 + \cdots + z_k\mathbf{e}_k$ et produit la suite $z_1(x_{1,0}, x_{1,1}, \ldots) + \cdots + z_k(x_{k,0}, x_{k,1}, \ldots) \bmod m$.

La réduction modulo m se fait en soustrayant des vecteurs $m\mathbf{e}_i$.

Ainsi, pour $t \geq k$, $(x_0, x_1, \ldots, x_{t-1})$ suit la récurrence ssi c'est une combinaison linéaire à coefficients entiers de

$$(1, 0, \ldots, 0, x_{1,k}, \ldots, x_{1,t-1})$$
$$\vdots$$
$$(0, 0, \ldots, 1, x_{k,k}, \ldots, x_{k,t-1})$$
$$(0, 0, \ldots, 0, m, \ldots, 0)$$
$$\vdots$$
$$(0, 0, \ldots, 0, 0, \ldots, m).$$

En divisant par m, on obtient que $(u_0, \ldots, u_{t-1}) \in [0,1)^t$ est dans Ψ_t ssi c'est une combinaison linéaire (sur les entiers) de

$$\begin{aligned}
\mathbf{v}_1 &= (1, 0, \ldots, 0, x_{1,k}, \ldots, x_{1,t-1})^{\mathsf{t}}/m \\
&\vdots \qquad\quad \vdots \\
\mathbf{v}_k &= (0, 0, \ldots, 1, x_{k,k}, \ldots, x_{k,t-1})^{\mathsf{t}}/m \\
\mathbf{v}_{k+1} &= (0, 0, \ldots, 0, 1, \ldots, 0)^{\mathsf{t}} \\
&\vdots \qquad\quad \vdots \\
\mathbf{v}_t &= (0, 0, \ldots, 0, 0, \ldots, 1)^{\mathsf{t}}.
\end{aligned}$$

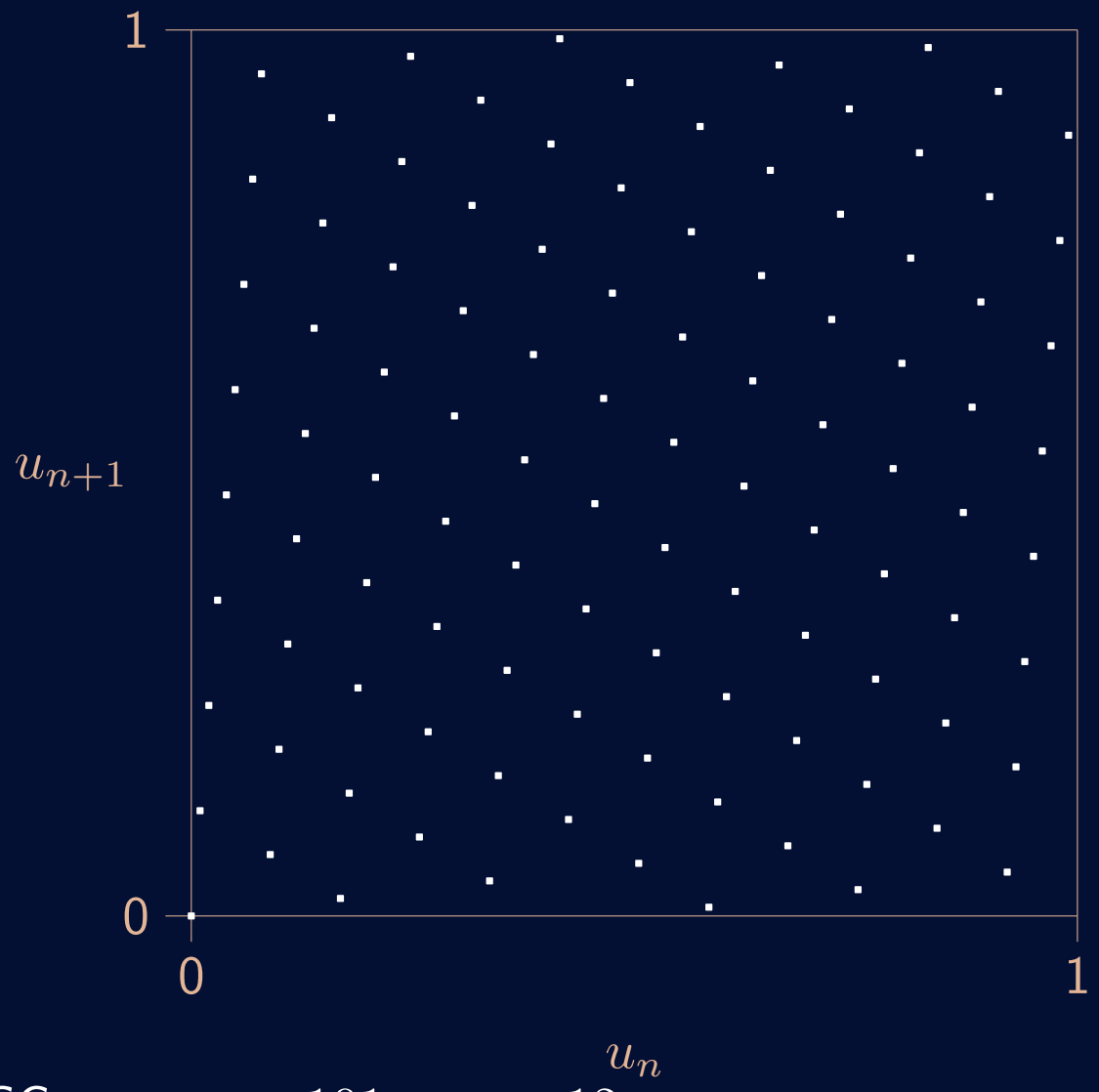
En divisant par m, on obtient que $(u_0, \ldots, u_{t-1}) \in [0,1)^t$ est dans Ψ_t ssi c'est une combinaison linéaire (sur les entiers) de

$$\begin{aligned}
\mathbf{v}_1 &= (1, 0, \ldots, 0, x_{1,k}, \ldots, x_{1,t-1})^{\mathsf{t}}/m \\
&\vdots \qquad\quad \vdots \\
\mathbf{v}_k &= (0, 0, \ldots, 1, x_{k,k}, \ldots, x_{k,t-1})^{\mathsf{t}}/m \\
\mathbf{v}_{k+1} &= (0, 0, \ldots, 0, 1, \ldots, 0)^{\mathsf{t}} \\
&\vdots \qquad\quad \vdots \\
\mathbf{v}_t &= (0, 0, \ldots, 0, 0, \ldots, 1)^{\mathsf{t}}.
\end{aligned}$$

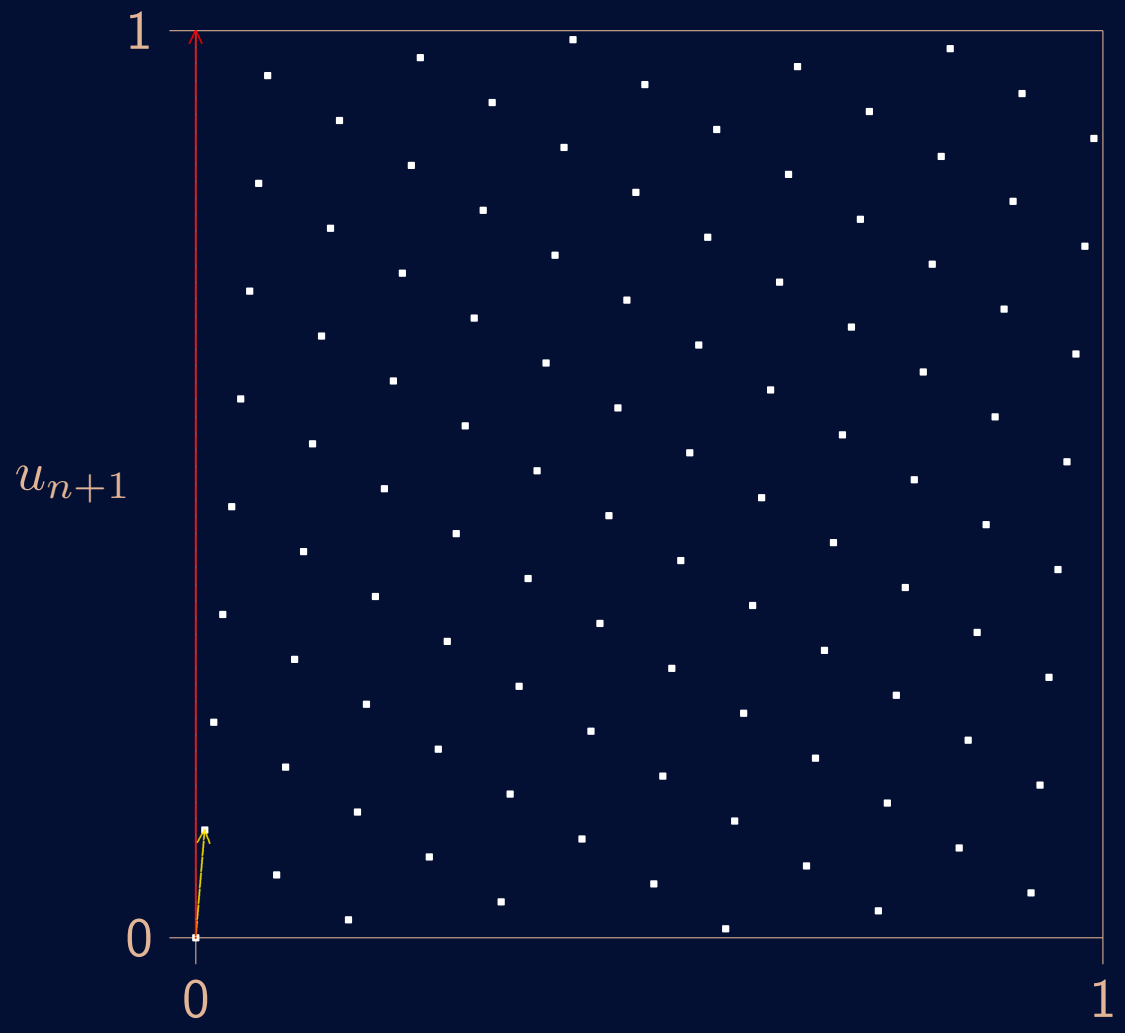
Si

$$L_t = \left\{ \mathbf{v} = \sum_{i=1}^{t} z_i \mathbf{v}_i \;\middle|\; z_i \in \mathbb{Z} \right\}$$

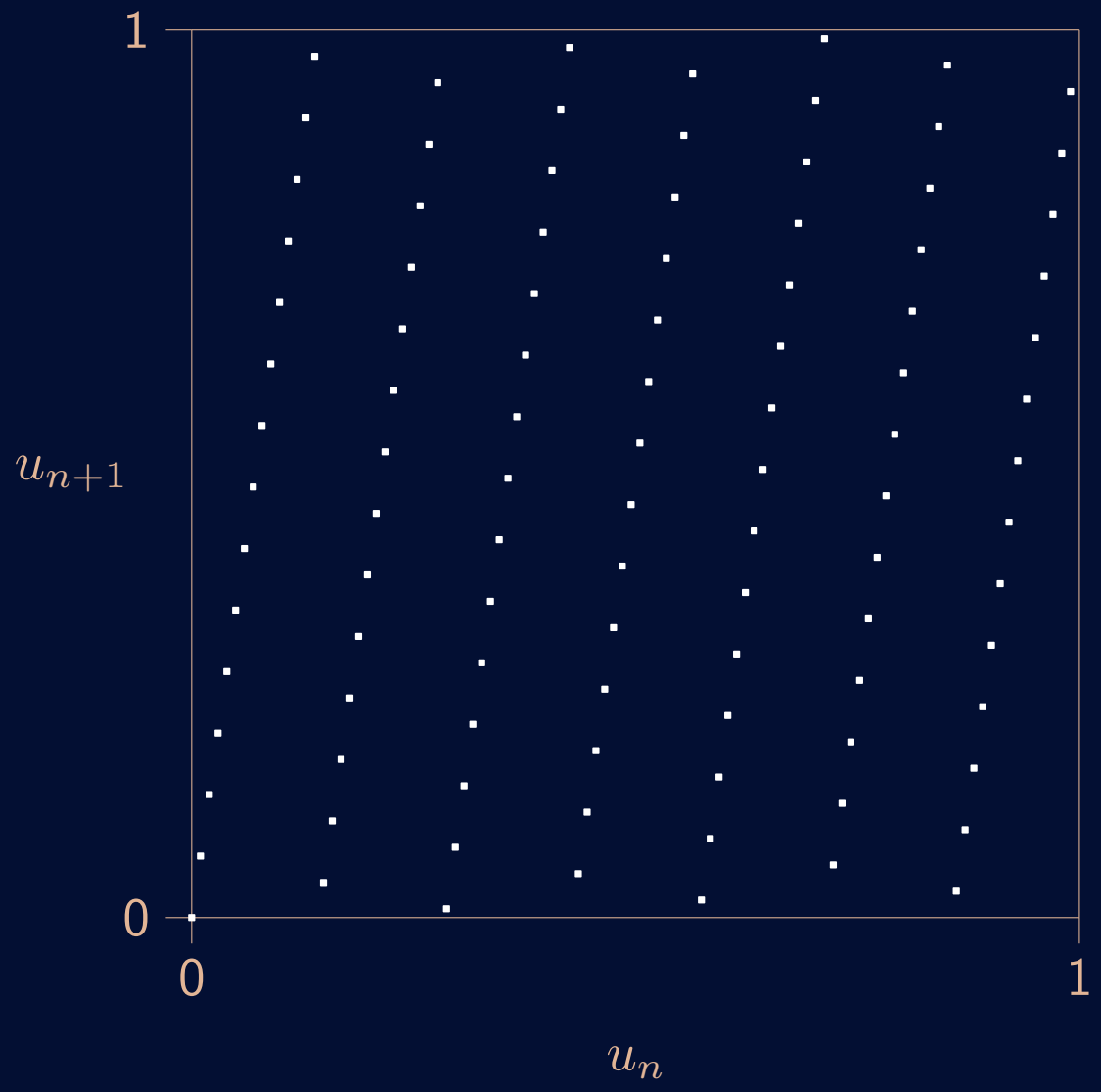
est le réseau ayant ces vecteurs pour base, alors $\Psi_t = L_t \cap [0,1)^t$.

