Simulation

Fabian Bastin

Avril 2011

Génération de nombres aléatoires

A la base de toute génération de nombres aléatoires se trouvent la génération de nombres uniformes sur l'intervalle (0,1).

A peu près tout système et outil d'estimation/simulation propose un tel générateur. La plupart ont de très mauvaises propriétés statistiques.

Le choix d'un bon générateur est important pour garantir des résultats crédibles. Il est parfois préférable de recoder un générateur plutôt que d'utiliser les routines disponibles.

Normbre de bons générateurs sont disponibles dans la litérature : MRG32k3a, Mersenne-Twister,. . .
Nous nous concentrerons ici uniquements sur les générateurs de type MRG.

Principes de base

Définissons une fonction de transition $f : \mathcal{S} \to \mathcal{S}$, où $\mathcal{S}$ est l'espace d'états, de cardinalité finie.

Nous notons l'état initial par s_0, et écrivons

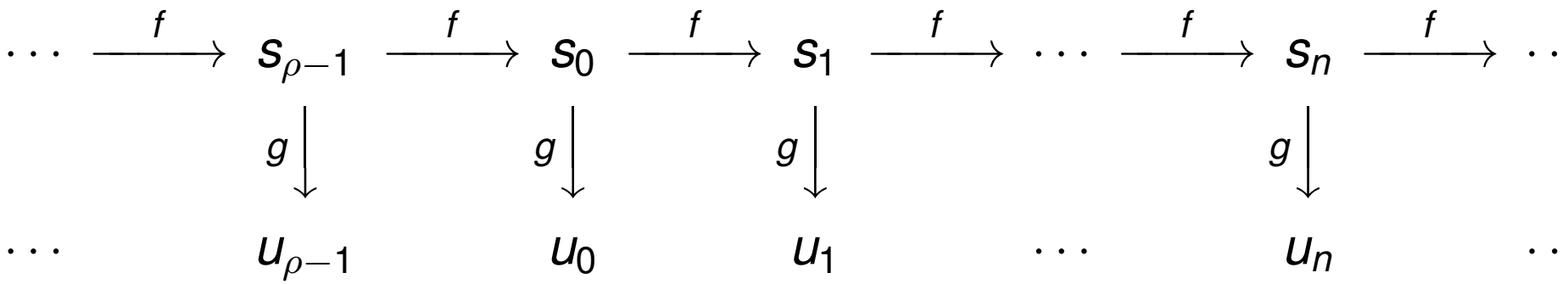
$$s_n = f(s_{n-1}).$$

Nous supposerons de plus que f est périodique pour tout n plus grand ou égal à un certain τ connu (souvent égal à 0), avec la période ρ.
En d'autres termes, nous avons

$$s_{n+\rho} = s_n \; \forall \, n \geq \tau.$$

Fonctions de transition et de sortie

- Espace de sortie : $\mathcal{U}$.
- Nous supposons ici que $\mathcal{U} = (0, 1)$.
- Fonction de sortie $g : \mathcal{S} \to \mathcal{U}$.
  Elle transforme l'état s_n en une valeur de sorite u_n.



Comment choisr f et g ?

Buts : grand ρ, bonne uniformité, comportement "aléatoire".

Générateurs congruentiels linéaires

Les générateurs congruentiels linéaires (GCLs) ont été introduits par by Lehmer em 1951, et se basent sur la formule récursive

$$Z_i = f(Z_{i-1}) = (aZ_{i-1} + c) \mod m.$$

Etant donné deux nombres a (le dividende) et n (le diviseur),

$$a \text{ modulo } n$$

(abrégé comme $a \mod n$) est le reste de la division de a par n. Par exemple, l'expression "7 mod 3" sera évalué à 1, tandis que "9 mod 3" sera évalué à 0. En d'autres termes,

$$a \bmod n = a - n \left\lfloor \frac{a}{n} \right\rfloor.$$

Générateur ”standard minimal”

Un GCL populaire est le ””standard minimal”, selon la terminologie introduite par Park et Miller en 1988 :

$$x_{n+1} = 16807 x_n \bmod 2147483647.$$

Observons que $2147483647 = 2^{31} - 1$; sur une architecture 32-bit, le plus grand entier (signé) représentable est 2^{31}.

De plus, $2^{31} - 1$ est premier, de sorte que le générateur a comme longueur de période $2^{31} - 1$.

