A short tutorial of random numbers generation

Fabian Bastin
Department of Computing Science
and Operational Research
Université de Montréal
Quebec, Canada
http://www.iro.umontreal.ca/~bastin

April 29th, 2008

U[0,1]-distributed random numbers

- A good uniform random generator on the interval $[0, 1]$ is a major component of any good random generator library.
- Draws from other distributions are usually obtained by adequately transform an uniformly distributed sample.

Define a transition function $f : \mathcal{S} \to \mathcal{S}$, where $\mathcal{S}$ is the state space. The cardinality of $\mathcal{S}$ is assumed to be finite.

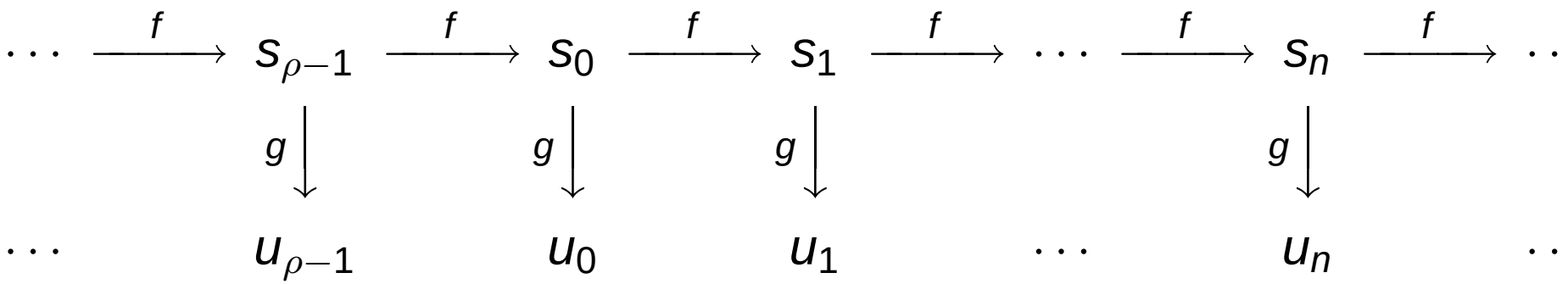
The initial state is denoted by s_0, and we will write

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

We will furthermore assume that f is periodic for all n greater of equal to some known τ (often equal to 0), with the period denoted by ρ. In other terms, we have $s_{n+\rho} = s_n$ for all $n \geq \tau$.

U[0,1]-distributed random numbers (2)

- Output space : $\mathcal{U}$.
- We assume here that $\mathcal{U} = (0, 1)$.
- Output function $g : \mathcal{S} \to \mathcal{U}$.
  It transforms the state s_n into an output value u_n.



How to choose f and g ?

Goals : large ρ, good uniformity, "random" behavior.

Linear congruential generators

Linear congruential generators (LCGs) have been introduced by Lehmer in 1951. We use the recursive formula

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

Given two numbers, a (the dividend) and n (the divisor),

$$a \text{ modulo } n$$

(abbreviated as a mod n) is the remainder, on division of a by n. For instance, the expression ”7 mod 3” would evaluate to 1, while ”9 mod 3” would evaluate to 0. In other terms,

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

Linear congruential generators : full period ?

Full period : m is $c \neq 0$, $m-1$ otherwise (if $c = 0$, 0 is a fixed point for the recurrence).à Considere the case $c \neq 0$.

Theorem (Period)

The LCG has full period if and only if the following three conditions hold :

1. *the only positive integer that (exactly) divides both m and c is 1 ;*
2. *if q is a prime number that divides m, then q divides $a-1$;*
3. *if 4 divides m, then 4 divides a-1.*

A popular LCG is the "standard minimal", as known from the terminology introduced by Park and Miller in 1988 :

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

Observe that $2147483647 = 2^{31} - 1$; on 32-bit architectures, the largest representable (signed) integer is 2^{31}.

