Uniform and non-uniform pseudo random numbers generators for high dimensional applications

Régis LEBRUN

EADS Innovation Works

January 28th 2010

Outline

1. Pseudo-random numbers generator (PRNG)
2. Uniform PRNGs
3. Non-uniform PRNGs

Pseudo-random numbers generator (PRNG)

Any one who considers arithmetical methods of producing random digits is, of coures, in a state of sin.

– John Von NEUMANN (1951)

cited in The Art of Computer Programming, Donald E. KNUTH

Stream of independent, $\mathcal{U}(0,1)$ random variables

Uniformity and independence

- The basic building block of stochastic simulation software is the uniform random numbers generator.
- We are looking for a routine capable of producing independent, identically distributed realizations of the uniform distribution over $[0,1]$.
- We want to do it algorithmically and deterministically !

What is a pseudo-number generator ?

Definition, from [1]

A (pseudo)random number generator is a structure $\mathcal{G} = (S, s_0, T, U, G)$ where :

- S is a finite set of states,
- $s_0 \in S$ is the initial state (or seed),
- the mapping $T : S \to S$ is the transition function,
- U is a finite set of output symbols,
- and $G : S \to U$ is the output function.

The state of the generator evolves according to the recurrence $s_n = T(s_{n-1})$ and the generator outputs the number $u_n = G(s_n)$, called the random numbers.

What is a pseudo-number generator ?

Some comments

- Nothing about randomness in this definition !
- No condition on the repartition of $(u_n)_{n\in\mathbb{N}}$!
- Will be periodic, with a period $P \leq \operatorname{card} T$.
- Such a device is suppose to be able to produce a stream of numbers that "looks like" a sequence of realizations of iid uniform random variables, i.e. that is not statistically different from a true random sequence of random numbers.
- The temptative to define what is a random sequence fills more than 30 pages in [2]...

What is a good pseudo-number generator ?

Uniformity and independence

- Uniformity over $[0, 1]$ cannot be achieved on a computer due to the finite precision of numbers representation, but uniform distribution over $\Omega = \{0/m, 1/m, \ldots, m/m\}$ can be (quite well) approximated through the equirepartition property over Ω.
- Independence between realizations cannot be achieved due to the deterministic nature of the recurrence in the generator, but can be (quite well) approximated due to the mixing property of T. This last point can be made more consistent using the copula theory.

We will illustrate the potential pitfalls in designing a PRNG through the example of Linear Congruential Generator and more precisely the Multiplicative Congruential Generators.

Empirical validation of PRNGs

Statistical tests

So, a PRNG has to be extensively checked to be used in serious simulations. The tests must check :

- The uniformity of the PRNG output : Kolmogorov-Smirnov, Chi-square, gap tests.
- The independence of the PRNG output : serial test, spectral test, autocorrelation

These tests can be classified into :

- Empirical tests : one test for a property on a large sample produced by the PRNG
- Analytical tests : one check a property on the whole set U of possible output of a PRNG

Linear congruential generators (LCG)

Definition

Let $m > 0$ (the modulus), $a > 0$ (the multiplier) and c (the additive constant) be integers. The linear congruential generator associated with the triplet (m, a, c) is given by :

- $\mathcal{S} = \{0, \ldots, m-1\}$ for $c > 0$ and $\mathcal{S} = \{1, \ldots, m-1\}$ for $c = 0$
- $s_n = T(s_{n-1}) = a.s_{n-1} + c \bmod m$
- $U = \{0, 1/m, \ldots, 1-1/m\}$ for $c > 0$ and $U = \{1/m, \ldots, 1-1/m\}$ for $c = 0$
- $u_n = G(s_n) = s_n/m$

When $c = 0$, these generators are called multiplicative congruencial generators. They are often used due to the extra gain in speed.

Linear congruential generators

Properties

- It is the first class of PRNGs for which a mathematical theory has been built,
- They are fast,
- They are easy to implement,
- They are (still) widespreads (rand(), drand48()...)