Le générateur Standard Minimal

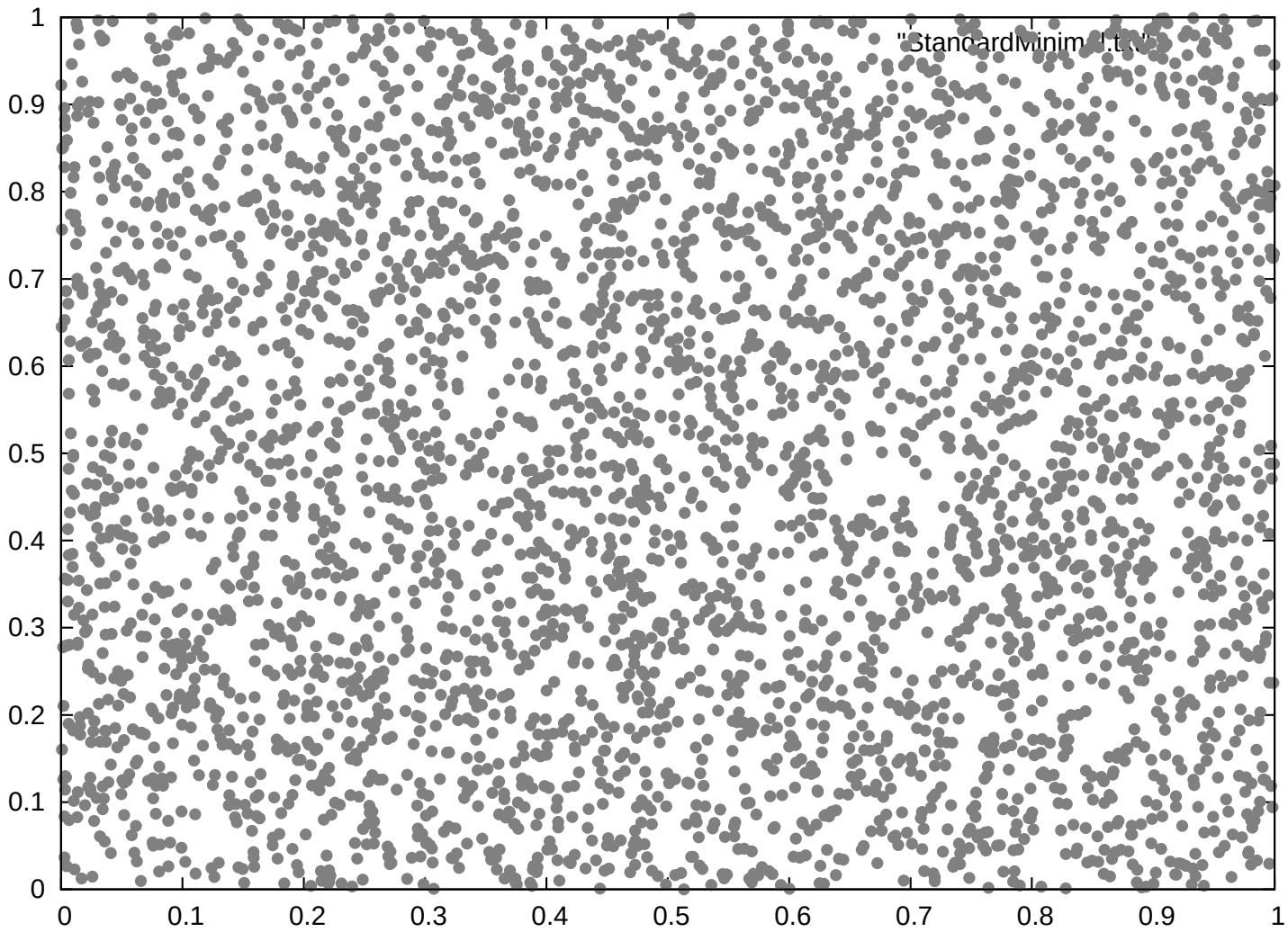
```
#define a       16807
#define m       2147483647
#define AM_MIL (1.0/m)
#define IQ_MIL 127773
#define IR_MIL 2836

double ran_standardminimal_get_val(long *state)
{
  long k;
  double ans;

  k=(*state)/IQ_MIL;
  idum=a*(*state-k*IQ_MIL)-IR_MIL*k;
  if (*state < 0) *state += m;
  ans=AM_MIL*(*state);

  return ans;
}
```

Générateur standard minimal : illustration



4096 points générés sur le carré unité.

Multiple Recursive Generator (MRG)

On peut généraliser la récurrence comme suit :

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

En pratique, nous prendrons $u_n = (x_n + 1)/(m + 1)$, ou $u_n = x_n/(m + 1)$ si $x_n > 0$ et $u_n = m/(m + 1)$ sinon, mais la structure reste la même. Ce genre de générateurs est très populaire.

Etat à l'étape n :

$$s_n = (x_{n-k+1}, \ldots, x_n)^T.$$

Espace d'état : $\mathcal{Z}_m^k$, of cardinality m^k.

La période est maximale si $\rho = m^k - 1$.

On peut montrer que pour $k > 1$, il est suffisant d'avoir au moinx deux coefficients non nuls, incluant a_k, afin d'obtenir la période maximale.

Dès lors, la récurrence la moins coûteuse a la forme

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

Mais comment choisir a_r et a_k ?

Il est possible d'étudier les propriétés théoriques des MRG's, et d'exclure directement certains générateurs dont on sait qu'ils auront de fortes déficiences.

Choix d'un bon MRG

Exemple : Lagged-Fibonacci

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

On peut montrer que les vecteurs $(u_n, u_{n+k-r}, u_{n+k})$ sont tous contenus dans deux plans. Nous savons dès lors que les nombres obtenus ne peuvent être considérés aléatoires, et ce sans tests supplémentaires.

En pratique, nous pouvons imposer diverses conditions sur les coefficients, et calculer des générateurs attrayants théoriquement, en maximisant une certaines mesure de qualité. Ces maximisation est numériquement coûteuse.

MRG's combinés

Considérons deux MRG's (ou plus) travaillant en parallèle :

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

Nous définissons les deux combinaisons

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

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

MRG32k3a

Le MRG combiné suivant a été proposé par L'Ecuyer, et est aujourd'hui un des générateurs les plus populaires.

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

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

MRG correspondant : $k = 3$,
$m = m_1 m_2 = 18446645023178547541$,
$a_1 = 181696684712528925 57$,
$a_2 = 3186860506199273833$,
$a_3 = 873861326439822262 2$.

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

MRG32k3a : implementation

```
#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;
```

MRG32k3a : implementation (continued) - alternatives

```
    /* 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);
}
```

Alternative : générateur dans $\mathcal{F}_2$. Efficaces, mais beaucoup plus complexes à implémenter, et peuvent souffrir d'un mauvais état initial.