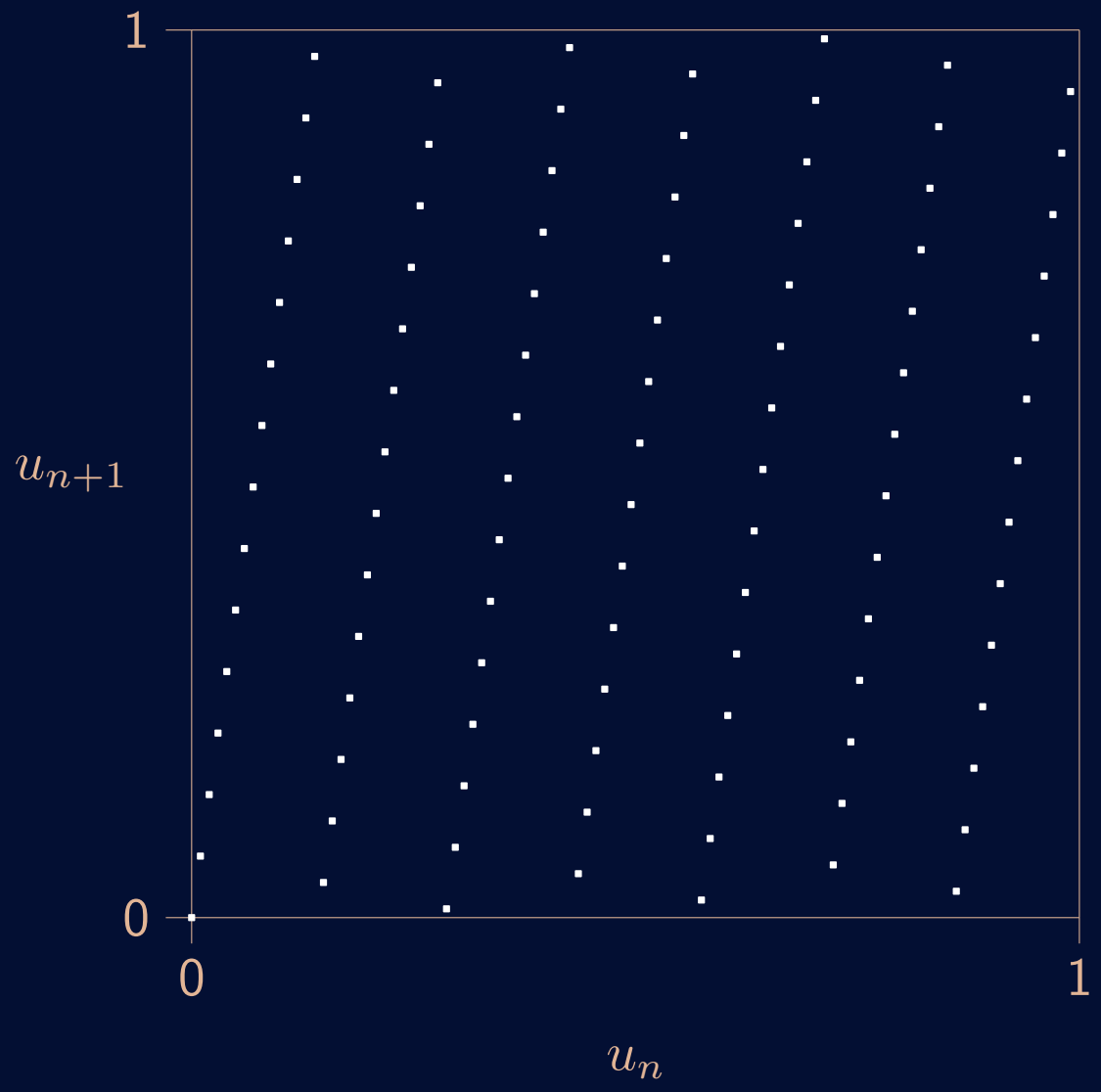


Exemple: LCG avec $m = 101$ et $a = 12$;

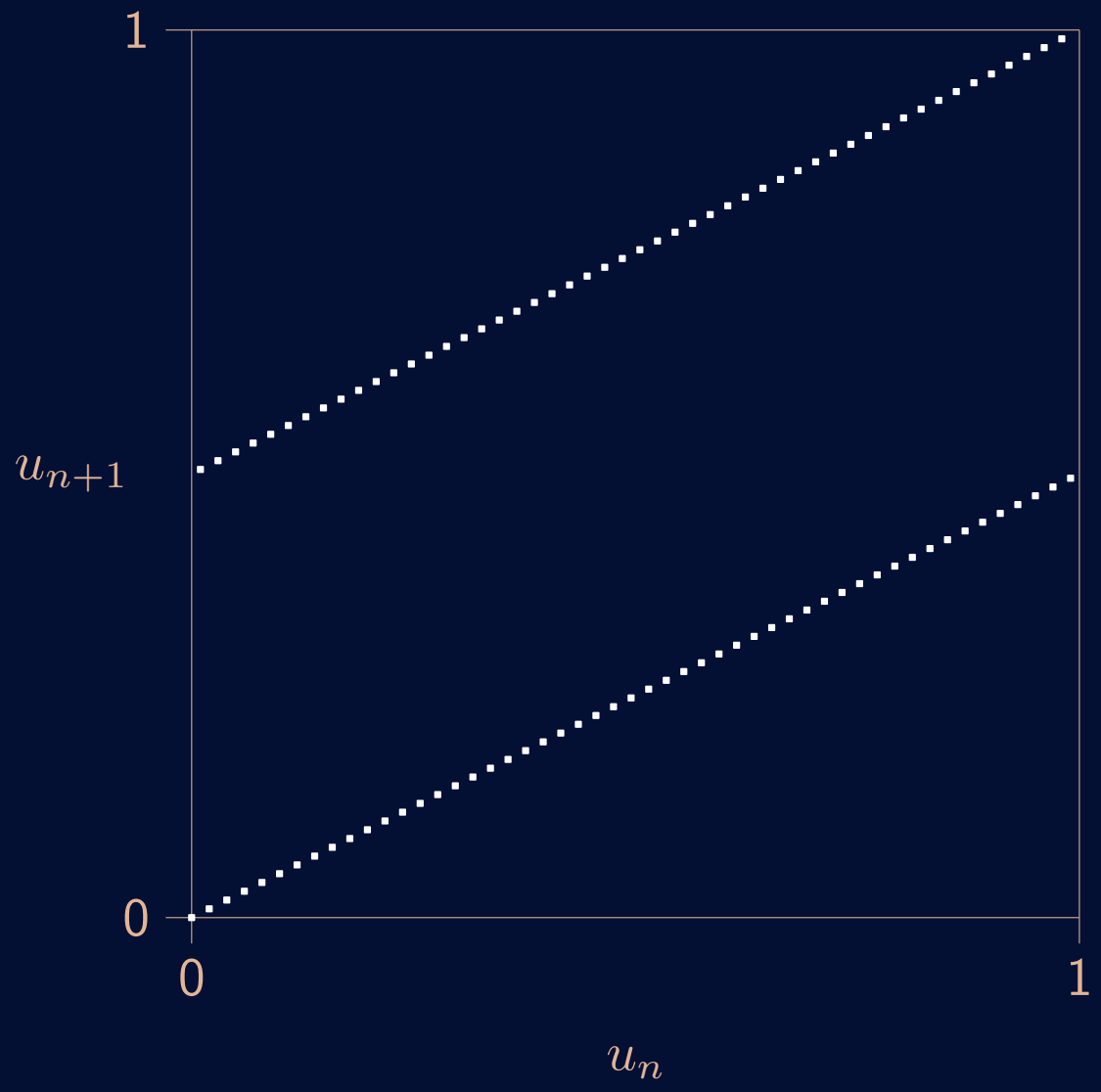


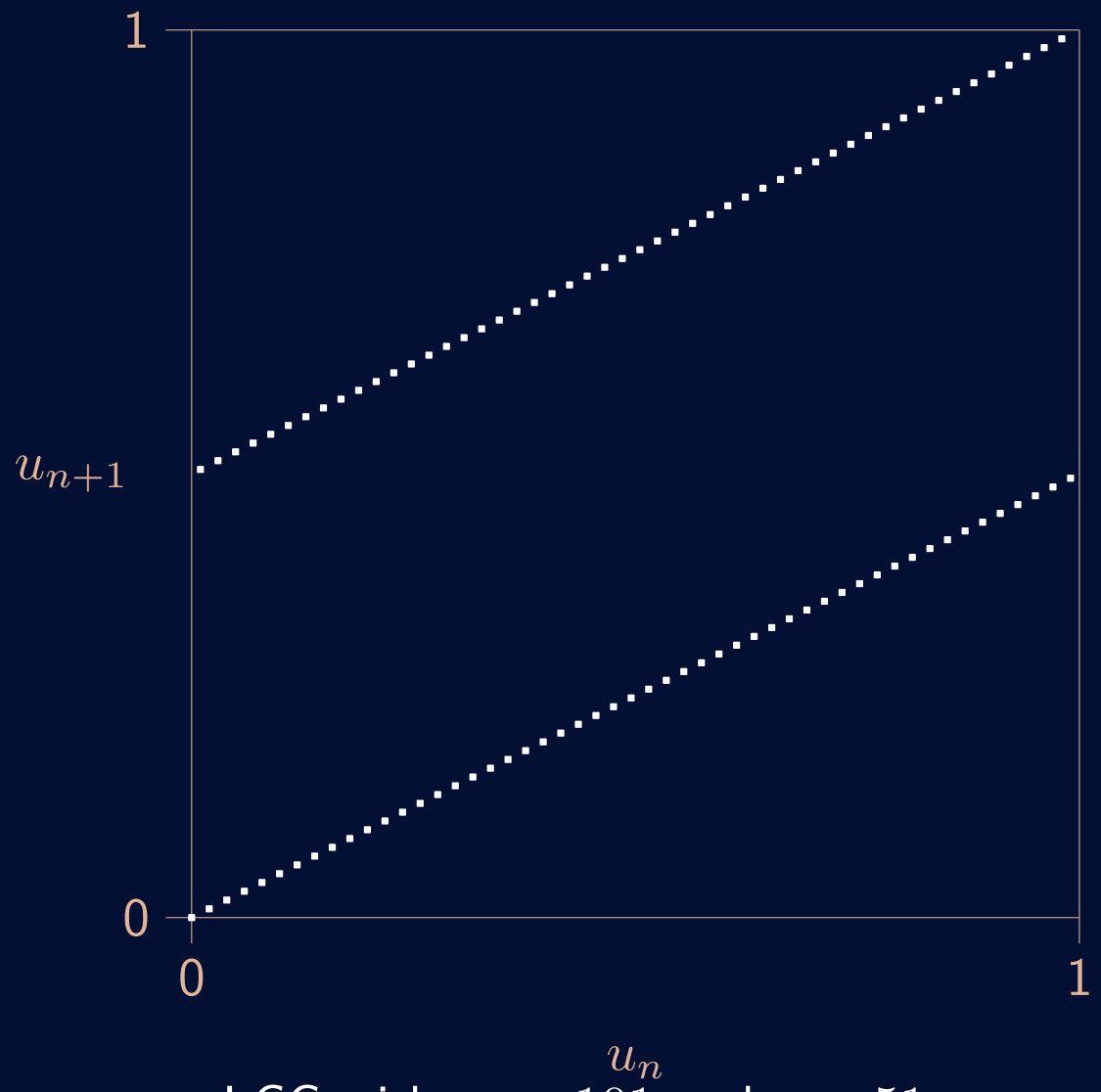
Exemple: LCG avec $m = 101$ et $a = 12$; $\mathbf{v}_1 = (1, 12)/101$, $\mathbf{v}_2 = (0, 1)$


1
u_{n+1}
0
0
1
u_n



LCG with $m = 101$ and $a = 7$


1
u_{n+1}
0
0
1
u_n



LCG with $m = 101$ and $a = 51$

Pour $t > k$, il y a m^t vecteurs dont les coordonnées sont des multiples de $1/m$, mais seulement m^k sont dans Ψ_t, soit une proportion de $1/m^{t-k}$.