The Standard Minimal Generator

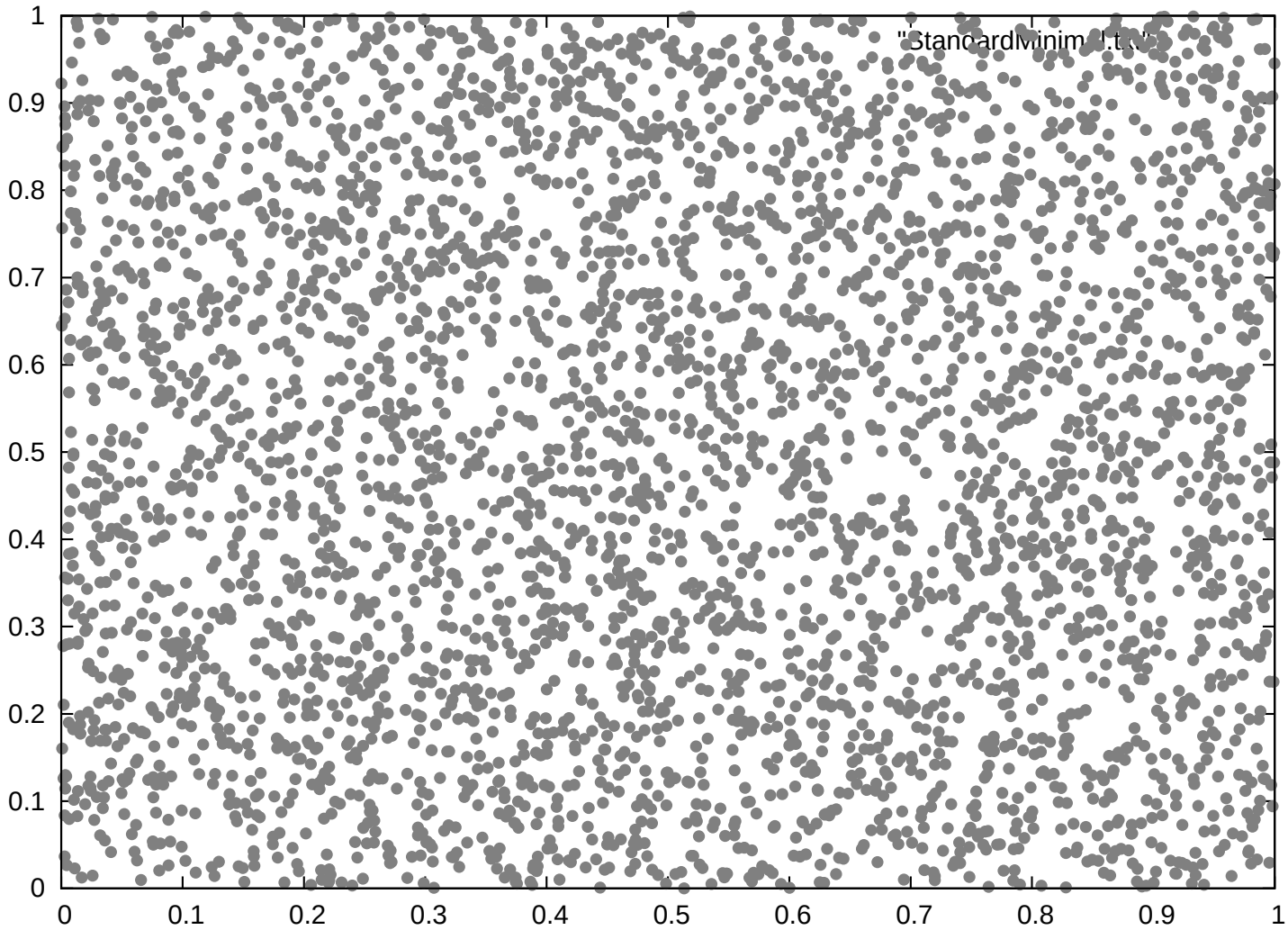
```
#define a      16807
#define m      2147483647
#define AM_MIL (1.0/m)
#define IQ_MIL 127773
#define IR_MIL 2836

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;
}
```

Standard minimal generator : illustration



4096 generated points on the unit square.

Multiple Recursive Generator (MRG)

But we want better ! Generalize the linear congruential generator :

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

In practice, we will take $u_n = (x_n + 1)/(m + 1)$, or $u_n = x_n/(m + 1)$ if $x_n > 0$ and $u_n = m/(m + 1)$ otherwise, but the structure remains the same, and is easier when studying theoretical properties. This kind of generators is very popular.

State at step n :

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

State space : $\mathcal{Z}_m^k$, of cardinality m^k.

The maximal period if $\rho = m^k - 1$.

Period of MRG's

It can be shown that for $k > 1$, it is sufficient to have at least two non-zero coefficient, including a_k, in order to get the maximal period.

The cheapest recurrence has therefore the form

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

But how to choose a_r and a_k ?

It is possible to study theoretical properties of MRG's, and excluding directly some generators that have known strong deficiencies.

Choosing a good MRG's

Example : Lagged-Fibonacci

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

It can be shown the vectors $(u_n, u_{n+k-r}, u_{n+k})$ are all contained in two plans ! We therefore know without additional tests that the numbers cannot be considered as random.

In practice, we can impose various conditions on the coefficients, and compute theoretically appealing generators by maximizing some quality measure. This maximization is numerically expensive.

Combined MRG's

Consider two (or more) MRG's working in parallel :

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

We define the two combinations

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

The sequence $\{w_n, n \geq 0\}$ is the output of another MRG, of module $m = m_1 m_2$, and $\{u_n, n \geq 0\}$ is nearly the same sequence if m_1 and m_2 are close.
We can achieve the period $(m_1^k - 1)(m_2^k - 1)/2$.

MRG32k3a

The following combined MRG was proposed by L'Ecuyer, and is amongst the most popular and efficient known generators. It combines 2 MRG's.

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

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

Corresponding MRG : $k = 3$,
$m = m_1 m_2 = 18446645023178547541$,
$a_1 = 1816966847125289255 7$,
$a_2 = 3186860506199273833$,
$a_3 = 8738613264398222622$.