Génération de variables non-uniformes

Référence principale : Luc Devroye, *Non-Uniform Random Variate Generation*, `http://cg.scs.carleton.ca/~luc/rnbookindex.html`.

Supposons que nous avons un bon générateur de nombres aléatoires uniformes, mais nous voulons générer des variables suivant d'autres lois de probabilité : normal, Weibull, Poisson, binomial,...

Les propriétés souhaitées sont

- méthode correcte (ou tout au moins une bonne approximation) ;
- aussi simple que possible, mais aussi rapide que possible ;
- faible consommation mémoire ;
- robuste ;
- compatible avec les techniques de réduction de variance (comme quasi-Monte Carlo).

Inversion

Il s'agit de la méthode de choix, dans les cas où elle applicable. La raison est qu'elle est compatible avec les techniques de réduction de variance.

Considérons une variable aléatoire X de fonction de répartition F. Soit $U \sim U(0,1)$ et

$$X = F^{-1}(U) = \min\{x : F(x) \geq U\}.$$

Alors

$$P[X \leq x] = P[F^{-1}(U) \leq x] = P[U \leq F(x)] = F(x),$$

i.e., X a la distribution voulue. En effet,

- dans le cas continu, $F(X)$ $U[0,1]$;
- dans le cas discret, $P[X = x_i] = p(x_i)$, $\forall i$, en supposant $x_1 < x_2 < \ldots < x_n$;
- le principe tient toujours pour les distributions mixtes.

Inversion (2)

Pour certaines lois, F est très difficile à inverser, mais nous pouvons souvent approximer F^{-1}.

Example : loi normale.
Si $Z \sim N(0,1)$, alors $X = \sigma Z + \mu : N(\mu, \sigma^2)$.

Il est dès lors suffisant de pouvoir générer une $N(0,1)$.

Il n'y a pas de formule analytique pour $F(x)$ ou $F^{-1}(x)$, mais des codes efficaces existent pour les approximer.

Chi-square, gamma, beta, etc. : beaucoup plus compliqué car la forme F^{-1} dépend des paramètres de la distribution.

Inversion pour les distributions discrètes

$$p(x_i) = P[X = x_i]; \quad F(x) = \sum_{x_i \leq x} p(x_i).$$

Générer U, chercher $I = \min\{i | F(x_i) \geq U\}$ et retourner x_I.

Initialisation : placer x_i et $F(x_i)$ dans des tableaux, pour $i = 1, \ldots, n$.

1. Recherche linéaire (temps en $\mathcal{O}(n)$) :
   $U \leftarrow U(0,1)$; $\quad i \leftarrow 1$;
   tant que $F(x_i) < U$, faire $i \leftarrow i + 1$; renvoyer x_i.
2. Recherche binaire (temps en $\mathcal{O}(\log(n))$) :
   $U \leftarrow U(0,1)$; $\quad L \leftarrow 0$; $\quad R \leftarrow n$;
   tant que $L < R - 1$
   $m \leftarrow \lfloor (L + R)/2 \rfloor$;
   si $F(x_m) < U$ alors $L \leftarrow m$ sinon $R \leftarrow m$;
   renvoyer x_R.

Autres approches

Composition, convolution, acceptation/rejet.

La technique d'acceptation/rejet est la plus importante après l'inversion.

Nous considérons le cas où X is continu (le cas discret est similaire). Soit $f(x)$ la densité de X, et t une fonction "'chapeau" telle que $f(x) \leq t(x) \; \forall \, x$.

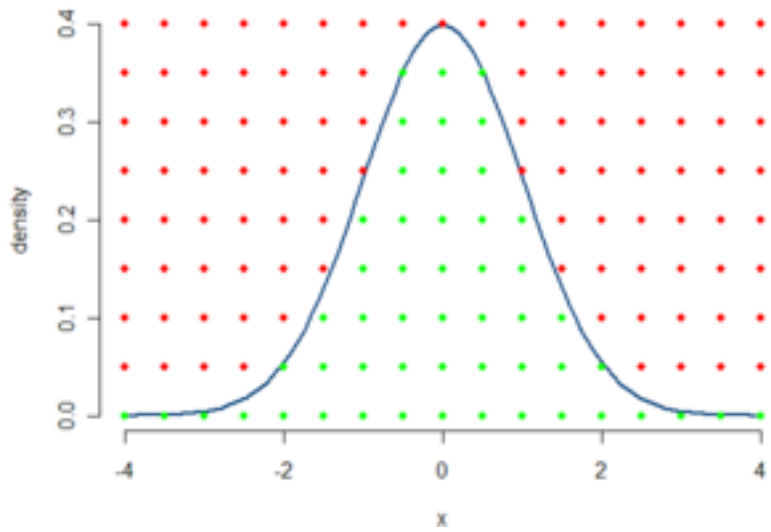
Acceptance/rejection

Normalisation de t en densité r :

$$r(x) = t(x)/a, \text{ où } a = \int_{-\infty}^{\infty} t(s)ds.$$

Algorithme : Répéter

1. générer Y de densité $r(x)$;
2. générer $U : U(0,1)$ indépendamment de Y ;
3. jusqu'à obtenir $U \leq f(Y)/t(Y)$;
4. renvoyer Y.



Cas particulier

Parfois, nous pouvons bénéficier de transformations mathématiques. . . mais rarement compatibles avec les techniques de réduction de variance.