Pour $t > k$, il y a m^t vecteurs dont les coordonnées sont des multiples de $1/m$, mais seulement m^k sont dans Ψ_t, soit une proportion de $1/m^{t-k}$.

En t dimensions, les points sont dans des hyperplans parallèles équidistants, et ont une structure très régulière.

Pour $t > k$, il y a m^t vecteurs dont les coordonnées sont des multiples de $1/m$, mais seulement m^k sont dans Ψ_t, soit une proportion de $1/m^{t-k}$.

En t dimensions, les points sont dans des hyperplans parallèles équidistants, et ont une structure très régulière.

On peut calculer la distance $1/\ell_t$ entre les hyperplans successifs, pour la famille où ils sont le plus éloignés.

Cela se fait en résolvant un problème d'optimisation quadratique en nombres entiers.

Indices lacunaires.

Pour $I = \{i_1, i_2, \cdots, i_t\}$ on a

$$\begin{aligned} \Psi_I &= \{(u_{i_1}, \ldots, u_{i_t}) \mid s_0 = (x_0, \ldots, x_{k-1}) \in \mathbb{Z}_m^k\}, \\ &= L_I \cap [0,1)^t, \\ 1/\ell_I &= \text{ distance entre les hyperplans dans } L_I. \end{aligned}$$

Mesures d'uniformité.

On connait des bornes supérieures de la forme $\ell_t \leq \ell_t^*(n)$ pour un réseau général de densité n dans $\mathbb{R}^t$.

Mesures d'uniformité.

On connait des bornes supérieures de la forme $\ell_t \leq \ell_t^*(n)$ pour un réseau général de densité n dans $\mathbb{R}^t$.

On peut alors standardiser ℓ_t par $\ell_t/\ell_t^*(m^k)$ pour avoir une mesure dans $[0, 1]$.

Mesures d'uniformité.

On connait des bornes supérieures de la forme $\ell_t \leq \ell_t^*(n)$ pour un réseau général de densité n dans $\mathbb{R}^t$.

On peut alors standardiser ℓ_t par $\ell_t/\ell_t^*(m^k)$ pour avoir une mesure dans $[0,1]$.

Figure de mérite générale:

$$M_{\mathcal{J}} = \min_{I \in \mathcal{J}} \ \ell_I/\ell_{|I|}^*(m^k)$$

où $\mathcal{J}$ est une famille d'ensembles de la forme $I = \{0, i_2, \cdots, i_t\}$.

Mesures d'uniformité.

On connait des bornes supérieures de la forme $\ell_t \leq \ell_t^*(n)$ pour un réseau général de densité n dans $\mathbb{R}^t$.

On peut alors standardiser ℓ_t par $\ell_t/\ell_t^*(m^k)$ pour avoir une mesure dans $[0, 1]$.

Figure de mérite générale:

$$M_{\mathcal{J}} = \min_{I \in \mathcal{J}} \ \ell_I/\ell_{|I|}^*(m^k)$$

où $\mathcal{J}$ est une famille d'ensembles de la forme $I = \{0, i_2, \cdots, i_t\}$.

On peut chercher par ordinateur des paramètres qui "maximisent" cette mesure.

Autres bornes.

Proposition. Si $i \in I$ lorsque $a_{k-i} \neq 0$ (avec $a_0 = -1$), alors

$$\ell_I^2 \leq 1 + a_1^2 + \cdots + a_k^2.$$

Autres bornes.

Proposition. Si $i \in I$ lorsque $a_{k-i} \neq 0$ (avec $a_0 = -1$), alors

$$\ell_I^2 \leq 1 + a_1^2 + \cdots + a_k^2.$$

Il faut donc que cette somme soit grande!

Autres bornes.

Proposition. Si $i \in I$ lorsque $a_{k-i} \neq 0$ (avec $a_0 = -1$), alors

$$\ell_I^2 \leq 1 + a_1^2 + \cdots + a_k^2.$$

Il faut donc que cette somme soit grande!

Exemple: Lagged-Fibonacci

$$x_n = (\pm x_{n-r} \pm x_{n-k}) \bmod m.$$

Autres bornes.

Proposition. Si $i \in I$ lorsque $a_{k-i} \neq 0$ (avec $a_0 = -1$), alors

$$\ell_I^2 \leq 1 + a_1^2 + \cdots + a_k^2.$$

Il faut donc que cette somme soit grande!

Exemple: Lagged-Fibonacci

$$x_n = (\pm x_{n-r} \pm x_{n-k}) \bmod m.$$

Pour $I = \{0, k-r, k\}$, on a $1/\ell_I \geq 1/\sqrt{3} \approx .577$.
Les vecteurs $(u_n, u_{n+k-r}, u_{n+k})$ sont tous contenus dans deux plans!

MRGs avec retenue ("carry")

$$
\begin{aligned}
x_n &= (a_1 x_{n-1} + \cdots + a_k x_{n-k} + c_{n-1}) d \bmod b, \\
c_n &= \lfloor (a_0 x_n + a_1 x_{n-1} + \cdots + a_k x_{n-k} + c_{n-1})/b \rfloor, \\
u_n &= \sum_{\ell=1}^{\infty} x_{n+\ell-1} b^{-\ell},
\end{aligned}
$$

où $d = (-a_0)^{-1} \bmod b$, i.e., $(-a_0 d) \bmod b = 1$.

MRGs avec retenue ("carry")

$$
\begin{aligned}
x_n &= (a_1 x_{n-1} + \cdots + a_k x_{n-k} + c_{n-1}) d \bmod b, \\
c_n &= \lfloor (a_0 x_n + a_1 x_{n-1} + \cdots + a_k x_{n-k} + c_{n-1})/b \rfloor, \\
u_n &= \sum_{\ell=1}^{\infty} x_{n+\ell-1} b^{-\ell},
\end{aligned}
$$

où $d = (-a_0)^{-1} \bmod b$, i.e., $(-a_0 d) \bmod b = 1$.

Équivalent à un LCG avec $m = a_0 + a_1 b + \cdots + a_k b^k$ et $a = b^{-1} \bmod m$.

On peut montrer que si $\{j : a_j \neq 0\} \subseteq I$, alors $\ell_I \leq a_0^2 + \cdots + a_k^2$.

MRGs avec retenue ("carry")

$$
\begin{aligned}
x_n &= (a_1 x_{n-1} + \cdots + a_k x_{n-k} + c_{n-1}) d \bmod b, \\
c_n &= \lfloor (a_0 x_n + a_1 x_{n-1} + \cdots + a_k x_{n-k} + c_{n-1})/b \rfloor, \\
u_n &= \sum_{\ell=1}^{\infty} x_{n+\ell-1} b^{-\ell},
\end{aligned}
$$

où $d = (-a_0)^{-1} \bmod b$, i.e., $(-a_0 d) \bmod b = 1$.

Équivalent à un LCG avec $m = a_0 + a_1 b + \cdots + a_k b^k$ et $a = b^{-1} \bmod m$.

On peut montrer que si $\{j : a_j \neq 0\} \subseteq I$, alors $\ell_I \leq a_0^2 + \cdots + a_k^2$.

Générateurs add-with-carry et subtract-with-borrow (Marsaglia et Zaman 1991):
$-a_0 = \pm a_r = \pm a_k = 1$ pour $0 < r < q$ et les autres a_j sont nuls.

MRGs avec retenue ("carry")

$$
\begin{aligned}
x_n &= (a_1 x_{n-1} + \cdots + a_k x_{n-k} + c_{n-1}) d \bmod b, \\
c_n &= \lfloor (a_0 x_n + a_1 x_{n-1} + \cdots + a_k x_{n-k} + c_{n-1})/b \rfloor, \\
u_n &= \sum_{\ell=1}^{\infty} x_{n+\ell-1} b^{-\ell},
\end{aligned}
$$

où $d = (-a_0)^{-1} \bmod b$, i.e., $(-a_0 d) \bmod b = 1$.

Équivalent à un LCG avec $m = a_0 + a_1 b + \cdots + a_k b^k$ et $a = b^{-1} \bmod m$.

On peut montrer que si $\{j : a_j \neq 0\} \subseteq I$, alors $\ell_I \leq a_0^2 + \cdots + a_k^2$.

Générateurs add-with-carry et subtract-with-borrow (Marsaglia et Zaman 1991):
$-a_0 = \pm a_r = \pm a_k = 1$ pour $0 < r < q$ et les autres a_j sont nuls.

Pour $I = \{0, r, k\}$, on a $\ell_I \leq \sqrt{3}$. Tous les vecteurs de la forme (u_n, u_{n+r}, u_{n+k}) sont dans seulement deux plans, distancés de $1/\sqrt{3}$.

MRGs avec retenue ("carry")

$$
\begin{aligned}
x_n &= (a_1 x_{n-1} + \cdots + a_k x_{n-k} + c_{n-1}) d \bmod b, \\
c_n &= \lfloor (a_0 x_n + a_1 x_{n-1} + \cdots + a_k x_{n-k} + c_{n-1})/b \rfloor, \\
u_n &= \sum_{\ell=1}^{\infty} x_{n+\ell-1} b^{-\ell},
\end{aligned}
$$

où $d = (-a_0)^{-1} \bmod b$, i.e., $(-a_0 d) \bmod b = 1$.

Équivalent à un LCG avec $m = a_0 + a_1 b + \cdots + a_k b^k$ et $a = b^{-1} \bmod m$.

On peut montrer que si $\{j : a_j \neq 0\} \subseteq I$, alors $\ell_I \leq a_0^2 + \cdots + a_k^2$.

Générateurs add-with-carry et subtract-with-borrow (Marsaglia et Zaman 1991):
$-a_0 = \pm a_r = \pm a_k = 1$ pour $0 < r < q$ et les autres a_j sont nuls.

Pour $I = \{0, r, k\}$, on a $\ell_I \leq \sqrt{3}$. Tous les vecteurs de la forme (u_n, u_{n+r}, u_{n+k}) sont dans seulement deux plans, distancés de $1/\sqrt{3}$.

Correctif proposé: sauter plusieurs valeurs après chaque bloc de k.
Mieux: ne pas utiliser.

Mise en oeuvre efficace

Il faut calculer $ax \bmod m$ pour de grands m.

Mise en oeuvre efficace

Il faut calculer $ax \bmod m$ pour de grands m.

Factorisation approximative.
Valide si ($a^2 < m$) ou ($a = \lfloor m/i \rfloor$ où $i^2 < m$). Calculs sur des entiers.
Précalculer $q = \lfloor m/a \rfloor$ et $r = m \bmod a$. Ensuite,

$$y = \lfloor x/q \rfloor; \quad x = a(x - yq) - yr; \quad \text{if } x < 0 \text{ then } x = x + m.$$

Mise en oeuvre efficace

Il faut calculer $ax \bmod m$ pour de grands m.

Factorisation approximative.
Valide si ($a^2 < m$) ou ($a = \lfloor m/i \rfloor$ où $i^2 < m$). Calculs sur des entiers.
Précalculer $q = \lfloor m/a \rfloor$ et $r = m \bmod a$. Ensuite,

$$y = \lfloor x/q \rfloor; \quad x = a(x - yq) - yr; \quad \text{if } x < 0 \text{ then } x = x + m.$$

Justification:

$$\begin{aligned} ax \bmod m &= (ax - \lfloor x/q_j \rfloor m) \bmod m \\ &= (ax - \lfloor x/q \rfloor (aq + r)) \bmod m \\ &= (a(x - \lfloor x/q \rfloor q) - \lfloor x/q \rfloor r) \bmod m \\ &= (a(x \bmod q) - \lfloor x/q \rfloor r) \bmod m. \end{aligned}$$

Toutes les quantités intermédiaires demeurent entre $-m$ et m.

Calculs en point flottant, double précision.
Valide si $am < 2^{53}$.