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);
}
```

The generator ”L’Ecuyer” present in AMLET is a combination of two LCG’s. It is a bit less efficient than MRG32K3a (which will replace it in a future version), but still very good.

Random numbers generators on $\mathcal{F}_2$

Alternatives to MRG's : random numbers generators based on linear recurrence in $\mathcal{F}_2$.

Galois field $\mathcal{F}_2$: set $\{0, 1\}$ on which we define addition and multiplication operation modulo 2.

We construct two sequence of bits vectors $\boldsymbol{x}_n$ and $\boldsymbol{y}_n$ with the linear recurrences

$$\boldsymbol{x}_n = A\boldsymbol{x}_{n-1} \quad \text{(state vector, } k \text{ bits)},$$

and

$$\begin{aligned} \boldsymbol{y}_n &= B\boldsymbol{x}_n \quad \text{(output vector, } w \text{ bits)}, \\ u_n &= \sum_{j=1}^{w} y_{n,j-1} 2^{-j} = .y_{n,0}\, y_{n,1}\, y_{n,2} \cdots \quad \text{(sortie)}. \end{aligned}$$

Random numbers generators on $\mathcal{F}_2$ (2)

The implementation of $\mathcal{F}_2$-generators is often quite complex, but it is possible to operate bitwise, so that they are numerically very fast.

The LFSR (linear feedback shift register), while known to have important deficiencies, gives an illustration of such generators.

We use the relations (with $a_k \neq 0$)

$$u_n = \sum_{l=1}^{w} x_{n\nu+j-1} 2^{-l} = .x_{n\nu} x_{n\nu+1} x_{n\nu+2} \ldots x_{n\nu+l-1}$$

$$X = \begin{pmatrix} & 1 & & \\ & & \ddots & \\ & & & 1 \\ a_k & a_{k-1} & \ldots & a_1 \end{pmatrix}^{\nu} \text{ et } B = I.$$

Tausworthe Generator (2)

Maximum period. $\rho = 2^k - 1$ iff
$Q(z) = z^k - a_1 z^{k-1} - \cdots - a_{k-1} z - a_k$ is primitive and
$\text{pgcd}(\nu, 2^k - 1) = 1$.

In most applications, only two coefficients are nonzero in order to simplify implementation, so we obtain
$Q(z) = z^k - a_r z^{k-r} - a_k$.

Since we are working in $\mathcal{F}_2$, the recurrence on x_n becomes

$$x_n = (x_{n-r} + x_{n-k}) \mod 2.$$

The execution of the addition modulo 2 is equivalent to the instruction exclusive-or (xor) ont the bits :