Example : méthode Box-Muller pour la loi normale.
Idée : il est plus facile de générer un point (X, Y) de la normale bivariée, de densitée sur $\mathbb{R}^2$

$$f(x, y) = \frac{1}{2\pi} e^{-(x^2+y^2)/2}.$$

Nous changeons les coordonnées cartésiennes (X, Y) par les coordonnées polaires (R, Θ) :

$$R^2 = X^2 + Y^2; Y = R \sin \Theta.$$

Approche élégante, mais incompatible avec les techniques de réduction de variance, et en pratique plus lente que l'inversion.

Méthodes RQMC

RQMC : randomized quasi-Monte Carlo.

Comme dans le cadre MC, on va estimer

$$\mu = \int_{(0,1)^s} f(\boldsymbol{u})\, d\boldsymbol{u}$$

en moyennisant les évaluations de f sur un ensemble de n points $P_n = \{\boldsymbol{U}_0, \ldots, \boldsymbol{U}_{n-1}\}$:

$$\hat{\mu}_{n,\text{rqmc}} = \frac{1}{n} \sum_{i=0}^{n-1} f(\boldsymbol{U}_i).$$

Un ensemble de points RQMC vise les critères suivant :

(a) P_n *couvre* $(0,1)^s$ *très uniformément en tant qu'ensemble ;*

(b) chaque point individuel $\boldsymbol{U}_i$ *suit une distribution uniforme sur* $(0,1)^s$.

Méthodes RQMC (suite)

La condition (b) assure que la moyenne $\hat{\mu}_{n,\text{rqmc}}$ est un estimateur non biaisé de μ.

Sous (a), nous *espérons* que l'estimateur RQMC aura une plus petite variance que l'estimateur MC classique, si f est suffisamment douce.

Différentes manières de mesurer l'uniformité de P_n sont utilisées pour différents types de constructions et classes d'intégrants, afin d'obtenir des bornes de variance.

Méthodes RQMC (suite)

On spécifie habituellement une classe $\mathcal{H}$ de functions f, et on dérive une borne de pire cas sur l'intégration d'erreur, de la forme

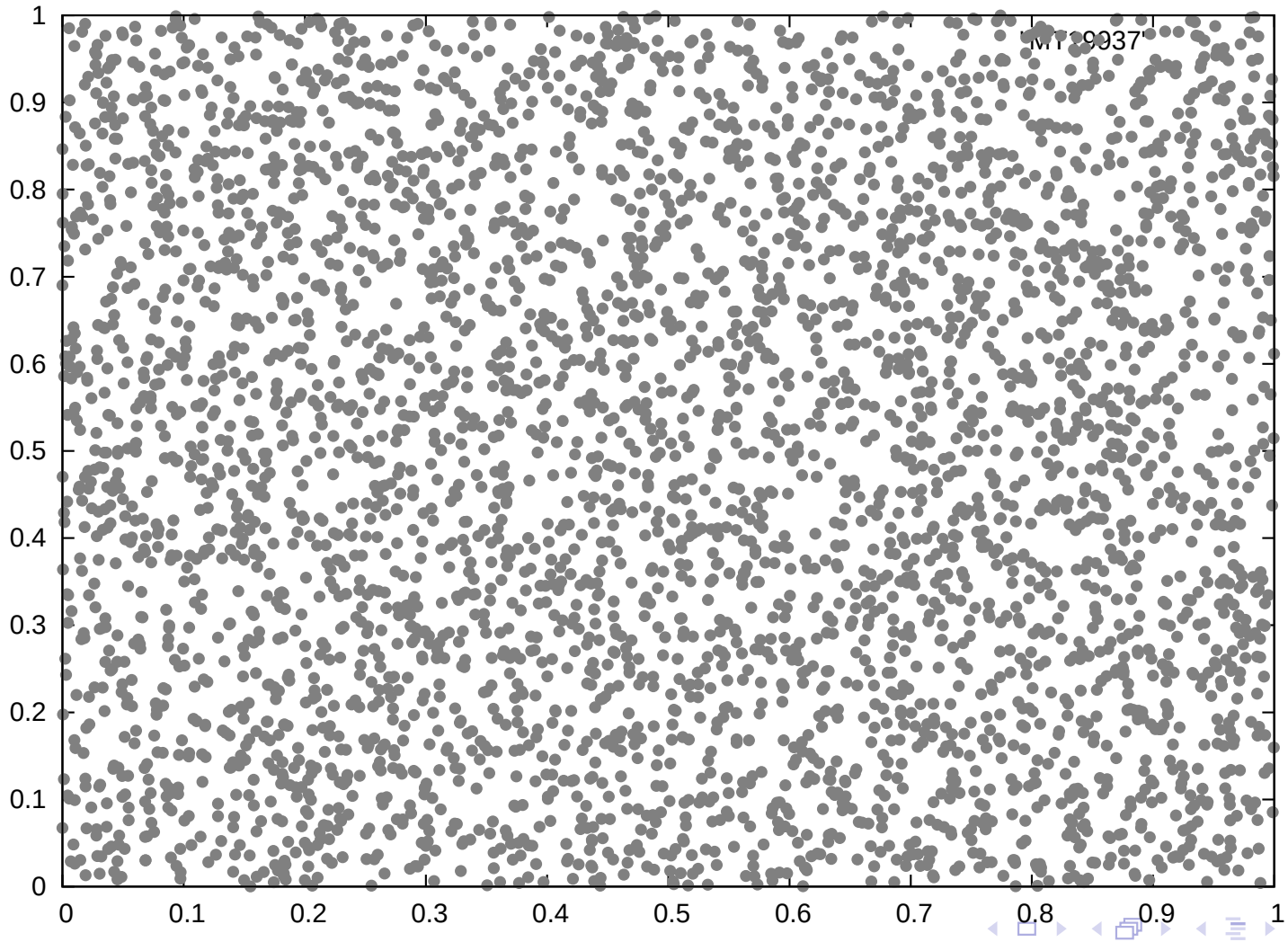
$$|\hat{\mu}_{n,\text{rqmc}} - \mu| \leq D(P_n)\, V(f) \tag{1}$$

pour tout $f \in \mathcal{H}$ et n'importe quel ensemble de point $P_n \subset (0,1)^s$, où $D(P_n)$ mesure la discrépance de P_n par rapport à une distribution uniforme et $V(f) = \|f - \mu\|_{\mathcal{H}}$ mesure the variabilité de f dans $\mathcal{H}$.