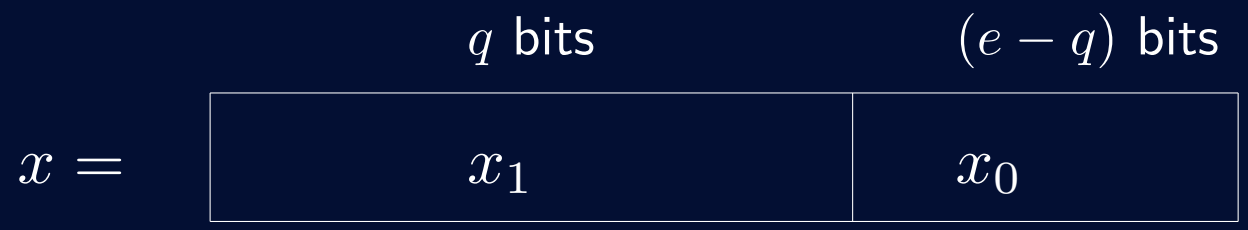
double $m, a, x, y;$ int $k;$

$y = a * x; \quad k = \lfloor y/m \rfloor; \quad x = y - k * m;$

Décomposition en puissances de 2.

Supposons que $a = \pm 2^q \pm 2^r$ et $m = 2^e - h$ pour h petit.
(Wu 1997 pour $h = 1$; L'Ecuyer et Simard 1999 pour $h > 1$.)

Pour calculer $y = 2^q x \bmod m$, décomposer $x = x_0 + 2^{e-q} x_1$;



Pour $h \geq 1$,

$$y = 2^q(x_0 + 2^{e-q}x_1) \bmod (2^e - h) = (2^q x_0 + h x_1) \bmod (2^e - h),$$

car
$2^q(2^{e-q}x_1) \bmod (2^e - h) = (2^e - h + h)x_1 \bmod (2^e - h) = h x_1 \bmod (2^e - h)$.

Décomposition en puissances de 2.

Supposons que $a = \pm 2^q \pm 2^r$ et $m = 2^e - h$ pour h petit.
(Wu 1997 pour $h = 1$; L'Ecuyer et Simard 1999 pour $h > 1$.)

Pour calculer $y = 2^q x \bmod m$, décomposer $x = x_0 + 2^{e-q} x_1$;

	q bits	$(e-q)$ bits
$x =$	x_1	x_0

Pour $h \geq 1$,

$$y = 2^q(x_0 + 2^{e-q}x_1) \bmod (2^e - h) = (2^q x_0 + h x_1) \bmod (2^e - h),$$

car
$2^q(2^{e-q}x_1) \bmod (2^e - h) = (2^e - h + h)x_1 \bmod (2^e - h) = hx_1 \bmod (2^e - h)$.

Si $h < 2^q$ et $h(2^q - (h+1)2^{-e+q}) < m$, chaque terme est $< m$.
Operation modulo: soustraire m si la somme est $\geq m$.

Pour $h = 1$, on obtient y en permutant x_0 et x_1.

Coefficients égaux.

Par exemple, prendre tous les a_j non nuls égaux à a (Deng et Xu 2002).

Alors, $x_n = a(x_{n-i_1} + \cdots + x_{n-k}) \bmod m$. Une seule multiplication.

Coefficients égaux.
Par exemple, prendre tous les a_j non nuls égaux à a (Deng et Xu 2002).
Alors, $x_n = a(x_{n-i_1} + \cdots + x_{n-k}) \bmod m$. Une seule multiplication.

Lagged-Fibonacci (très utilisé, mais mauvaise idée):

$$x_n = (\pm x_{n-r} \pm x_{n-k}) \bmod m.$$

Tous les vecteurs $(u_n, u_{n+k-r}, u_{n+k})$ sont dans seulemnt deux plans!

Coefficients égaux.
Par exemple, prendre tous les a_j non nuls égaux à a (Deng et Xu 2002).
Alors, $x_n = a(x_{n-i_1} + \cdots + x_{n-k}) \bmod m$. Une seule multiplication.

Lagged-Fibonacci (très utilisé, mais mauvaise idée):

$$x_n = (\pm x_{n-r} \pm x_{n-k}) \bmod m.$$

Tous les vecteurs $(u_n, u_{n+k-r}, u_{n+k})$ sont dans seulemnt deux plans!

Même problème avec add-with-carry et subtract-with-borrow.

Erreur fréquente: croire qu'on est ok si la période est assez longue...

Coefficients égaux.
Par exemple, prendre tous les a_j non nuls égaux à a (Deng et Xu 2002).
Alors, $x_n = a(x_{n-i_1} + \cdots + x_{n-k}) \bmod m$. Une seule multiplication.

Lagged-Fibonacci (très utilisé, mais mauvaise idée):

$$x_n = (\pm x_{n-r} \pm x_{n-k}) \bmod m.$$

Tous les vecteurs $(u_n, u_{n+k-r}, u_{n+k})$ sont dans seulemnt deux plans!

Même problème avec add-with-carry et subtract-with-borrow.

Erreur fréquente: croire qu'on est ok si la période est assez longue...

Des variantes qui sautent des valeurs sont recommandées par Luscher (1994) et Knuth (1997).

Mais pas très efficace...

MRGs combinés. Considérons deux [ou plusieurs...] MRGs évoluant en parallèle:

$$\begin{aligned} x_{1,n} &= (a_{1,1}x_{1,n-1} + \cdots + a_{1,k}x_{1,n-k}) \bmod m_1, \\ x_{2,n} &= (a_{2,1}x_{2,n-1} + \cdots + a_{2,k}x_{2,n-k}) \bmod m_2. \end{aligned}$$

On définit les deux combinaisons:

$$\begin{aligned} z_n &:= (x_{1,n} - x_{2,n}) \bmod m_1; \qquad & u_n &:= z_n/m_1; \\ w_n &:= (x_{1,n}/m_1 - x_{2,n}/m_2) \bmod 1. \end{aligned}$$

MRGs combinés. Considérons deux [ou plusieurs...] MRGs évoluant en parallèle:

$$\begin{aligned} x_{1,n} &= (a_{1,1}x_{1,n-1} + \cdots + a_{1,k}x_{1,n-k}) \bmod m_1, \\ x_{2,n} &= (a_{2,1}x_{2,n-1} + \cdots + a_{2,k}x_{2,n-k}) \bmod m_2. \end{aligned}$$

On définit les deux combinaisons:

$$\begin{aligned} z_n &:= (x_{1,n} - x_{2,n}) \bmod m_1; \qquad & u_n &:= z_n/m_1; \\ w_n &:= (x_{1,n}/m_1 - x_{2,n}/m_2) \bmod 1. \end{aligned}$$

La suite $\{w_n,\, n \geq 0\}$ est la sortie d'un autre MRG, de module $m = m_1 m_2$, et $\{u_n,\, n \geq 0\}$ est presque la même suite si m_1 et m_2 sont proches. Peut atteindre la période $(m_1^k - 1)(m_2^k - 1)/2$.

MRGs combinés. Considérons deux [ou plusieurs...] MRGs évoluant en parallèle:

$$\begin{aligned} x_{1,n} &= (a_{1,1}x_{1,n-1} + \cdots + a_{1,k}x_{1,n-k}) \bmod m_1, \\ x_{2,n} &= (a_{2,1}x_{2,n-1} + \cdots + a_{2,k}x_{2,n-k}) \bmod m_2. \end{aligned}$$

On définit les deux combinaisons:

$$\begin{aligned} z_n &:= (x_{1,n} - x_{2,n}) \bmod m_1; \qquad & u_n &:= z_n/m_1; \\ w_n &:= (x_{1,n}/m_1 - x_{2,n}/m_2) \bmod 1. \end{aligned}$$

La suite $\{w_n,\, n \geq 0\}$ est la sortie d'un autre MRG, de module $m = m_1 m_2$, et $\{u_n,\, n \geq 0\}$ est presque la même suite si m_1 et m_2 sont proches. Peut atteindre la période $(m_1^k - 1)(m_2^k - 1)/2$.

Permet d'implanter efficacement un MRG ayant un grand m et plusieurs grands coefficients non nuls.

MRGs combinés. Considérons deux [ou plusieurs...] MRGs évoluant en parallèle:

$$\begin{aligned} x_{1,n} &= (a_{1,1}x_{1,n-1} + \cdots + a_{1,k}x_{1,n-k}) \bmod m_1, \\ x_{2,n} &= (a_{2,1}x_{2,n-1} + \cdots + a_{2,k}x_{2,n-k}) \bmod m_2. \end{aligned}$$

On définit les deux combinaisons:

$$\begin{aligned} z_n &:= (x_{1,n} - x_{2,n}) \bmod m_1; \qquad & u_n &:= z_n/m_1; \\ w_n &:= (x_{1,n}/m_1 - x_{2,n}/m_2) \bmod 1. \end{aligned}$$

La suite $\{w_n,\, n \geq 0\}$ est la sortie d'un autre MRG, de module $m = m_1 m_2$, et $\{u_n,\, n \geq 0\}$ est presque la même suite si m_1 et m_2 sont proches. Peut atteindre la période $(m_1^k - 1)(m_2^k - 1)/2$.

Permet d'implanter efficacement un MRG ayant un grand m et plusieurs grands coefficients non nuls.

Tableaux de paramètres: L'Ecuyer (1999); L'Ecuyer et Touzin (2000).

MRG32k3a

$J = 2$, $k = 3$,
$m_1 = 2^{32} - 209$, $a_{11} = 0$, $a_{12} = 1403580$, $a_{13} = -810728$,
$m_2 = 2^{32} - 22853$, $a_{21} = 527612$, $a_{22} = 0$, $a_{23} = -1370589$.

MRG32k3a

$J = 2$, $k = 3$,
$m_1 = 2^{32} - 209$, $a_{11} = 0$, $a_{12} = 1403580$, $a_{13} = -810728$,
$m_2 = 2^{32} - 22853$, $a_{21} = 527612$, $a_{22} = 0$, $a_{23} = -1370589$.

Combination: $z_n = (x_{1,n} - x_{2,n}) \bmod m_1$.

MRG32k3a

$J = 2$, $k = 3$,
$m_1 = 2^{32} - 209$, $a_{11} = 0$, $a_{12} = 1403580$, $a_{13} = -810728$,
$m_2 = 2^{32} - 22853$, $a_{21} = 527612$, $a_{22} = 0$, $a_{23} = -1370589$.

Combination: $z_n = (x_{1,n} - x_{2,n}) \bmod m_1$.

MRG correspondant: $k = 3$,
$m = m_1 m_2 = 18446645023178547541$,
$a_1 = 18169668471252892557$,
$a_2 = 31868605061992738 33$,
$a_3 = 87386132643982226 22$.

MRG32k3a

$J = 2$, $k = 3$,
$m_1 = 2^{32} - 209$, $a_{11} = 0$, $a_{12} = 1403580$, $a_{13} = -810728$,
$m_2 = 2^{32} - 22853$, $a_{21} = 527612$, $a_{22} = 0$, $a_{23} = -1370589$.

Combination: $z_n = (x_{1,n} - x_{2,n}) \bmod m_1$.

MRG correspondant: $k = 3$,
$m = m_1 m_2 = 18446645023178547541$,
$a_1 = 18169668471252892557$,
$a_2 = 3186860506199273833$,
$a_3 = 873861326439822622$.

Période $\rho = (m_1^3 - 1)(m_2^3 - 1)/2 \approx 2^{191}$.

```
#define norm 2.328306549295728e-10  /* 1/(m1+1) */
#define m1   4294967087.0
#define m2   4294944443.0
#define a12     1403580.0
#define a13n     810728.0
#define a21      527612.0
#define a23n    1370589.0

double  s10, s11, s12, s20, s21, s22;

double MRG32k3a ()
   {
   long   k;
   double p1, p2;
   /* Component 1 */
   p1 = a12 * s11 - a13n * s10;
   k = p1 / m1;   p1 -= k * m1;   if (p1 < 0.0) p1 += m1;
   s10 = s11;   s11 = s12;   s12 = p1;
   /* Component 2 */
   p2 = a21 * s22 - a23n * s20;
   k  = p2 / m2;  p2 -= k * m2;   if (p2 < 0.0) p2 += m2;
   s20 = s21;   s21 = s22;   s22 = p2;
   /* Combination */
   if (p1 <= p2) return ((p1 - p2 + m1) * norm);
   else return ((p1 - p2) * norm);
   }
```

GVAs basés sur des récurrences linéaires dans $\mathbb{F}_2$

Récurrence linéaire matricielle sur $\mathbb{F}_2 \quad (\equiv \{0,1\},\ \bmod 2)$:

$$\begin{aligned}
\mathbf{x}_n &= \mathbf{A}\mathbf{x}_{n-1} \qquad \text{(vecteur d'état, } k \text{ bits)}, \\
\mathbf{y}_n &= \mathbf{B}\mathbf{x}_n \qquad \text{(vecteur de sortie, } w \text{ bits)}, \\
u_n &= \sum_{j=1}^{w} y_{n,j-1} 2^{-j} \;=\; .y_{n,0}\, y_{n,1}\, y_{n,2} \cdots \qquad \text{(sortie)}.
\end{aligned}$$

GVAs basés sur des récurrences linéaires dans $\mathbb{F}_2$

Récurrence linéaire matricielle sur $\mathbb{F}_2 \quad (\equiv \{0,1\},\ \bmod 2)$:

$$\begin{aligned} \mathbf{x}_n &= \mathbf{A}\mathbf{x}_{n-1} \qquad \text{(vecteur d'état, } k \text{ bits)}, \\ \mathbf{y}_n &= \mathbf{B}\mathbf{x}_n \qquad \text{(vecteur de sortie, } w \text{ bits)}, \\ u_n &= \sum_{j=1}^{w} y_{n,j-1} 2^{-j} = .y_{n,0}\, y_{n,1}\, y_{n,2} \cdots \qquad \text{(sortie)}. \end{aligned}$$

Chaque coordonnée de $\mathbf{x}_n$ et de $\mathbf{y}_n$ suit la récurrence linéaire

$$x_{n,j} = (\alpha_1 x_{n-1,j} + \cdots + \alpha_k x_{n-k,j}),$$

de polynôme caractéristique

$$P(z) = z^k - \alpha_1 z^{k-1} - \cdots - \alpha_{k-1} z - \alpha_k = \det(\mathbf{A} - z\mathbf{I}).$$

La période maximale $\rho = 2^k - 1$ est atteinte ssi $P(z)$ est primitif sur $\mathbb{F}_2$.