But they have serious weaknesses :

- Short period in typical implementations ($m = 2^{32}$) ;
- Bad repartition in high dimension ;
- Lower order bits have a shorter period than full numbers (so don't use $y_n = s_n \bmod M$ if you want integers in $\{0, \ldots, M-1\}$ from integers in $\{0, \ldots, m-1\}$).

Multiplicative congruential generators (MCG)

How to choose m, a ?

- If $m = 2^M, M \geq 3$, the maximal period is 2^{M-2}, attained for all odd seed if $a \bmod 8 \equiv 3$ or 5.
- The period $m-1$ is attained only if m is prime. In this case, the period divides $m-1$, and is $m-1$ if and only if a is a primitive root, that is, $a \neq 0$ and $a^{(m-1)/p} \not\equiv 1 \bmod m$ for each prime factor p of $m-1$.
- More to read in [2].

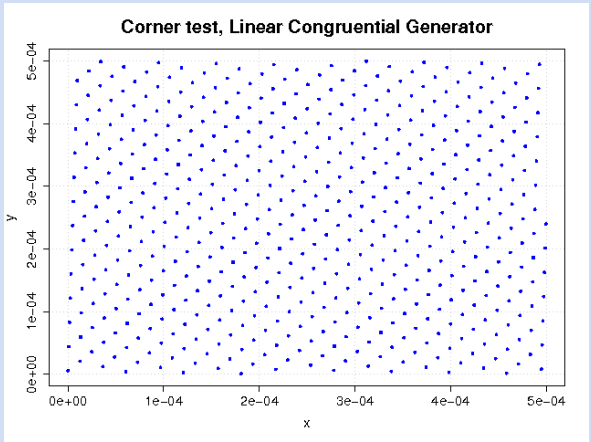
Multiplicative congruential generators

Problem with high dimension

Suppose that one want to sample uniformly the hypercube $[0,1]^d$. The natural way to do this is to consider the points $(u_{dn}, u_{dn+1}, \ldots, u_{(d+1)n-1})$.

One can generate at most m (or $m-1$) points in $[0,1]^d$, which will necessarily be very scarcely distributed in high dimension.

In the case of LCGs or MCGs, these points appear to be distributed according to a very regular pattern : a lattice.



Multiplicative congruential generators

Bad high dimension properties : Marsaglia's theorem, from [3]

Let $d > 0$ be a positive integer. If $\alpha_0, \ldots, \alpha_{d-1}$ is any choice of integers such that :

$$\alpha_0 + \cdots + \alpha_{d-1} a^{d-1} \equiv 0 \bmod m \tag{1}$$

then all points of the form $(u_n, \ldots, u_{n+d-1})$ will lie in the set of parallel hyperplanes defined by the equations :

$$\alpha_0 x_0 + \cdots + \alpha_{d-1} x_{d-1} = 0, \pm 1, \pm 2, \ldots \tag{2}$$

There are at most $|\alpha_0| + \cdots + |\alpha_{d-1}|$ of these hyperplanes which intersect $]0,1[^d$, and there is always a choice of $\alpha_0, \ldots, \alpha_{d-1}$ such that all of the points fall in fewer than $(d!m)^{1/d}$ hyperplanes.

Multiplicative congruential generators

Bad high dimension properties

	$d=3$	$d=4$	$d=5$	$d=6$	$d=7$	$d=8$	$d=9$	$d=10$
$m=2^{23}$	369	119	63	42	32	27	24	22
$m=2^{32}$	2953	566	220	120	80	60	48	41
$m=2^{52}$	300079	18131	3520	1216	582	340	227	166
$m=2^{64}$	4801279	145055	18578	4866	1910	963	573	382

2^{23} for float, 2^{32} for uint32_t, 2^{52} for double, 2^{64} for uint64_t

Multiplicative congruential generators

Demonstration (1/3)

To demonstrate Marsaglia's theorem, we need the following result from number theory :