$$x_n = \begin{cases} 0 & \text{si } x_{n-r} = x_{n-k}, \\ 1 & \text{si } x_{n-r} \neq x_{n-k}. \end{cases}$$

Implementations

More generally, we construct a fast implementation by using shifts, xor's, masks,. . . . We can also combine them.

The Mersenne twister generator (available in AMLET is a pseudorandom number generator developed in 1997 by Makoto Matsumoto and Takuji Nishimura, and constructed on $\mathcal{F}_2$. While it is reputated to be robust and fast, it is not exempt of defaults, since if fails some tests define in TestU01 (L'Ecuyer and Simard, 2007). It is nevertheless a very good generator for most of the applications.

The state length is a vector of 624 integers.

This generator, also known as MT19937, has a period of $2^{19937} - 1$. The code is freely available.

Non-uniform random variables generation

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

Assume that we have a good uniform random variates generator, but we want to generate random variables following various probability laws : Normal, Weibull, Poisson, binomial,...

The desired properties are :

- correct method (or good approximation) ;
- as simple as possible, but as fast as possible ;
- low memory consumption ;
- robust ;
- compatible with variance reduction technique (as quasi-Monte Carlo).

Inversion

This is the prefered method, if it can be applied. The reason is that it is compatible with variance reduction.

Consider a random variable X with cumulative distribution function F. Let $U \sim U(0,1)$ and

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

Then

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

i.e., X has the desired distribution. Indeed,

- in the continuous case, $F(X)$ $U[0,1]$;
- in the discrete case, it is easy to prove that $P[X = x_i] = p(x_i)$, for all i, and we assume $x_1 < x_2 < \ldots < x_n$;
- The principle still works for mixte distributions.

Inversion (2)

- Advantage : monotone, only one U for all X.
- Weakness : for some laws, F is very difficult to invert. But we can often approximate F^{-1}.