Avec un choix astucieux de $\mathbf{A}$, le calcul de transition se fait avec des décalages, xor, and, masques, etc., sur des blocs de bits. Très rapide.

Cas particuliers: Tausworthe, "linear feedback shift register" (LFSR), GFSR, twisted GFSR, Mersenne twister, WELL, xorshift, polynomial LCG, etc.

Avec un choix astucieux de $\mathbf{A}$, le calcul de transition se fait avec des décalages, xor, and, masques, etc., sur des blocs de bits. Très rapide.

Cas particuliers: Tausworthe, "linear feedback shift register" (LFSR), GFSR, twisted GFSR, Mersenne twister, WELL, xorshift, polynomial LCG, etc.

Pour sauter en avant:

$$\mathbf{x}_{n+\nu} = \underbrace{(\mathbf{A}^\nu \bmod 2)}_{\text{précalculé}} \mathbf{x}_n \bmod 2.$$

Haramoto, L'Ecuyer, Matsumoto, Nishimura, Panneton (2006) proposent une méthode plus efficace pour les grandes valeurs de k.

Equidistribution.

Pour $j = 1, \ldots, t$, partitionnons l'axe j de $[0,1)^t$ en 2^{q_j} intervalles égaux. Cela donne 2^{k-q} boîtes rectangulaires, où $q = k - q_1 - \cdots - q_t$.

Equidistribution.

Pour $j = 1, \ldots, t$, partitionnons l'axe j de $[0,1)^t$ en 2^{q_j} intervalles égaux. Cela donne 2^{k-q} boîtes rectangulaires, où $q = k - q_1 - \cdots - q_t$.

Si chaque boîte contient exactement 2^q points de Ψ_t, on dit que Ψ_t est $\mathbf{q}$-équidistribué pour $\mathbf{q} = (q_1, \ldots, q_t)$.

Equidistribution.

Pour $j = 1, \ldots, t$, partitionnons l'axe j de $[0,1)^t$ en 2^{q_j} intervalles égaux. Cela donne 2^{k-q} boîtes rectangulaires, où $q = k - q_1 - \cdots - q_t$.

Si chaque boîte contient exactement 2^q points de Ψ_t, on dit que Ψ_t est $\mathbf{q}$-équidistribué pour $\mathbf{q} = (q_1, \ldots, q_t)$.

Veut dire que chaque vecteur en t-dim., avec q_j bits de précision pour chaque coordonnée j, apparaît le même nombre de fois, car la boîte dans laquelle tombe $(u_0, \ldots, u_{t-1})$ est déterminée par les q_1 premiers bits de u_0 (ou de $\mathbf{y}_0$), les q_2 premiers bits de u_1 (ou de $\mathbf{y}_1$), . . . , et les q_t premiers bits de u_{t-1} (ou de $\mathbf{y}_{t-1}$).

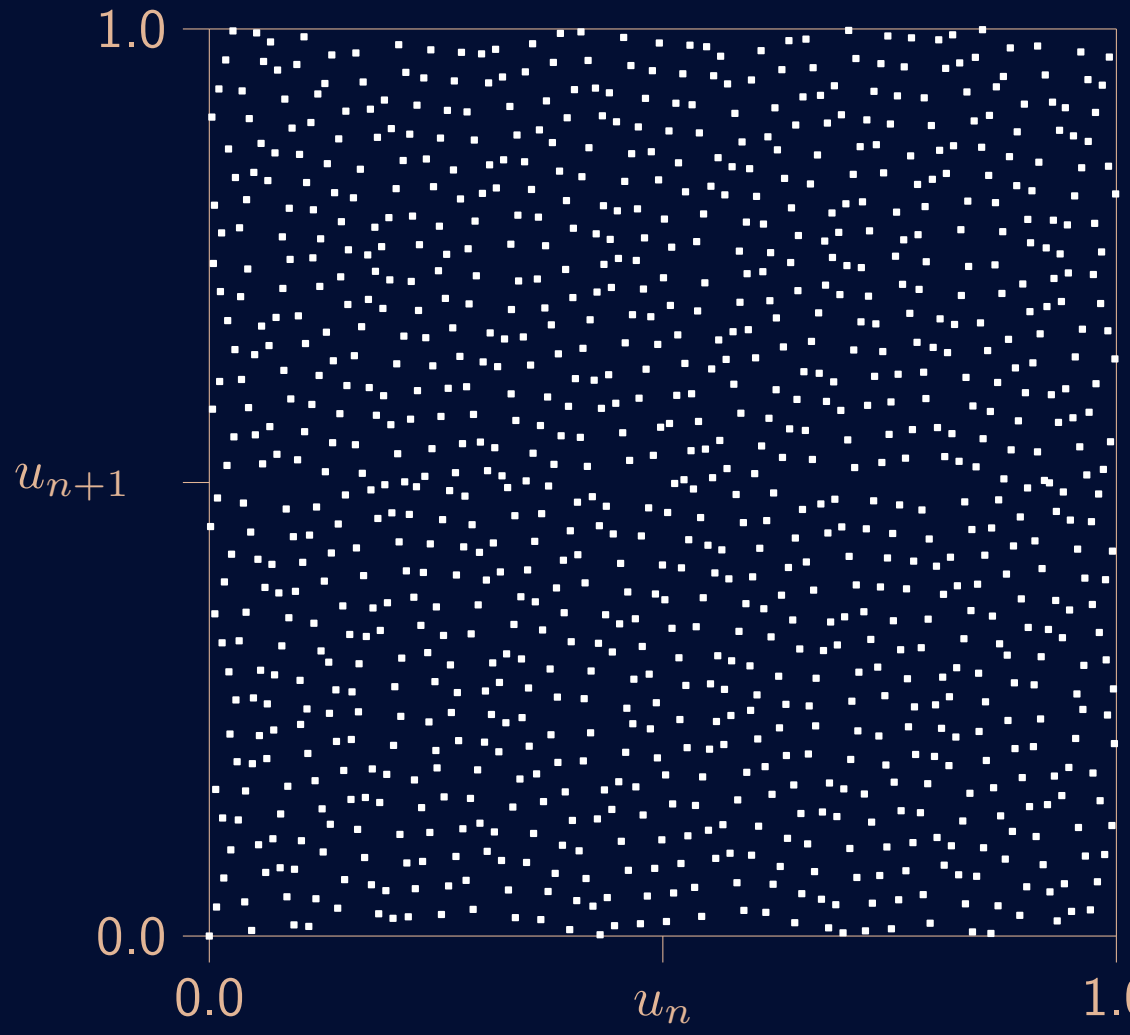
Equidistribution.

Pour $j = 1, \ldots, t$, partitionnons l'axe j de $[0,1)^t$ en 2^{q_j} intervalles égaux. Cela donne 2^{k-q} boîtes rectangulaires, où $q = k - q_1 - \cdots - q_t$.

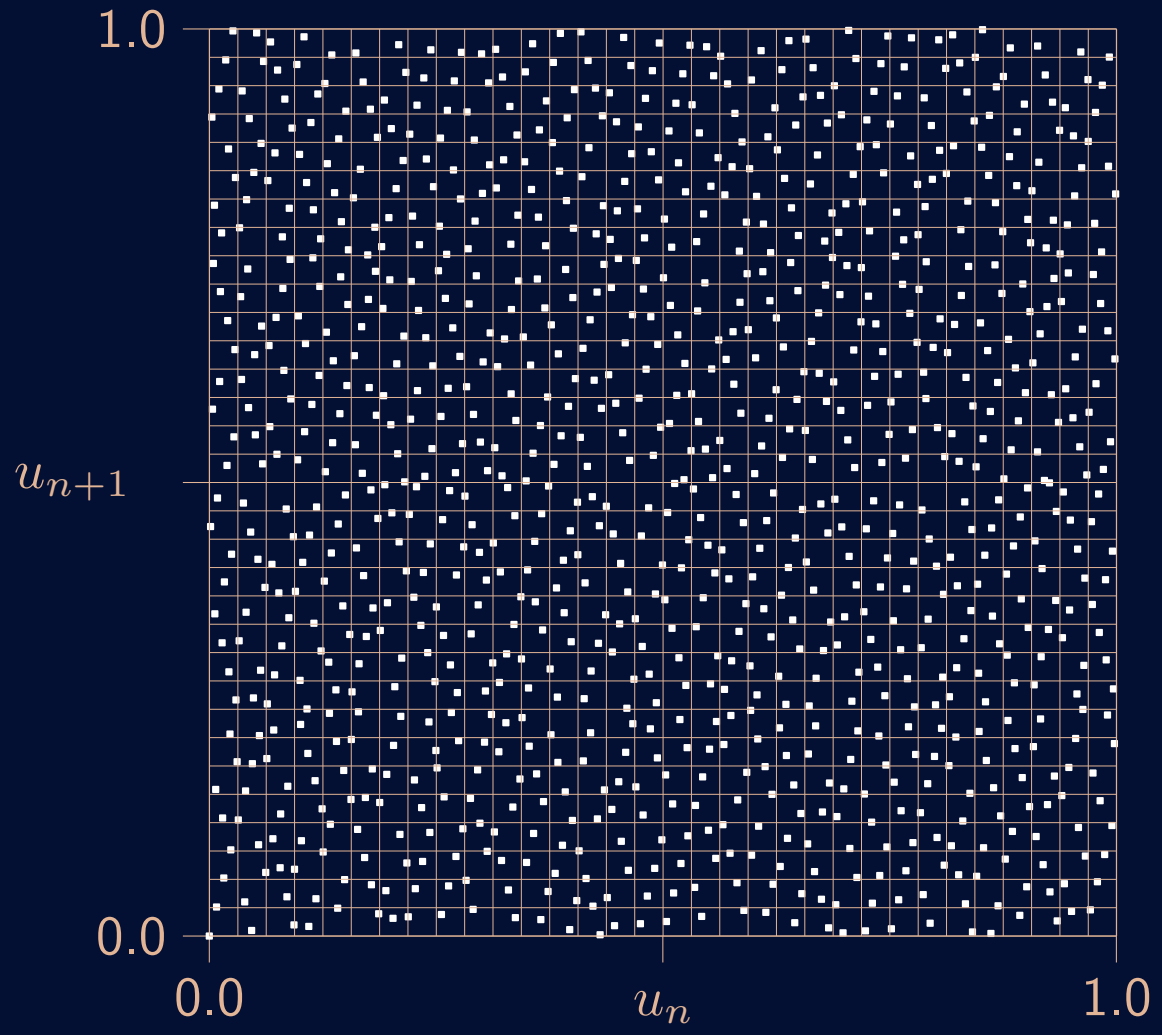
Si chaque boîte contient exactement 2^q points de Ψ_t, on dit que Ψ_t est $\mathbf{q}$-équidistribué pour $\mathbf{q} = (q_1, \ldots, q_t)$.

Veut dire que chaque vecteur en t-dim., avec q_j bits de précision pour chaque coordonnée j, apparaît le même nombre de fois, car la boîte dans laquelle tombe $(u_0, \ldots, u_{t-1})$ est déterminée par les q_1 premiers bits de u_0 (ou de $\mathbf{y}_0$), les q_2 premiers bits de u_1 (ou de $\mathbf{y}_1$), . . . , et les q_t premiers bits de u_{t-1} (ou de $\mathbf{y}_{t-1}$).