Theorem [4, Theorem 449] Let $\xi_0, \ldots, \xi_{d-1}$ be d linear forms $\xi_i = \sum_{j=0}^{d-1} k_{i,j} t_i$ with real coefficients, such that :

$$\Delta = \begin{vmatrix} k_{0,0} & \cdots & k_{0,d-1} \\ \vdots & & \vdots \\ k_{d-1,0} & \cdots & k_{d-1,d-1} \end{vmatrix} \neq 0 \qquad (3)$$

There exit integers $t_0, \ldots, t_{d-1}$ not all 0, for which $|\xi_0| + \cdots + |\xi_{d-1}| \leq (d!|\Delta|)^{1/n}$.

Multiplicative congruential generators

Demonstration (2/3)

1. If $\alpha_0 + \cdots + \alpha_{d-1} a^{d-1} \equiv 0 \bmod m$, $\sigma = \alpha_0 u_n + \cdots + \alpha_{d-1} u_{n+d-1}$ is an integer for all n, due to the fact that $u_{n+k} = s_n a^k / m - [a^k s_n / m]$ so :

$$\sigma = \overbrace{s_n(\alpha_0 + \cdots + \alpha_{d-1} a^{d-1})/m}^{\in \mathbb{Z}} - \overbrace{(\alpha_0 [s_n/m] + \cdots + \alpha_{d-1} [a^{d-1} s_n/m])}^{\in \mathbb{Z}}$$

2. Thus, the points $(u_n, \ldots, u_{n+d-1})$ must lie in one of the hyperplanes $\alpha_0 x_0 + \cdots + \alpha_{d-1} x_{d-1} = 0, \pm 1, \pm 2, \ldots$;

3. The number of such hyperplanes which intersect $]0,1[^d$ is at most $|\alpha_1| + \cdots + |\alpha_d|$ by inspection : $\sum_{i \in I_1} \alpha_i < \alpha_0 x_0 + \cdots + \alpha_{d-1} x_{d-1} < \sum_{i \in I_2} \alpha_i$ where $I_1 = \{i | \alpha_i < 0\}$ and $I_2 = \{i | \alpha_i > 0\}$;

Multiplicative congruential generators

Demonstration (3/3)

4. There exist a set $\alpha_0, \ldots, \alpha_{d-1}$ not all zeros with $|\alpha_0| + \cdots + |\alpha_{d-1}| \leq (d!m)^{1/d}$ because (1) is equivalent to show that there exist integers $t_0, \ldots, t_{d-1}$ not all zeros such that :

$$|mt_0 - at_1| + |t_1 - at_2| + \cdots + |t_{d-2} - at_{d-1}| + |t_{d-1}| \leq (d!m)^{1/d}$$

by writing that

$$(\alpha_0, \ldots, \alpha_{d-1}) = (t_0, \ldots, t_{d-1}) \begin{pmatrix} m & 0 & \cdots & \cdots & 0 \\ -a & 1 & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -a & 1 \end{pmatrix}$$

and it is true as stated in the preliminary theorem (the determinant of the matrix is m).

Statistical tests

The gap test

This test is one of the most basic ones. It aims at testing if a PRNG associated with a measure μ produces ouputs U that cover evenly the support of μ.

More precisely :

- Let $\rho = 1/\text{card}U \ll 1$ be the theoretical optimal resolution of a PRNG ;
- Find $\mu(I^*) = \max_{I=]s_i,s_j[, I\cap U=\emptyset} \mu(I)$, where $s_i, s_j \in U$;
- Check that $\mu(I^*) = \mathcal{O}(\rho)$: the PRNG does not leave significant gaps in the support of μ.

One can note that a LCG with full period is perfect with regard to this test, as it has a gap value of ρ, or $\rho/(1-\rho)$ for MCGs.