Example : normal law.
If $Z \sim N(0,1)$, then $X = \sigma Z + \mu : N(\mu, \sigma^2)$.
It is therefore sufficient to be able to generate a $N(0,1)$, of density $f(x) = (2\pi)^{-1/2} e^{-x^2/2}$.
We do not have any formula for $F(x)$ or $F^{-1}(x)$. Efficient codes however exist to approximate $F^{-1}(x)$.

This method is used in AMLET.

Chi-square, gamma, beta, etc. : it is much more complicated since the form of F^{-1} depends of the distribution parameters.

Inversion for discrete distributions

Recall that

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

We have to generate U, search $I = \min\{i | F(x_i) \ge U\}$ and return x_I.
Various algorithms perform this search. Their efficiency depends of the distribution.
Initialization : store the x_i and $F(x_i)$ in arrays, for $i = 1, \ldots, n$.

1. Linear search (time in $O(n)$) : $U \leftarrow U(0,1)$; $i \leftarrow 1$;
   while $F(x_i) < U$ do $i \leftarrow i + 1$; return x_i.
2. Binary search (time in $O(\log(n))$) :
   $U \leftarrow U(0,1)$; $L \leftarrow 0$; $R \leftarrow n$;
   while $L < R - 1$
   $m \leftarrow \lfloor (L + R)/2 \rfloor$;
   if $F(x_m) < U$ then $L \leftarrow m$ otherwise $R \leftarrow m$;
   return x_R.

Application to mixed logit modeling

Recall that in mixed-logit, we are interested in solving

$$\max_{\theta} LL(\theta) = \max_{\theta} \frac{1}{N} \sum_{n=1}^{N} \ln E\left[P_{ij_i}(x, \theta, \xi)\right],$$

where θ represent the parameters of the random vector over the all distribution (the constants can be seen as Dirak distributions, that is with all the probability mass concentrated on one point).

We usually assume that the marginal distributions are independant, so that we can generate each of them by transforming an $U(0,1)$ and making proper use of the current θ_k (at iteration k).

Multivariate distributions are usually more difficult to generate, except in specific cases like multivariate normals. The question is how to correctly reproduce the correlations.

Other approaches : composition