Si Ψ_t est $(\ell, \ell, \ldots, \ell)$-équidistribué,
on dit qu'il est t-distribué avec ℓ bits de précision.

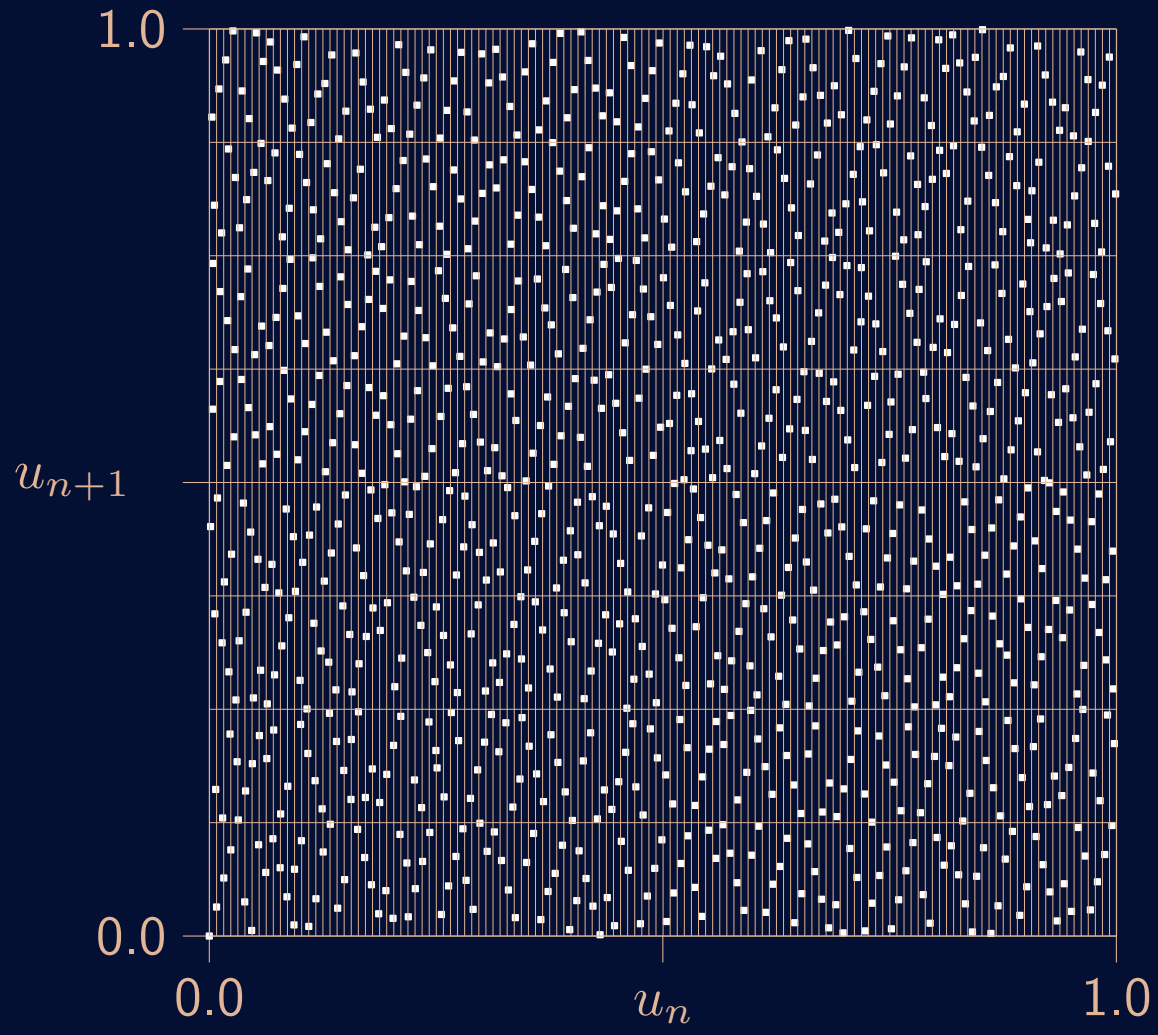


Exemple jouet: LFSR (Tausworthe combiné) avec $n = 1024 = 2^{10}$.



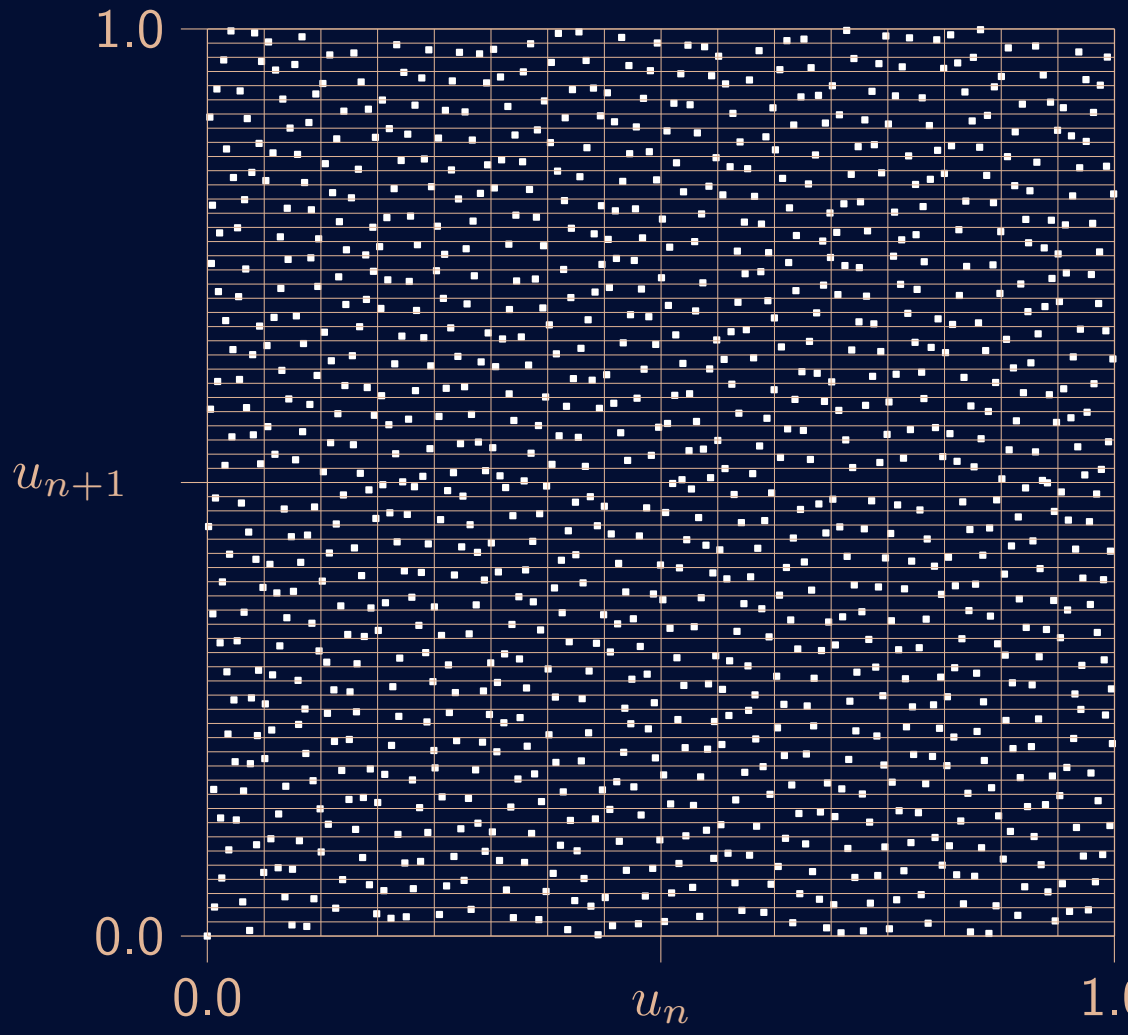
Exemple jouet: LFSR (Tausworthe combiné) avec $n = 1024 = 2^{10}$.

Vrai aussi en 3, 4, 5, ... 10 dimensions.



Exemple jouet: LFSR (Tausworthe combiné) avec $n = 1024 = 2^{10}$.

128×8



Exemple jouet: LFSR (Tausworthe combiné) avec $n = 1024 = 2^{10}$.

16×64

Calcul:

On peut exprimer les $k - q$ bits pertinents comme $\mathbf{M_q x}_0$ pour une matrice $\mathbf{M_q}$.
En effet, $\mathbf{y}_n = \mathbf{BA}^n \mathbf{x}_0$, de sorte que $\mathbf{M_q}$ contiendra les
q_1 premières lignes de $\mathbf{BA}^0 = \mathbf{B}$, suivies des
q_2 premières lignes de $\mathbf{BA}^1$,
. . . , et finalement les
q_t premières lignes de $\mathbf{BA}^{t-1}$.

Ψ_t est $\mathbf{q}$-équidistribué ssi la transformation linéaire est surjective (la dimension du noyau de la transformation est q), ssi $\mathbf{M_q}$ est de plein rang $k - q$.

Écart de résolution pour un ensemble d'indices $I = \{i_1, i_2, \ldots, i_t\}$:

$$\delta_I = \min(\lfloor k/t \rfloor, w) - \max\{\ell : \Psi_I \text{ est } (\ell, \ldots, \ell)\text{-équidist.}\}.$$

Mesures d'uniformité potentielles:

$$\Delta_{\mathcal{J}} = \max_{I \in \mathcal{J}} \delta_I \quad \text{ou} \quad V_{\mathcal{J}} = \sum_{I \in \mathcal{J}} \delta_I$$

où $\mathcal{J}$ est une classe donnée d'ensembles I.

Écart de résolution pour un ensemble d'indices $I = \{i_1, i_2, \ldots, i_t\}$:

$$\delta_I = \min(\lfloor k/t \rfloor, w) - \max\{\ell : \Psi_I \text{ est } (\ell, \ldots, \ell)\text{-équidist.}\}.$$

Mesures d'uniformité potentielles:

$$\Delta_{\mathcal{J}} = \max_{I \in \mathcal{J}} \delta_I \quad \text{ou} \quad V_{\mathcal{J}} = \sum_{I \in \mathcal{J}} \delta_I$$

où $\mathcal{J}$ est une classe donnée d'ensembles I.
Le choix de $\mathcal{J}$ est arbitraire. Question de compromis.

Écart de résolution pour un ensemble d'indices $I = \{i_1, i_2, \ldots, i_t\}$:

$$\delta_I = \min(\lfloor k/t \rfloor, w) - \max\{\ell : \Psi_I \text{ est } (\ell, \ldots, \ell)\text{-équidist.}\}.$$

Mesures d'uniformité potentielles:

$$\Delta_{\mathcal{J}} = \max_{I \in \mathcal{J}} \delta_I \quad \text{ou} \quad V_{\mathcal{J}} = \sum_{I \in \mathcal{J}} \delta_I$$

où $\mathcal{J}$ est une classe donnée d'ensembles I.
Le choix de $\mathcal{J}$ est arbitraire. Question de compromis.

Dans ce qui vient, on prendra $w = 32$ et

$$V = \sum_{\ell=1}^{w} \left(\lfloor k/\ell \rfloor - \max\{t : \Psi_t \text{ est } (t, \ell)\text{-équidist.}\} \right).$$

Écart de résolution pour un ensemble d'indices $I = \{i_1, i_2, \ldots, i_t\}$:

$$\delta_I = \min(\lfloor k/t \rfloor, w) - \max\{\ell : \Psi_I \text{ est } (\ell, \ldots, \ell)\text{-équidist.}\}.$$

Mesures d'uniformité potentielles:

$$\Delta_{\mathcal{J}} = \max_{I \in \mathcal{J}} \delta_I \quad \text{ou} \quad V_{\mathcal{J}} = \sum_{I \in \mathcal{J}} \delta_I$$

où $\mathcal{J}$ est une classe donnée d'ensembles I.
Le choix de $\mathcal{J}$ est arbitraire. Question de compromis.

Dans ce qui vient, on prendra $w = 32$ et

$$V = \sum_{\ell=1}^{w} \left(\lfloor k/\ell \rfloor - \max\{t : \Psi_t \text{ est } (t, \ell)\text{-équidist.}\} \right).$$

On veut aussi que le nombre N_1 de coefficients non nuls α_j's soit proche de $k/2$.

Générateur de Tausworthe (ou LFSR)

(Tausworthe 1965):

$$
\begin{aligned}
x_n &= (a_1 x_{n-1} + \cdots + a_k x_{n-k}) \bmod 2, \\
u_n &= \sum_{j=1}^{\infty} x_{n\nu+j-1} 2^{-j} = .x_{n\nu} x_{n\nu+1} x_{n\nu+2} \cdots \\
\mathbf{A} &= \begin{pmatrix} & 1 & & \\ & & \ddots & \\ & & & 1 \\ a_k & a_{k-1} & \cdots & a_1 \end{pmatrix}^{\nu} \quad \text{et } B = I.
\end{aligned}
$$

Période max. $\rho = 2^k - 1$ ssi $Q(z) = z^k - a_1 z^{k-1} - \cdots - a_{k-1} z - a_k$ est primitif et $\text{pgcd}(\nu, 2^k - 1) = 1$.