Les définitions de $D(P_n)$ and $V(f)$ l'une de l'autre ; et une définition qui rend $V(f)$ plus petit rendra habituellement $D(P_n)$ plus grand, et vice-versa.

A brief look to QMC points : motivation

Avec les nombres pseudo-aléatoires, nous observons des trous et accumulations, comme illustré avec 4096 points MT19937.



Réseaux digitaux (digital nets)

Réseau digital linéaire $\mathcal{Z}_b$, avec $n = b^k$ points : pour $i = 0, \ldots, b^k - 1$,

$$
\begin{aligned}
i &= \sum_{\ell=0}^{k-1} a_{i,\ell} b^\ell, \\
\begin{pmatrix} u_{i,j,1} \\ u_{i,j,2} \\ \vdots \end{pmatrix} &= \boldsymbol{C}^j \begin{pmatrix} a_{i,0} \\ a_{i,1} \\ \vdots \\ a_{i,k-1} \end{pmatrix}, \\
u_{i,j} &= \sum_{\ell=1}^{\infty} u_{i,j,\ell} b^{-\ell}, \\
\boldsymbol{u}_i &= (u_{i,1}, \ldots, u_{i,s}),
\end{aligned}
$$

où la matrice génératrice $\boldsymbol{C}^j$ pour la coordonnée j, $1 \leq j \leq s$, est de dimensions $w \times k$, sur $\mathcal{Z}_b$. En pratique, w est fini.

Séquences digitales

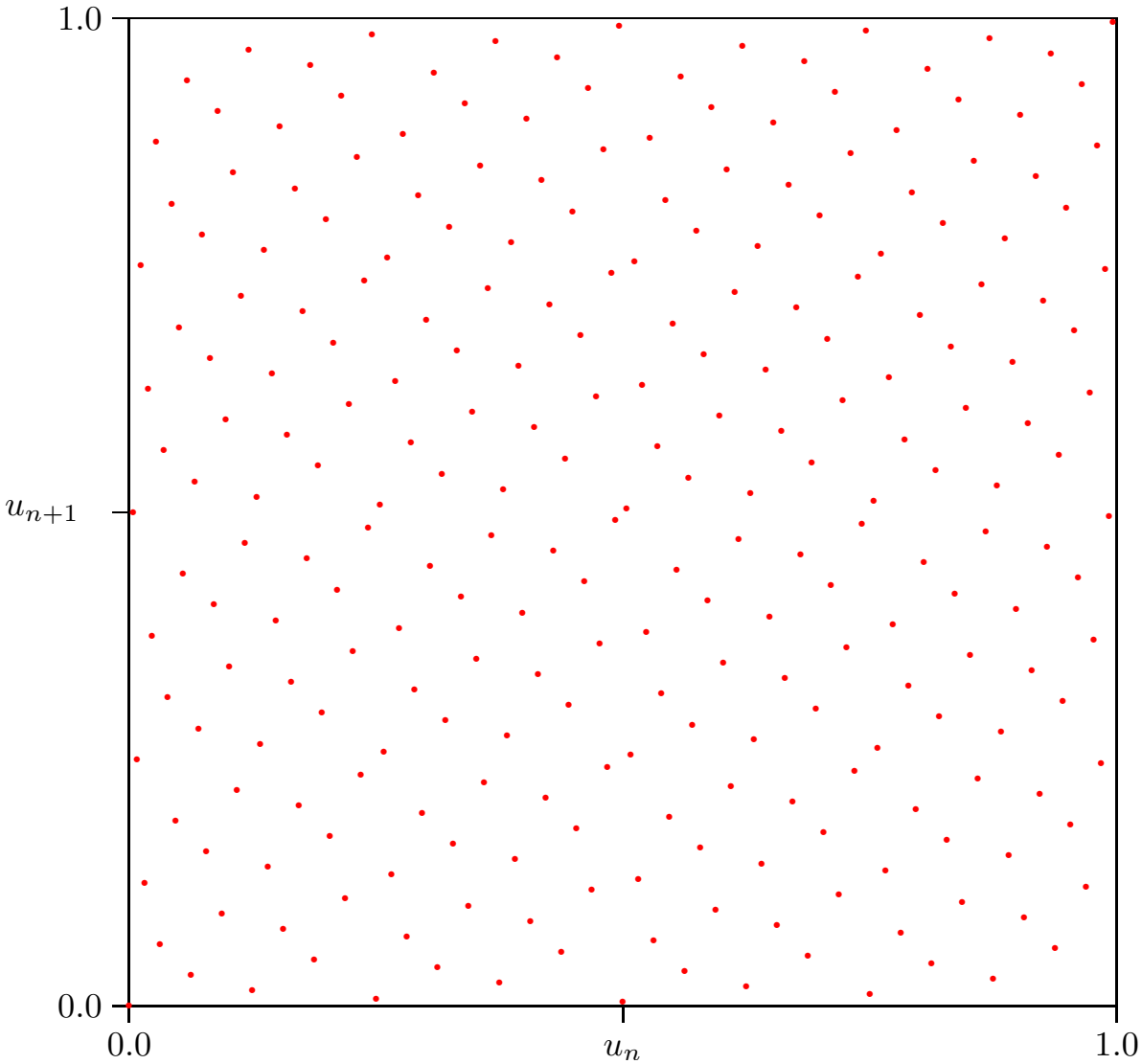
Séquence digitale : séquence infinie de points. Nous pouvons utiliser les $n = b^k$ premiers points pour un certain entier k.
Exemples : Sobol', Faure, Niederreiter,...