Statistical tests

The spectral test

The spectral test is specifically designed to detect non-uniformity in the multidimensional repartition of U in $[0,1]^d$. It was designed after a period of massive use of LCGs that where very poorly designed (as the famous RANDU shipped with IBM computers in the 80'). The test is defined by :

- Compute the quantity :

$$\nu_d = \min \left\{ ||\alpha||_2 \ \middle| \ \sum_{i=0}^{d-1} \alpha_i a^i \equiv 0 \bmod m \right\} \tag{4}$$

for $d = 2, \ldots, d_{max}$. The quantity ν_d can be interpreted as the number of bits of accuracy of the LCG in dimension d : it is the largest gap between adjacent hyperplanes in a family of parallel hyperplanes that covers the $d-$ dimensional points generated by the LCG.

Statistical tests

The spectral test

- Compute the normalized quantity :

$$\mu_d = \frac{\pi^{d/2}\nu_d^d}{\Gamma(d/2+1)m} \tag{5}$$

- The test is successful if $\mu_d > 0.1$ for $d = 2, \ldots, 6$.

This test has been experimentaly proved to be very effective in separating the good PRNGs from the bad ones.

Other related PRNGs

LCGs and derived not better than MRGs

- Marsaglia's theorem can be extended to more general LCGs, and even if the details change, the basic fact remains : LCGs are bad for high dimensional applications unless one is ready to use a HUGE value for m ($m > 10^{1500}$), for which one can work in dimension $d = 100$ without too much care.
- Generators that chain several LCGs suffer from the same defect : it is just a way to adjust m to higher values, but starting from $m = 2^{32}$ one cannot attain sufficient resolution without an obscene number of steps. Indeed, the combination of k LCGs with moduli $m_1, \ldots, m_k$ has a maximum period of $P = \frac{(m_1-1)\ldots(m_k-1)}{2^{k-1}} \simeq 2^{32(k-1)}$, which need $k = 157$ different generators to attain the needed resolution !

Better than LCGs

SIMD-oriented Fast Mersenne Twister (SFMT), from [5]

Three of the most important properties of a PRNG are :

- a huge period ;
- good equirepartition properties even in high dimension ;
- fast generation of random numbers.

The SIMD-oriented Fast Mersenne Twister fulfill these needs with a period of $2^{19937} - 1 \simeq 10^{6001}$ (can be easily extended to $2^{216091} - 1 \simeq 10^{65050}$), a provable equirepartition in dimension up to 382 (resp. 4077) as a double-precision floating-point uniform random numbers generator.

SFMT details

An efficient Linear Feedback Shift Register generator

The state space S of this generator is $(\mathbb{F}_2^w)^N$, the transition function T is linear and very sparse and can be seen as an N terms linear recurrence over $\mathbb{F}_2^w$. An element s in S is seen as an N dimensional vector $(x_0, \ldots, x_{N-1})$ over $\mathbb{F}_2^w$, each x_k is called a register and the linear transformation takes the form :

$$\begin{aligned}((x_n)_0, \ldots, (x_n)_{N-1}) &= ((x_{n-1})_1, \ldots, (x_{n-1})_{N-1}, z) \\ \text{with } z &= g((x_{n-1})_0, \ldots, (x_{n-1})_{N-1}))\end{aligned} \tag{6}$$

with g a sparse linear function. For SFMT, $w = 128$ in order to store the x_k in SIMD (Single Instruction, Multiple Data) registers of modern CPUs, and the function g is such that :

$$g(x_0, \ldots, x_{N-1}) = x_0\mathbf{A} + x_M\mathbf{B} + x_{N-2}\mathbf{C} + x_{N-1}\mathbf{D} \tag{7}$$