Générateur de Tausworthe (ou LFSR)

(Tausworthe 1965):

$$
\begin{aligned}
x_n &= (a_1 x_{n-1} + \cdots + a_k x_{n-k}) \bmod 2, \\
u_n &= \sum_{j=1}^{\infty} x_{n\nu+j-1} 2^{-j} = .x_{n\nu} x_{n\nu+1} x_{n\nu+2} \cdots \\
\mathbf{A} &= \begin{pmatrix} & 1 & & \\ & & \ddots & \\ & & & 1 \\ a_k & a_{k-1} & \cdots & a_1 \end{pmatrix}^{\nu} \quad \text{et } B = I.
\end{aligned}
$$

Période max. $\rho = 2^k - 1$ ssi $Q(z) = z^k - a_1 z^{k-1} - \cdots - a_{k-1} z - a_k$ est primitif et $\mathrm{pgcd}(\nu, 2^k - 1) = 1$.

Trinôme: $Q(z) = z^k - a_r z^{k-r} - a_k$.
Implantation très rapide par des shifts, xors, masques, etc., si $\nu \leq r$ et $2r > k$.

Exemple: $w = 32$ bits, $k = 31$, $k - r = 6$, et $\nu = 18$.

La récurrence peut se calculer via:

$$\begin{aligned} \mathbf{y} &= (\mathbf{x}_{n-1} \ll 6) \oplus \mathbf{x}_{n-1}, \\ \mathbf{x}_n &= ((\mathbf{x}_{n-1} \text{ with last bit at 0}) \ll 18) \oplus (\mathbf{y} \gg 13). \end{aligned}$$

We illustrate this algorithm for a given initial state $\mathbf{x}_{n-1}$. The bits in orange are discarded. The state $\mathbf{x}_n$ after the transition is shown in red. It is the juxtaposition of the two blocks of bits in blue.

$\mathbf{x}_{n-1} =$ 00010100101001101100110110100101

100101 00101001101100110110100101

$\mathbf{y} =$ 00111101000101011010010011100101

$\mathbf{y} \gg 13 =$ 00111101000101011010010011100101

$\mathbf{x}_{n-1}$ 00010100101001101100110110100100

00010100101001101100110110100100

$\mathbf{x}_n =$ 00110110100100011110100010101101

Voir les notes pour l'algorithme général et les détails techniques.

Generalized feedback shift register (GFSR) (Lewis et Payne 1973):

$$
\begin{aligned}
\mathbf{v}_n &= (a_1\mathbf{v}_{n-1} + \cdots + a_r\mathbf{v}_{n-r}) \bmod 2 = (v_{n,0}, \ldots, v_{n,w-1})^{\mathsf{t}}, \\
\mathbf{y}_n &= \mathbf{v}_n, \\
\mathbf{A} &= \begin{pmatrix}
 & & & & I_w & & I_w \\
I_w & & & & & & \\
 & I_w & & & & & \\
 & & I_w & & & & \\
 & & & \ddots & & & \\
 & & & & I_w & &
\end{pmatrix}.
\end{aligned}
$$

si $P(z)$ est un trinôme (habituel).

Ici, $P(z) = z^r - a_1 z^{r-1} - \cdots - a_{r-1}z - a_r$ et période max. $= 2^r - 1$ même si l'état a rw bits.

Twisted GFSR (Matsumoto et Kurita 1992, 1994):

$$
\begin{aligned}
\mathbf{v}_n &= (\mathbf{v}_{n+m-r} + A_0 \mathbf{v}_{n-r}) \bmod 2 \\
\mathbf{y}_n &= \mathbf{v}_n \quad \text{ou} \quad \mathbf{y}_n = T\mathbf{v}_n, \\
\mathbf{A} &= \begin{pmatrix}
 & & & I_w & & A_0 \\
I_w & & & & & \\
 & I_w & & & & \\
 & & I_w & & & \\
 & & & \ddots & & \\
 & & & & I_w &
\end{pmatrix}.
\end{aligned}
$$

La période max. est $2^{rw} - 1$, atteinte ssi $Q(z^r + z^m)$ est primitif de degré rw, où Q est le polynôme caractéristique de $\mathbf{A}$.

Twisted GFSR (Matsumoto et Kurita 1992, 1994):

$$\begin{aligned}
\mathbf{v}_n &= (\mathbf{v}_{n+m-r} + A_0 \mathbf{v}_{n-r}) \bmod 2 \\
\mathbf{y}_n &= \mathbf{v}_n \quad \text{ou} \quad \mathbf{y}_n = T\mathbf{v}_n, \\
\mathbf{A} &= \begin{pmatrix}
 & & & I_w & & A_0 \\
I_w & & & & & \\
 & I_w & & & & \\
 & & I_w & & & \\
 & & & \ddots & & \\
 & & & & I_w &
\end{pmatrix}.
\end{aligned}$$

La période max. est $2^{rw} - 1$, atteinte ssi $Q(z^r + z^m)$ est primitif de degré rw, où Q est le polynôme caractéristique de $\mathbf{A}$.

La vedette: `TT800`, période de $2^{800} - 1$.

Mersenne Twister (Matsumoto et Nishimura 1998):

$$
\begin{aligned}
\mathbf{v}_n &= (\mathbf{v}_{n+m-r} + A(\mathbf{v}_{n-r}^{(w-p)} | \mathbf{v}_{n-r+1}^{(p)}) \bmod 2 \\
\mathbf{y}_n &= T\mathbf{v}_n, \\
\mathbf{A} &= \begin{pmatrix}
 & & & I_w & & A\ \mathrm{rot}_p(I) \\
I_w & & & & & \\
 & I_w & & & & \\
 & & I_w & & & \\
 & & & \ddots & & \\
 & & & & I_w &
\end{pmatrix}.
\end{aligned}
$$

La période max. est $2^{rw-p} - 1$.

Mersenne Twister (Matsumoto et Nishimura 1998):

$$
\begin{aligned}
\mathbf{v}_n &= (\mathbf{v}_{n+m-r} + A(\mathbf{v}_{n-r}^{(w-p)} | \mathbf{v}_{n-r+1}^{(p)}) \bmod 2 \\
\mathbf{y}_n &= T\mathbf{v}_n, \\
\mathbf{A} &= \begin{pmatrix}
 & & & I_w & & A\ \mathrm{rot}_p(I) \\
I_w & & & & & \\
 & I_w & & & & \\
 & & I_w & & & \\
 & & & \ddots & & \\
 & & & & I_w &
\end{pmatrix}.
\end{aligned}
$$

La période max. est $2^{rw-p} - 1$.

La vedette: `MT19937`, période de $2^{19937} - 1$.

Générateurs combinés sur $\mathbb{F}_2$

J générateurs $\mathbb{F}_2$-linéaires de paramètres $(k_j, w, \mathbf{A}_j, \mathbf{B}_j)$ et états $\mathbf{x}_{j,i}$. Output:

$$\begin{aligned} \mathbf{y}_n &= \mathbf{B}_1\mathbf{x}_{1,n} \oplus \cdots \oplus \mathbf{B}_J\mathbf{x}_{J,n}, \\ u_n &= \sum_{\ell=1}^{w} y_{n,\ell-1} 2^{-\ell}, \end{aligned}$$

Générateurs combinés sur $\mathbb{F}_2$

J générateurs $\mathbb{F}_2$-linéaires de paramètres $(k_j, w, \mathbf{A}_j, \mathbf{B}_j)$ et états $\mathbf{x}_{j,i}$. Output:

$$\begin{aligned} \mathbf{y}_n &= \mathbf{B}_1\mathbf{x}_{1,n} \oplus \cdots \oplus \mathbf{B}_J\mathbf{x}_{J,n}, \\ u_n &= \sum_{\ell=1}^{w} y_{n,\ell-1} 2^{-\ell}, \end{aligned}$$

Cette combinaison est équivalente à un générateur $\mathbb{F}_2$-linéaire ayant $k = k_1 + \cdots + k_J$, $\mathbf{A} = \mathsf{diag}(\mathbf{A}_1, \ldots, \mathbf{A}_J)$, et $\mathbf{B} = (\mathbf{B}_1, \ldots, \mathbf{B}_J)$.

Générateurs combinés sur $\mathbb{F}_2$

J générateurs $\mathbb{F}_2$-linéaires de paramètres $(k_j, w, \mathbf{A}_j, \mathbf{B}_j)$ et états $\mathbf{x}_{j,i}$. Output:

$$\begin{aligned} \mathbf{y}_n &= \mathbf{B}_1\mathbf{x}_{1,n} \oplus \cdots \oplus \mathbf{B}_J\mathbf{x}_{J,n}, \\ u_n &= \sum_{\ell=1}^{w} y_{n,\ell-1} 2^{-\ell}, \end{aligned}$$

Cette combinaison est équivalente à un générateur $\mathbb{F}_2$-linéaire ayant $k = k_1 + \cdots + k_J$, $\mathbf{A} = \mathsf{diag}(\mathbf{A}_1, \ldots, \mathbf{A}_J)$, et $\mathbf{B} = (\mathbf{B}_1, \ldots, \mathbf{B}_J)$.

Si on combine des LFSRs ayant des polynômes caractéristiques $P_j(z)$, le générateur combiné a comme polynôme caractéristique $P(z) = P_1(z) \cdots P_J(z)$ et sa période peut atteindre le produit des périodes.

Générateurs combinés sur $\mathbb{F}_2$

J générateurs $\mathbb{F}_2$-linéaires de paramètres $(k_j, w, \mathbf{A}_j, \mathbf{B}_j)$ et états $\mathbf{x}_{j,i}$. Output:

$$\begin{aligned} \mathbf{y}_n &= \mathbf{B}_1\mathbf{x}_{1,n} \oplus \cdots \oplus \mathbf{B}_J\mathbf{x}_{J,n}, \\ u_n &= \sum_{\ell=1}^{w} y_{n,\ell-1} 2^{-\ell}, \end{aligned}$$

Cette combinaison est équivalente à un générateur $\mathbb{F}_2$-linéaire ayant $k = k_1 + \cdots + k_J$, $\mathbf{A} = \mathsf{diag}(\mathbf{A}_1, \ldots, \mathbf{A}_J)$, et $\mathbf{B} = (\mathbf{B}_1, \ldots, \mathbf{B}_J)$.

Si on combine des LFSRs ayant des polynômes caractéristiques $P_j(z)$, le générateur combiné a comme polynôme caractéristique $P(z) = P_1(z) \cdots P_J(z)$ et sa période peut atteindre le produit des périodes.

En combinant des LFSR, TGFSR, ou Mersenne twister entre eux, on obtient des générateurs ayant de bien meilleures équidistributions.

Exemple: LFSR113.

```
unsigned long z1, z2, z3, z4;

double lfsr113 ()
   {           /* Generates numbers between 0 and 1. */
   unsigned long b;
   b  = (((z1 <<  6) ^ z1) >> 13);
   z1 = (((z1 & 4294967294) << 18) ^ b);
   b  = (((z2 <<  2) ^ z2) >> 27);
   z2 = (((z2 & 4294967288) <<  2) ^ b);
   b  = (((z3 << 13) ^ z3) >> 21);
   z3 = (((z3 & 4294967280) <<  7) ^ b);
   b  = (((z4 <<  3) ^ z4) >> 12);
   z4 = (((z4 & 4294967168) << 13) ^ b);
   return ((z1 ^ z2 ^ z3 ^ z4) * 2.3283064365387e-10);
   }
```

Tests statistiques empiriques

Hypothèse nulle $\mathcal{H}_0$: "$\{u_0, u_1, u_2, \ldots\}$ sont les réalisations de v.a. indép. $U(0,1)$".
On sait à l'avance que $\mathcal{H}_0$ est fausse, mais peut-on le détecter?

Tests statistiques empiriques

Hypothèse nulle $\mathcal{H}_0$: "$\{u_0, u_1, u_2, \ldots\}$ sont les réalisations de v.a. indép. $U(0,1)$".
On sait à l'avance que $\mathcal{H}_0$ est fausse, mais peut-on le détecter?

Test:
— On définit une v.a. T, fonction des u_i, dont la loi sous $\mathcal{H}_0$ est connue (approx.).
— On rejette $\mathcal{H}_0$ si T prend une valeur trop extrême par rapport à cette loi.
Si la valeur est "suspecte", on peut répéter le test plusiques fois.

Quels sont les meilleurs tests? Pas de réponse à cela.
Différents tests permettent de détecter différents types de défauts.