Assume that F is a convex combination of several cumulative distribution functions :

$$F(x) = \sum_{j=0}^{\infty} p_j F_j(x),$$

and that it is easier to invert $F_j, j = 0, \ldots, \infty$ than F.

Generate $J = j$ with the probability p_j, than generate X following F_J.

The method therefore requires two uniforms for each random variable, and exploit the decomposition

$$P[X \leq x] = \sum_{j=1}^{\infty} P[X \leq x | J = j] P[J = j] = \sum_{j=1}^{\infty} F_j(x) p_j.$$

Convolution

Convolution. Assume that

$$X = Y_1 + Y_2 + \ldots + Y_n,$$

where the Y_i are independent, of given laws. We generate the Y_i, $i = 1, \ldots, n$, and we sum.

Exemples : Erlang (sum of exponentials with same mean), binomial.

Acceptance/rejection : the most important technique after inversion.

We consider the case where X is continuous (the discrete cas is analoge). Let $f(x)$ be the density of X, and let t be a "hat" function that majors f, i.e. $f(x) \leq t(x)\ \forall\, x$.

Acceptance/rejection

We can normalize t in a density r :

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

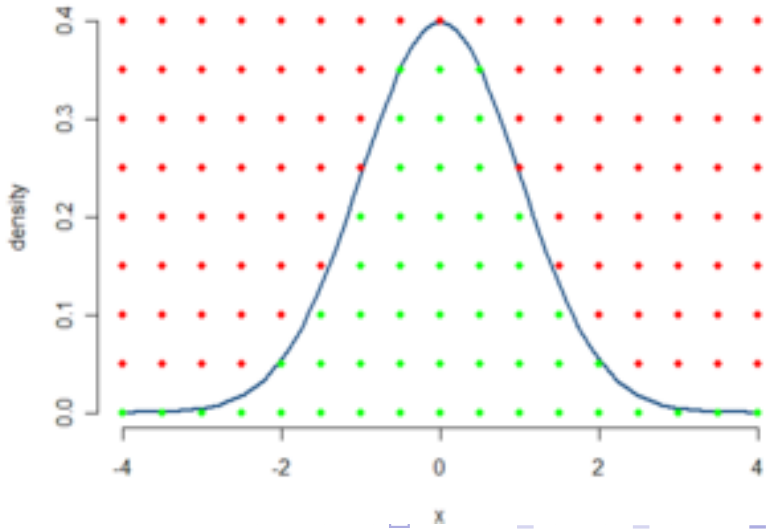
We choose t so that

1. it is easy to generate random variables of density r ;
2. a is small (close 1), or in other terms, $t(x)$ is close to $f(x)$.

The choice of t may be automatized.

Algorithm : Repeat

1. generate Y of density $r(x)$;
2. generate $U : U(0,1)$ independantly of Y ;
3. until $U \leq f(Y)/t(Y)$;
4. return Y.



Particular cases

Sometimes, we can benefit of mathematical transformations. The main weakness is that they are seldom compatible with variance reduction techniques.

Example : Box-Muller method for the normal law.

Idea : it is easier to generate a point (X, Y) from the bivariate normal law, of density on $\mathbb{R}^2$

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

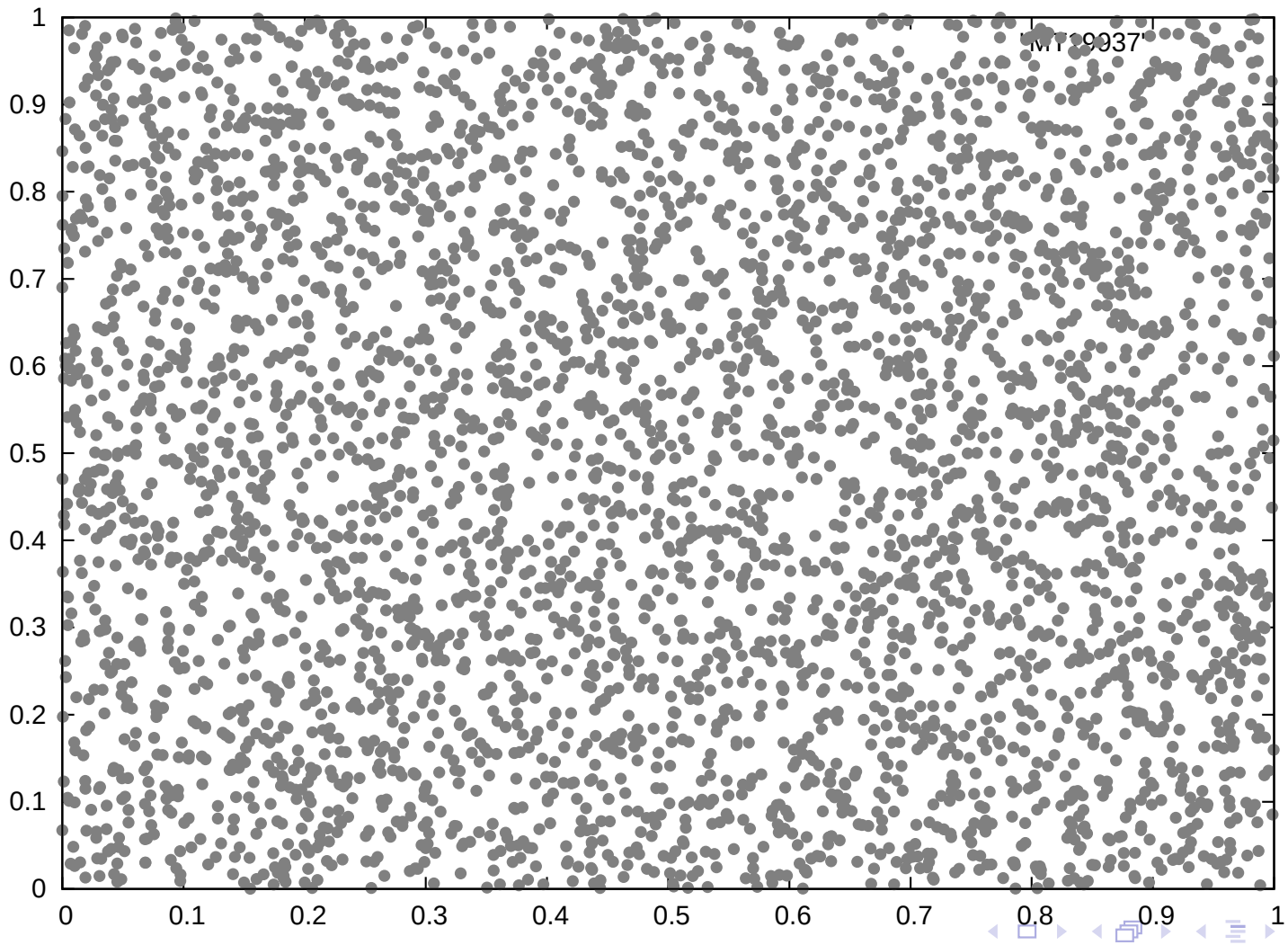
We change the cartesian coordinates (X, Y) by the polar coordinates (R, Θ) :

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

It gives an elegant approach, but incompatible with variance reduction techniques.