Le problème réside dans la construction des matrices $\boldsymbol{C}^j$.

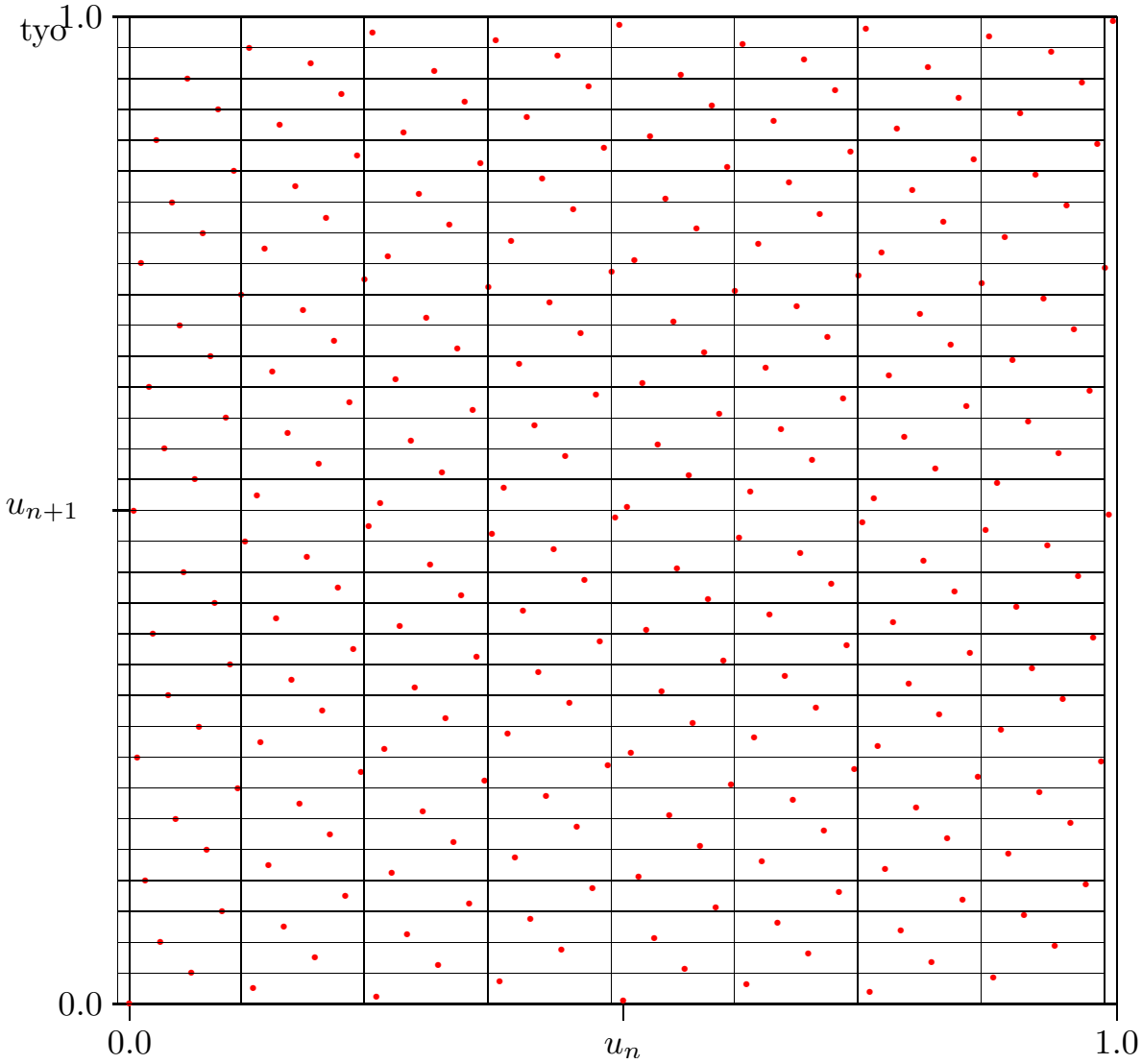
Exemple : let $n = 2^8 = 256$ et $t = 2$.
Ici, $\boldsymbol{C}^1$ est l'identité réfléchie et $\boldsymbol{C}^2$ est l'identité.

En général, nous pouvons prendre $n = 2^k$ points.
Si nous partitionnons $[0, 1)^2$ en blocs rectangulaires de taille 2^{-k_1} by 2^{-k_2}, où $k_1 + k_2 \leq k$, chaque rectangle contiendra le même nombre de points.