Tests statistiques empiriques

Hypothèse nulle $\mathcal{H}_0$: "$\{u_0, u_1, u_2, \ldots\}$ sont les réalisations de v.a. indép. $U(0,1)$".
On sait à l'avance que $\mathcal{H}_0$ est fausse, mais peut-on le détecter?

Test:
— On définit une v.a. T, fonction des u_i, dont la loi sous $\mathcal{H}_0$ est connue (approx.).
— On rejette $\mathcal{H}_0$ si T prend une valeur trop extrême par rapport à cette loi.
Si la valeur est "suspecte", on peut répéter le test plusiques fois.

Quels sont les meilleurs tests? Pas de réponse à cela.
Différents tests permettent de détecter différents types de défauts.

Idéal: le comportement de T ressemble à celui des v.a. qui nous intéressent dans nos simulations. Mais pas pratique...

Rêve: Construire un GPA qui passe tous les tests? Formellement impossible.

Tests statistiques empiriques

Hypothèse nulle $\mathcal{H}_0$: "$\{u_0, u_1, u_2, \ldots\}$ sont les réalisations de v.a. indép. $U(0,1)$".
On sait à l'avance que $\mathcal{H}_0$ est fausse, mais peut-on le détecter?

Test:
— On définit une v.a. T, fonction des u_i, dont la loi sous $\mathcal{H}_0$ est connue (approx.).
— On rejette $\mathcal{H}_0$ si T prend une valeur trop extrême par rapport à cette loi.
Si la valeur est "suspecte", on peut répéter le test plusiques fois.

Quels sont les meilleurs tests? Pas de réponse à cela.
Différents tests permettent de détecter différents types de défauts.

Idéal: le comportement de T ressemble à celui des v.a. qui nous intéressent dans nos simulations. Mais pas pratique...

Rêve: Construire un GPA qui passe tous les tests? Formellement impossible.

Compromis (heuristique): Se satisfaire d'un GPA qui passe les tests raisonnables.
Les tests échoués sont très difficiles à trouver et exécuter.
Formalisation: cadre de complexité algorithmique, populaire en cryptologie.

Exemple: Un test de collisions

On partitionne la boîte $[0,1)^t$ en $k = d^t$ boîtes cubiques de même taille.
On génère n points $(u_{ti}, \ldots, u_{ti+t-1})$ dans $[0,1)^t$, $i = 0, \ldots, n-1$.
Soit X_j le nombre de points dans la boîte j.
Nombre de collisions:

$$C = \sum_{j=0}^{k-1} \max(0,\, X_j - 1).$$

Sous $\mathcal{H}_0$, $C \approx$ Poisson de moyenne $\lambda = n^2/k$, si k est grand et λ petit.

Si on observe c collisions, on calcule les p-valeurs:

$$\begin{aligned} p^+(c) &= P[X \geq c \mid X \sim \text{ Poisson}(\lambda)], \\ p^-(c) &= P[X \leq c \mid X \sim \text{ Poisson}(\lambda)], \end{aligned}$$

On rejette $\mathcal{H}_0$ si $p^+(c)$ est régulièrement très proche de 0 (trop de collisions)
ou $p^-(c)$ est régulièrement très proche de 1 (pas assez de collisions).

Exemple: espacement des anniversaires

On partitionne encore $[0,1)^t$ en $k = d^t$ cubes et on génère n points.
Soient $I_1 \leq I_2 \leq \cdots \leq I_n$ les numéros des boîtes où tombent les points.
On calcule les espacements $S_j = I_{j+1} - I_j$, $1 \leq j \leq n-1$.
Soient $S_{(1)}, \ldots, S_{(n-1)}$ les espacements triés.
Nombre de collisions entre les espacements:

$$Y = \sum_{j=1}^{n-1} I[S_{(j+1)} = S_{(j)}].$$

Si k est grand, sous $\mathcal{H}_0$, Y est approx. Poisson de moyenne $\lambda = n^3/(4k)$.

Si Y prend la valeur y, la p-valeur à droite est

$$p^+(y) = P[X \geq y \mid X \sim \text{ Poisson}(\lambda)].$$

Autres exemples

Paires de points les plus proches $[0,1)^t$.

Trier des jeux de cartes (poker, etc.).

Rang d'une matrice binaire aléatoire.

Complexité linéaire d'une suite binaire.

Mesures d'entropie.

Mesures de complexité basées sur la facilité de compression de la suite.

Etc.

Le Logiciel TestU01

[L'Ecuyer et Simard, ACM Trans. on Math. Software, 2007].

- Implantation d'une grande variété de tests statistiques pour des générateurs quelconques (logiciels ou matériels). Écrit en C. Disponible sur ma page web.

- Contient aussi des batteries de tests prédéfinies:
  SmallCrush: vérification rapide, 15 secondes;
  Crush: 96 tests statistiques, 1 heure;
  BigCrush: 144 tests statistiques, 6 heures;
  Rabbit: pour les suites de bits.

- Plusieurs générateurs couramment utilisés échouent ces batteries.

Quelques résultats. $\rho =$ période du GPA;
t-32 et t-64 donnent le temps de CPU pour générer 10^8 nombres réels.

Nombre de tests échoués (p-valeur $< 10^{-10}$ ou $> 1 - 10^{-10}$) dans chaque batterie.

Résultats de batteries de tests appliqués à des GPA bien connus

Générateur	$\log_2 \rho$	t-32	t-64	SmallCrush	Crush	BigCrush
LCG in Microsoft VisualBasic	24	3.9	0.66	14	—	—
LCG(2^{31}, 65539, 0)	29	3.3	0.65	14	125 (6)	—
LCG(2^{32}, 69069, 1)	32	3.2	0.67	11 (2)	106 (2)	—
LCG(2^{32}, 1099087573, 0)	30	3.2	0.66	13	110 (4)	—
LCG(2^{46}, 5^{13}, 0)	44	4.2	0.75	5	38 (2)	—
LCG(2^{48}, 25214903917, 11), Unix	48	4.1	0.65	4	21 (1)	—
Java.util.Random	47	6.3	0.76	1	9 (3)	21 (1)
LCG(2^{48}, 5^{19}, 0)	46	4.1	0.65	4	21 (2)	—
LCG(2^{48}, 33952834046453, 0)	46	4.1	0.66	5	24 (5)	—
LCG(2^{48}, 44485709377909, 0)	46	4.1	0.65	5	24 (5)	—
LCG(2^{59}, 13^{13}, 0)	57	4.2	0.76	1	10 (1)	17 (5)
LCG(2^{63}, 5^{19}, 1)	63	4.2	0.75		5	8
LCG(2^{31}–1, 16807, 0), Wide use	31	3.8	3.6	3	42 (9)	—
LCG(2^{31}–1, $2^{15} - 2^{10}$, 0)	31	3.8	1.7	8	59 (7)	—
LCG(2^{31}–1, 397204094, 0), (SAS)	31	19.0	4.0	2	38 (4)	—
LCG(2^{31}–1, 742938285, 0)	31	19.0	4.0	2	42 (5)	—
LCG(2^{31}–1, 950706376, 0)	31	20.0	4.0	2	42 (4)	—
LCG(10^{12}–11, ..., 0), in Maple	39.9	87.0	25.0	1	22 (2)	34 (1)
LCG(2^{61}–1, $2^{30} - 2^{19}$, 0)	61	71.0	4.2		1 (4)	3 (1)

Générateur	$\log_2 \rho$	t-32	t-64	SmallCrush	Crush	BigCrush
Wichmann-Hill, in Excel	42.7	10.0	11.2	1	12 (3)	22 (8)
CombLec88	61	7.0	1.2		1	
Knuth(38)	56	7.9	7.4		1 (1)	2
ran2, in Numerical Recipes	61	7.5	2.5			
CLCG4	121	12.0	5.0			
Knuth(39)	62	81.0	43.3		(1)	3 (2)
MRGk5-93	155	6.5	2.0			
DengLin (2^{31}–1, 2, 46338)	62	6.7	15.3	(1)	11 (1)	19 (2)
DengLin (2^{31}–1, 4, 22093)	124	6.7	14.6	(1)	2	4 (2)
DX-47-3	1457	—	1.4			
DX-1597-2-7	49507	—	1.4			
Marsa-LFIB4	287	3.4	0.8			
CombMRG96	185	9.4	2.0			
MRG31k3p	185	7.3	2.0		(1)	
MRG32k3a SSJ + others	191	10.0	2.1			
MRG63k3a	377	—	4.3			
LFib(2^{31}, 55, 24, +)	85	3.8	1.1	2	9	14 (5)
LFib(2^{31}, 55, 24, −)	85	3.9	1.5	2	11	19
ran3, in Numerical Recipes		2.2	0.9	(1)	11 (1)	17 (2)
LFib(2^{48}, 607, 273, +)	638	2.4	1.4		2	2
Unix-random-32	37	4.7	1.6	5 (2)	101 (3)	—
Unix-random-64	45	4.7	1.5	4 (1)	57 (6)	—
Unix-random-128	61	4.7	1.5	2	13	19 (3)
Unix-random-256	93	4.7	1.5	1 (1)	8	11 (1)

Générateur	$\log_2 \rho$	t-32	t-64	SmallCrush	Crush	BigCrush
Knuth-ran_array2	129	5.0	2.6		3	4
Knuth-ranf_array2	129	11.0	4.5			
SWB(2^{24}, 10, 24)	567	9.4	3.4	2	30	46 (2)
SWB(2^{24}, 10, 24)[24, 48]	566	18.0	7.0		6 (1)	16 (1)
SWB(2^{24}, 10, 24)[24, 97]	565	32.0	12.0			
SWB(2^{24}, 10, 24)[24, 389]	567	117.0	43.0			
SWB(2^{32}–5, 22, 43)	1376	3.9	1.5	(1)	8	17
SWB(2^{31}, 8, 48)	1480	4.4	1.5	(2)	8 (2)	11
Mathematica-SWB	1479	—	—	1 (2)	15 (3)	—
SWB(2^{32}, 222, 237)	7578	3.7	0.9		2	5 (2)
GFSR(250, 103)	250	3.6	0.9	1	8	14 (4)
GFSR(521, 32)	521	3.2	0.8		7	8
GFSR(607, 273)	607	4.0	1.0		8	8
Ziff98	9689	3.2	0.8		6	6
T800	800	3.9	1.1	1	25 (4)	—
TT800	800	4.0	1.1		12 (4)	14 (3)
MT19937, widely used	19937	4.3	1.6		2	2
WELL1024a	1024	4.0	1.1		4	4
WELL19937a	19937	4.3	1.3		2 (1)	2
LFSR113	113	4.0	1.0		6	6
LFSR258	258	6.0	1.2		6	6
Marsa-xor32 (13, 17, 5)	32	3.2	0.7	5	59 (10)	—
Marsa-xor64 (13, 7, 17)	64	4.0	0.8	1	8 (1)	7

Générateur	$\log_2 \rho$	t-32	t-64	SmallCrush	Crush	BigCrush
Matlab-rand	1492	27.0	8.4		5	8 (1)
Matlab-LCG-Xor (normal)	64	3.7	0.8		3	5 (1)
SuperDuper-73, in S-Plus	62	3.3	0.8	1 (1)	25 (3)	—
SuperDuper64	128	5.9	1.0			
R-MultiCarry	60	3.9	0.8	2 (1)	40 (4)	—
KISS93	95	3.8	0.9		1	1
KISS99	123	4.0	1.1			
Brent-xor4096s	131072	3.9	1.1			
ICG(2^{31}–1, 22211, 11926380)	31	74.0	69.0		5	10 (8)
EICG(2^{31}–1, 1288490188, 1)	31	55.0	64.0		6	14 (6)
SNWeyl	32	12.0	4.2	1	56 (12)	—
Coveyou-32	30	3.5	0.7	12	89 (5)	—
Coveyou-64	62	—	0.8		1	2
LFib(2^{64}, 17, 5, $*$)	78	—	1.1			
LFib(2^{64}, 55, 24, $*$)	116	—	1.0			
LFib(2^{64}, 607, 273, $*$)	668	—	0.9			
LFib(2^{64}, 1279, 861, $*$)	1340	—	0.9			
ISAAC		3.7	1.3			
AES (OFB)		10.8	5.8			
AES (CTR)	130	10.3	5.4			(1)
AES (KTR)	130	10.2	5.2			
SHA-1 (OFB)		65.9	22.4			
SHA-1 (CTR)	442	30.9	10.0			

Conclusion

- Une foule d'applications informatiques reposent sur les GPAs.
  Un mauvais générateur peut fausser complètement les résultats d'une simulation, ou permettre de tricher dans les loteries ou déjouer les machines de jeux, ou mettre en danger la sécurité d'informations importantes.

- Ne jamais se fier aveuglément aux GPAs fournis dans les logiciels commerciaux ou autres, même les plus en vue, surtout s'ils utilisent des algorithmes secrets!

- Certains logiciels ont d'excellents GPAs; il faut toujours vérifier!