A brief look to QMC points : motivation

With pseudo-random numbers, we observed holes and accumaltions, as illustrated with 4096 MT19937 points.



Basic principles

Idea : replace the random points $\boldsymbol{U}_1, \ldots, \boldsymbol{U}_n$ by a set of n well-chosen points, more uniformly distributed over $[0,1)^t$ than typical random points.

The name quasi-random may be misleading, because no attempt is made to imitate randomness, but this name has become a de facto standard in the literature.

To implement this idea, one needs to define a measure of uniformity of a finite point set, or a measure of the discrepancy between the distribution of the point set and the uniform distribution, and then find specific constructions that do well with respect to this measure.

There are several ways of defining such measures. The ones motivated by their mathematical convenience for proving convergence results are very hard to compute.

QMC error

For $1 \leq p < \infty$, the quantity

$$D_n^{*(p)}(P_n) = \left(\int_{[0,1)^t} \left| \text{vol}[\mathbf{0}, \boldsymbol{u}) - \frac{|P_n \cap [\mathbf{0}, \boldsymbol{u})|}{n} \right|^p d\boldsymbol{u} \right)^{1/p}$$

is called the t-dimensional (rectangular) $\mathcal{L}_p$-star discrepancy of P_n.

These quantities measure the worst-case absolute difference, over the selected class of intervals, between the volume of the interval and the fraction of the points that it contains.

The error depends of the dimensionality, and it is always possible to construct an example on which QMC performance is worse than MC. But in general, QMC outperforms MC.

Digital nets

A linear digital net over $\mathcal{Z}_b$, with $n = b^k$ points : for $i = 0, \ldots, b^k - 1$,

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

where the generator matrix $\boldsymbol{C}^j$ for coordinate j, $1 \leq j \leq s$, is $w \times k$, over $\mathcal{Z}_b$. In practice, w is finite.

Digital sequences

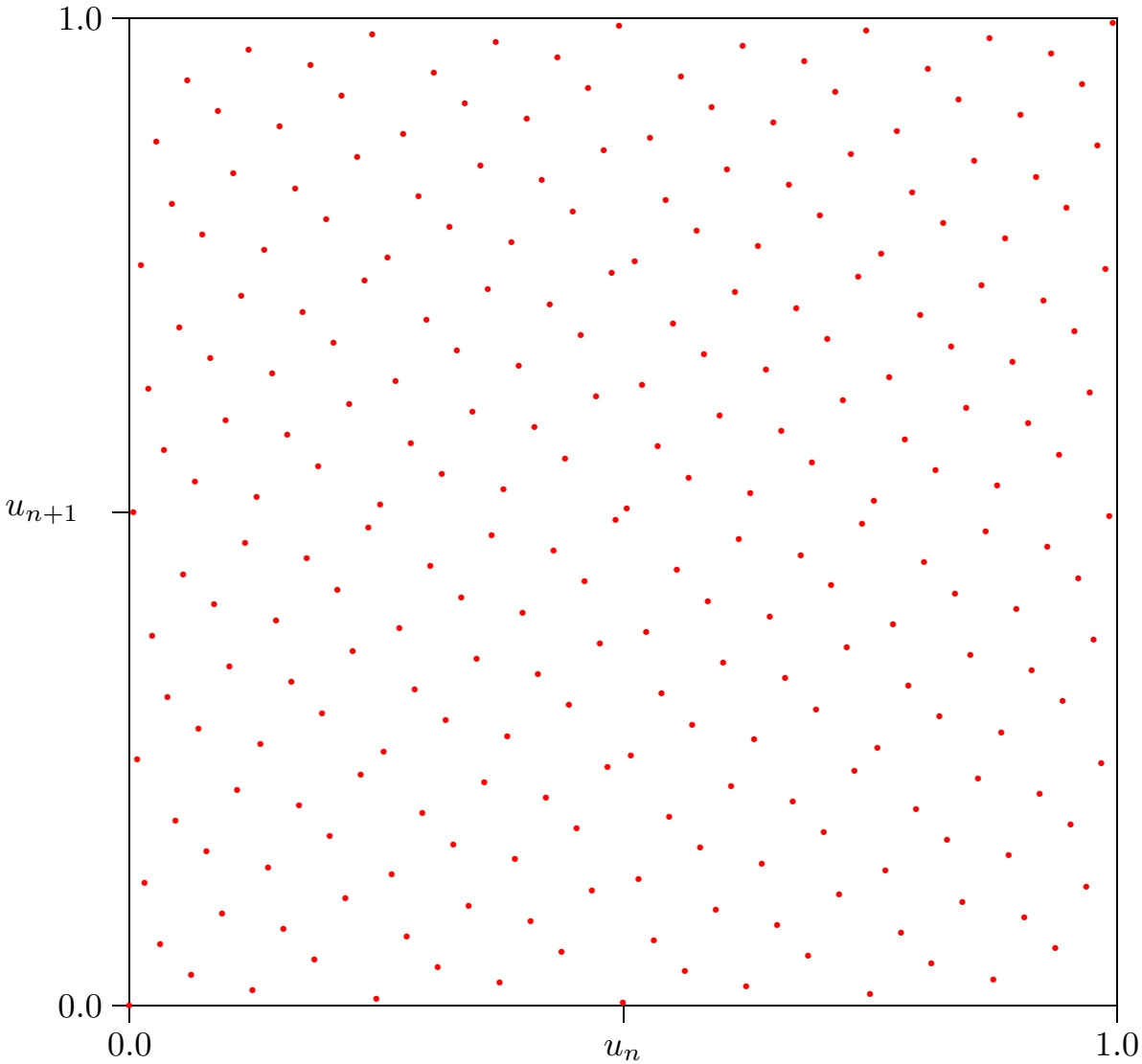
Digital sequence : infinite sequence of points. We can use the first $n = b^k$ points for some integer k.