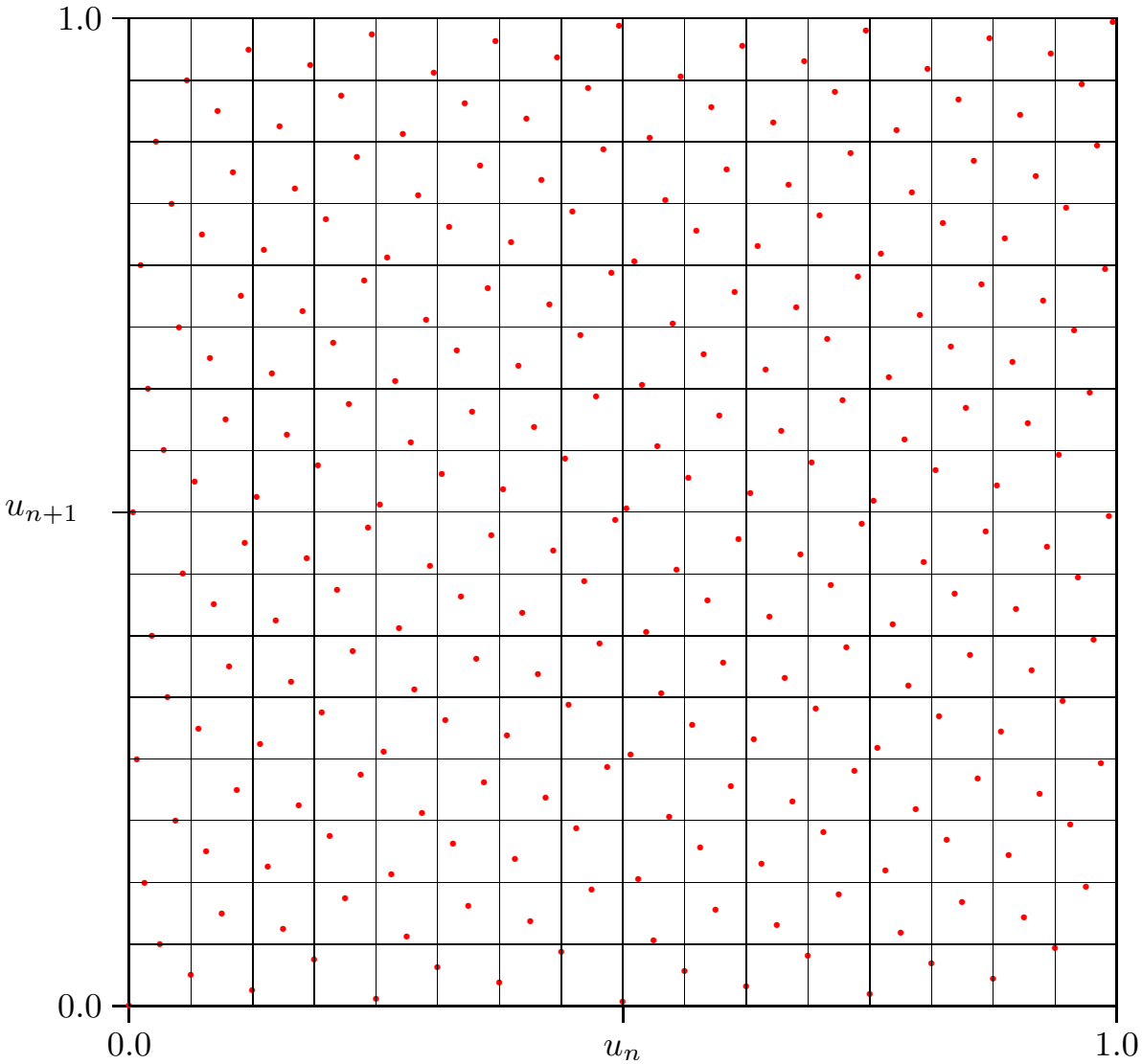
Illustration



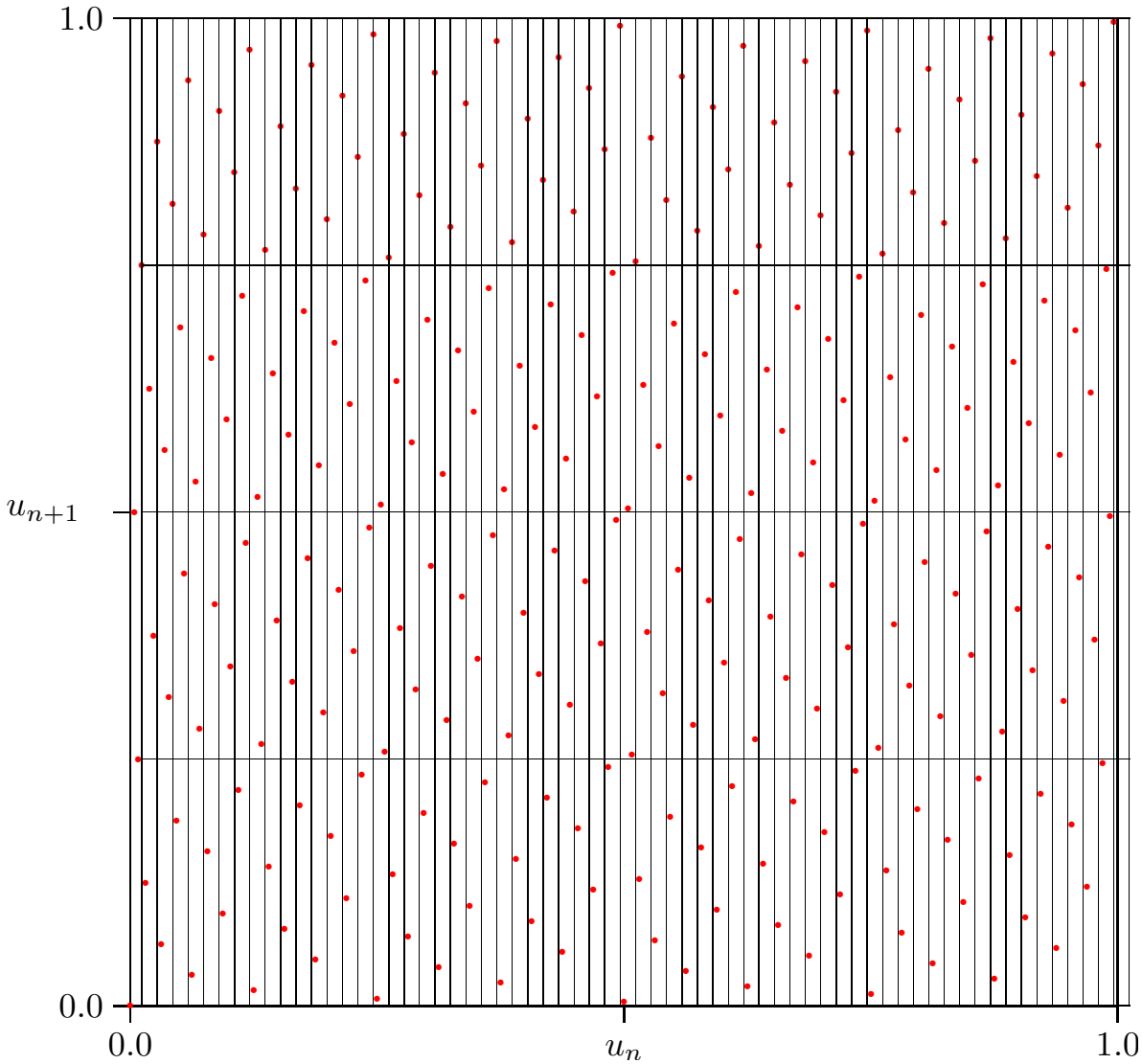
Illustration



Illustration



Illustration



Perturbation aléatoire (”randomisation”)

Comment calculer l'erreur d'intégration ? L'idée basique de RQMC est de partir avec un ensemble $P_n = \{\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{n-1}\}$ de n points qui couvrent l'hypercube unité $[0,1)^s$ d'une maniere très uniforme, et de randomiser P_n de manière à

1. conserver la haute uniformité quand pris en tant qu'ensemble, et
2. d'assurer que chaque point individuel suit la distribution uniforme sur $[0,1)^s$.

Soit $\tilde{P}_n = \{\boldsymbol{U}_0, \ldots, \boldsymbol{U}_{n-1}\}$ l'ensemble de points randomisés. L'estimateur de $\mu = E[f(\boldsymbol{U})]$ basé sur une copie de la randomisation est

$$X_{\text{rqmc}} = Q_n = \frac{1}{n}\sum_{i=0}^{n-1} f(\boldsymbol{U}_i).$$

Cette randomisation est répétée m fois, indépendamment, pour un certain entier positif m, avec le même P_n.

Randomisation (2)

Supposons que tous points pris de différentes randomisations sont indépendants, deux à deux, dans le sens que si $\boldsymbol{U}_i^{(j)}$ est le i^e point de la j^e randomisation, alors pour tous i_1, i_2 et $j_1 \neq j_2$, $\boldsymbol{U}_{i_1}^{(j_1)}$ et $\boldsymbol{U}_{i_2}^{(j_2)}$ sont indépendants.

Nous pouvons alors appliquer la théorie d'inférence statistique, comme dans les méthodes MC traditionnelles.

Règles de lattice décalée aléatoirement