Examples : Sobol', Faure, Niederreiter,...

The problem is how to construct the matrices $\boldsymbol{C}^j$.

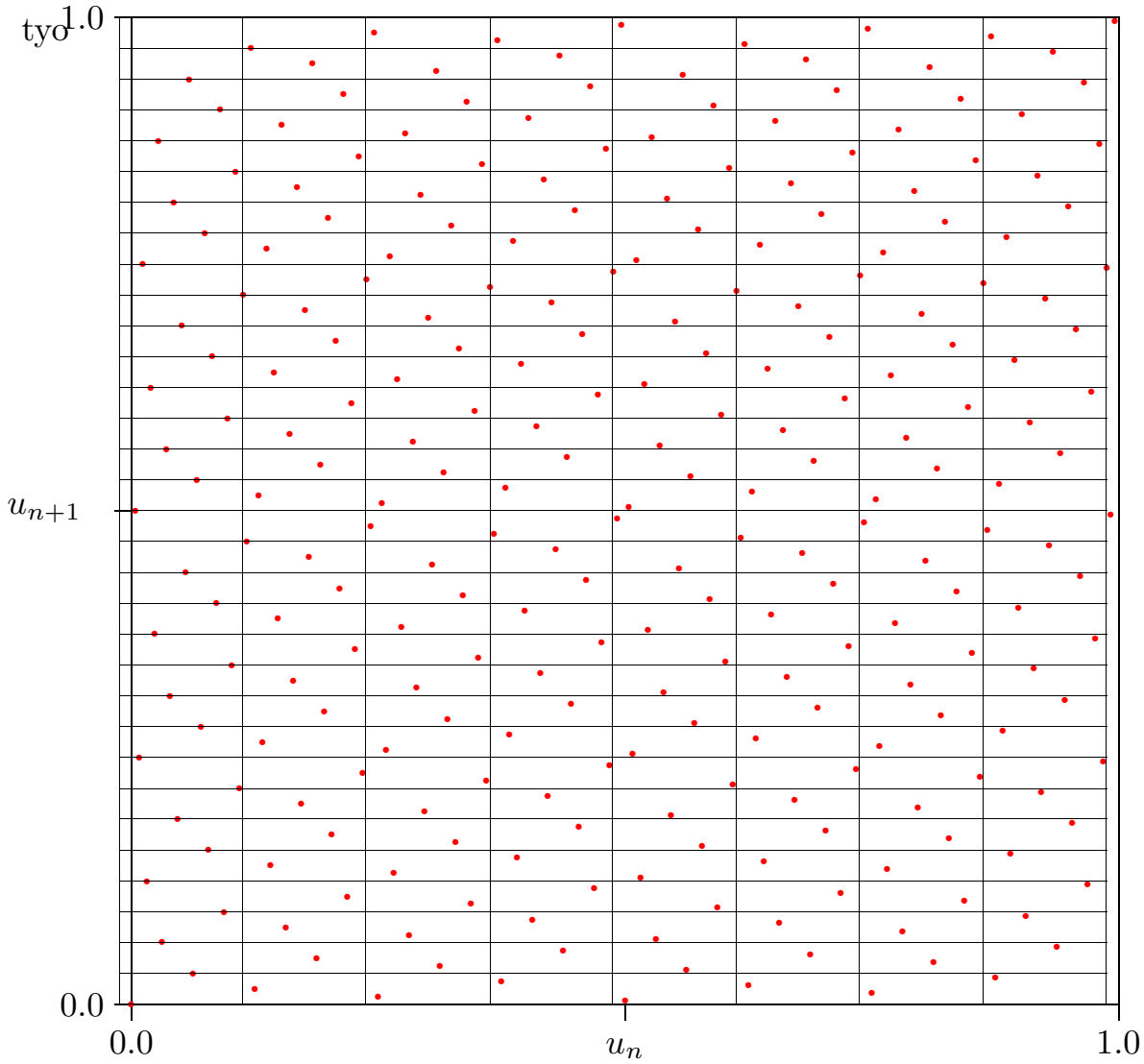
Example : let $n = 2^8 = 256$ et $t = 2$.
Here, $\boldsymbol{C}^1$ is the reflected identity and $\boldsymbol{C}^2$ is the identity.

In general, we can take $n = 2^k$ points.
If we partition $[0,1)^2$ in rectangulars of size 2^{-k_1} by 2^{-k_2}, where $k_1 + k_2 \leq k$, each rectangular will contain the same number of points.

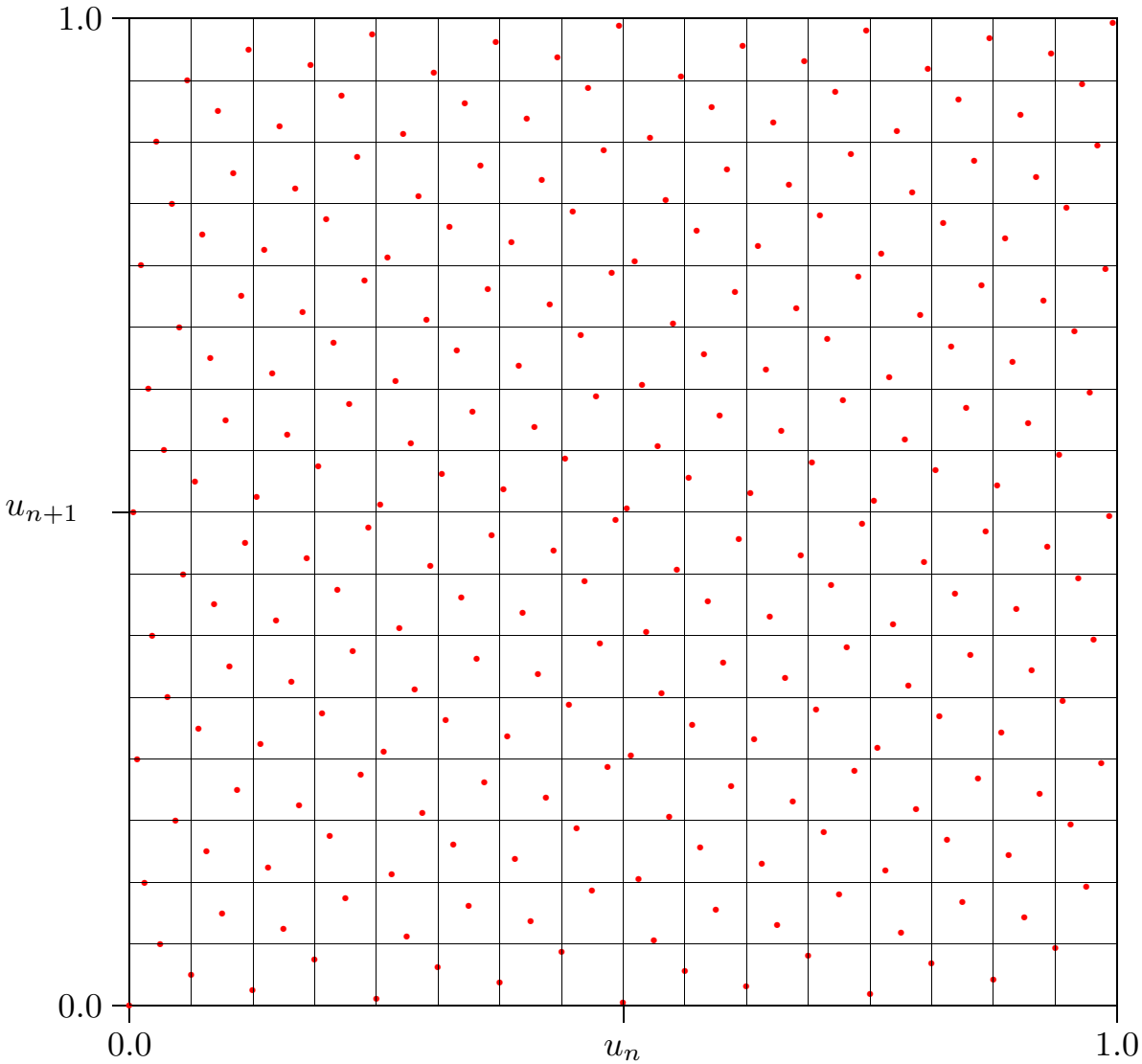
Illustration



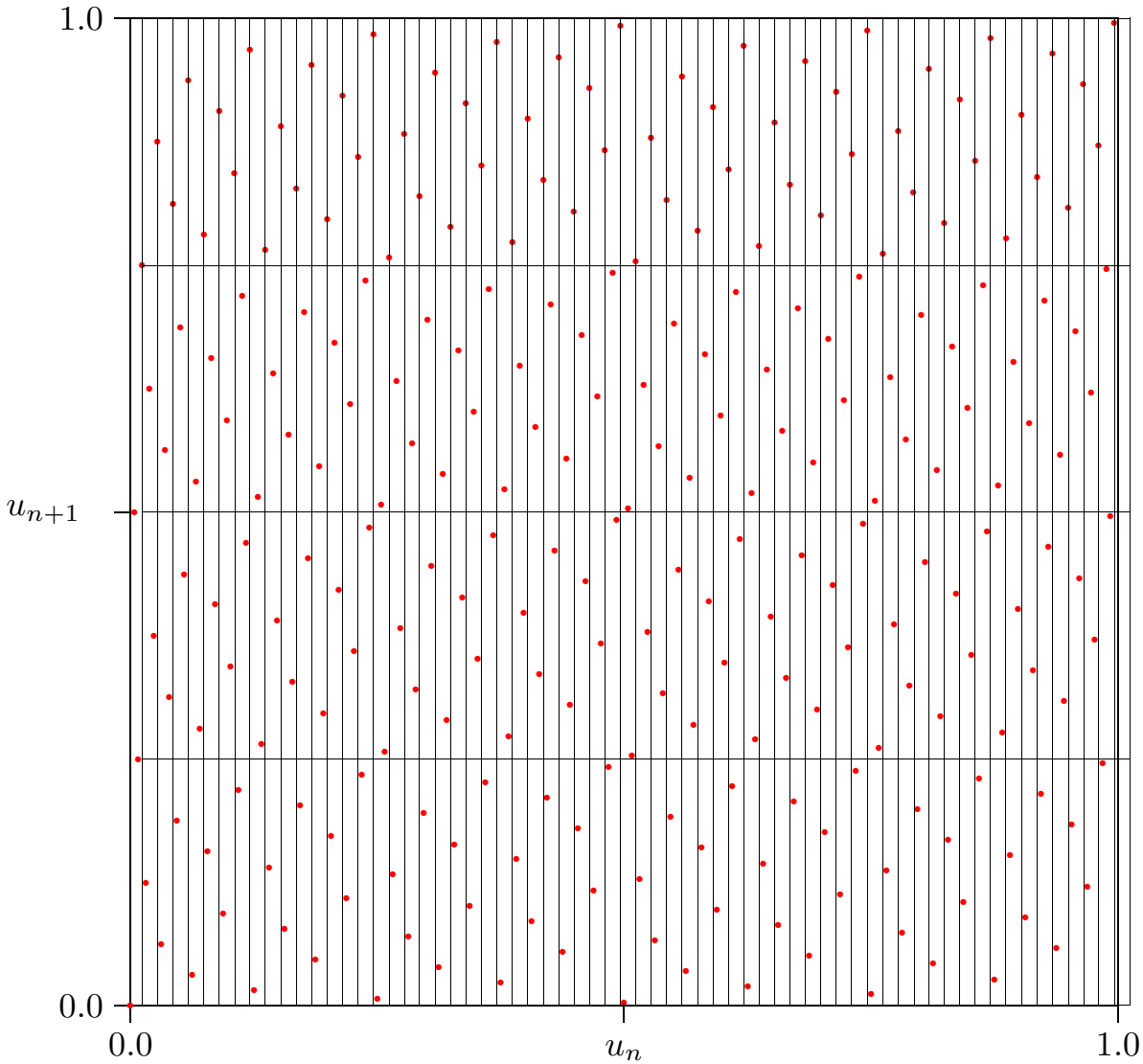
Illustration



Illustration



Illustration



Randomization

How to compute the integration error? The basic idea of RQMC is to start with a set $P_n = \{\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{n-1}\}$ of n points that cover the unit hypercube $[0,1)^t$ in a very uniform way, and randomize P_n so that after the

1. it retains its high uniformity when taken as a set and
2. each individual point has the uniform distribution over $[0,1)^t$.

Let $\tilde{P}_n = \{\boldsymbol{U}_0, \ldots, \boldsymbol{U}_{n-1}\}$ denote the randomized points. The estimator of $\mu = E[f(\boldsymbol{U})]$ based on one copy of the randomization is

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

This randomization is repeated m times, independently, for some positive integer m, with the same P_n.

Randomization (2)

This randomization is repeated m times, independently, for some positive integer m, with the same P_n. Assume that any points taken from different randomizations are pairwise independent, in the sense that if $\boldsymbol{U}_i^{(j)}$ is the ith point from the jth randomization, then for every i_1, i_2 and $j_1 \neq j_2$, $\boldsymbol{U}_{i_1}^{(j_1)}$ and $\boldsymbol{U}_{i_2}^{(j_2)}$ are independent.

We can then apply statistical inference, as in traditional MC methods.

You are welcome to the Eighth International Conference on Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing
Montreal, QC, July 6-11, 2008