Une règle de lattice de rang 1 avec n points à s dimensions est définie comme suit.

On sélectionne un vecteur $\boldsymbol{a}_1 = (a_1, \ldots, a_s)$ dont les coordonnées appartienne à $Z_n = \{0, \ldots, n-1\}$. Soit $\boldsymbol{v}_1 = (v_1, \ldots, v_s) = \boldsymbol{a}_1/n$, et définissons $P_n^0 = \{\boldsymbol{v} = i\boldsymbol{v}_1 \bmod 1,\ i = 0, 1, \ldots, n-1\}$, où la division et l'opération modulo se font coordonnée par coordonnée.

Les a_j sont généralement pris premiers relativement à n, de sorte que la projection de P_n^0 sur n'importe quelle coordonnées contient les n points distincts $\{0, 1/n, \ldots, (n-1)/n\}$.

Règles de lattice décalée aléatoirement (suite)

Nous randomisons ensuite P_n^0 en appliquant un decalage aléatoire modulo 1, lequel consiste dans la génération d'un seul point $\boldsymbol{U}$ uniformément sur $(0,1)^s$, et en l'ajoutant à point de P_n^0, modulo 1, coordonnée par coordonnée, pour obtenir P_n.

Nous appliquons enfin une transformation du boulanger, qui remplace chaque coordonnée u de chaque point par $2u$ si $u < 1/2$ et par $2 - 2u$ sinon.

Algorithme

Generate $\boldsymbol{U} = (U_1, \ldots, U_s)$ from the uniform distribution over $(0,1)^s$;
for $i = 0$ **to** $n-1$ **do**
  for $j = 1$ **to** s **do**
    $U_{i,j} \leftarrow 2((iv_j + U_j) \bmod 1)$;
    if $U_{i,j} \geq 1$ **then** $U_{i,j} \leftarrow 2 - U_{i,j}$;
  end for
  $\boldsymbol{U}_i = (U_{i,1}, \ldots, U_{i,s})$.
end for