SFMT details

An efficient Linear Feedback Shift Register generator

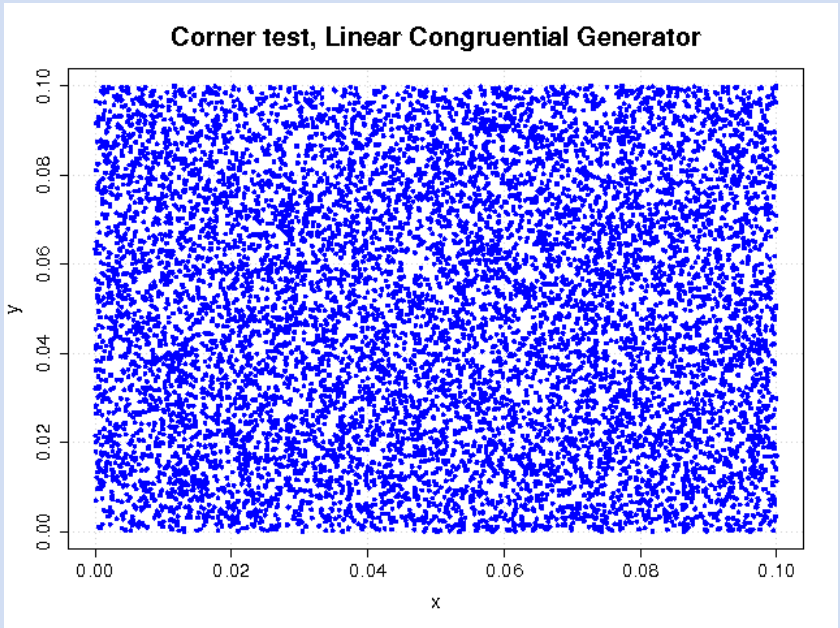
The parameters M, N, **A**, **B**, **C** and **D** have been chosen in order to have a period multiple of $2^{19937} - 1$ and provide equirepartition in high dimension, which leads to :

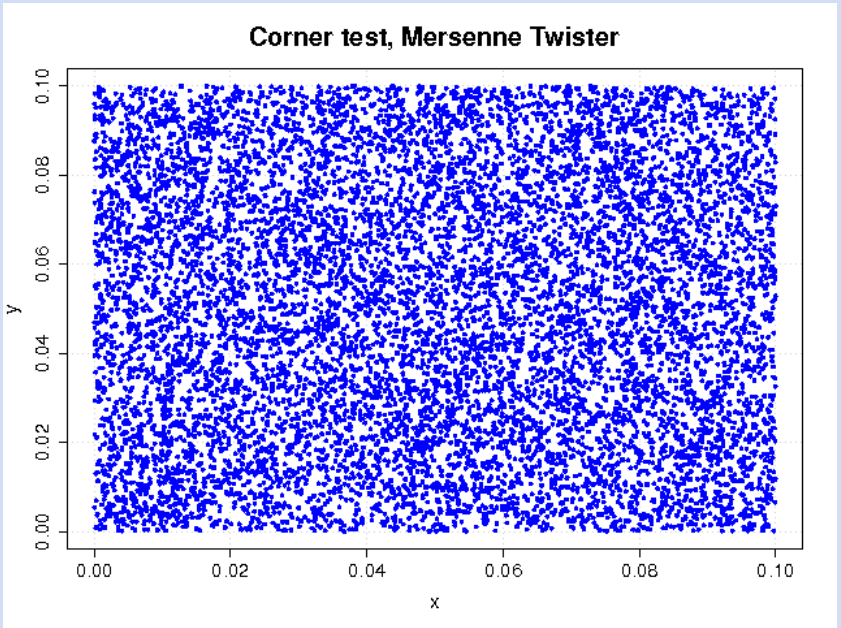
- $N = 156$ and $M = 122$,
- $x\mathbf{A} = (x \overset{128}{<<} 8) \wedge x$,
- $x\mathbf{B} = (x \overset{32}{>>} 11) \oplus (BFFFFFF6 \;\; BFFAFFFF \;\; DDFECB7F \;\; DFFFFFEF)$,
- $x\mathbf{C} = (x \overset{128}{>>} 8)$,
- $x\mathbf{D} = (x \overset{32}{<<} 18)$.

Working on a SIMD-oriented computer, the above computations can be implemented in a very concise and very effective way.

Corner test : realizations in $[0, a]^2$

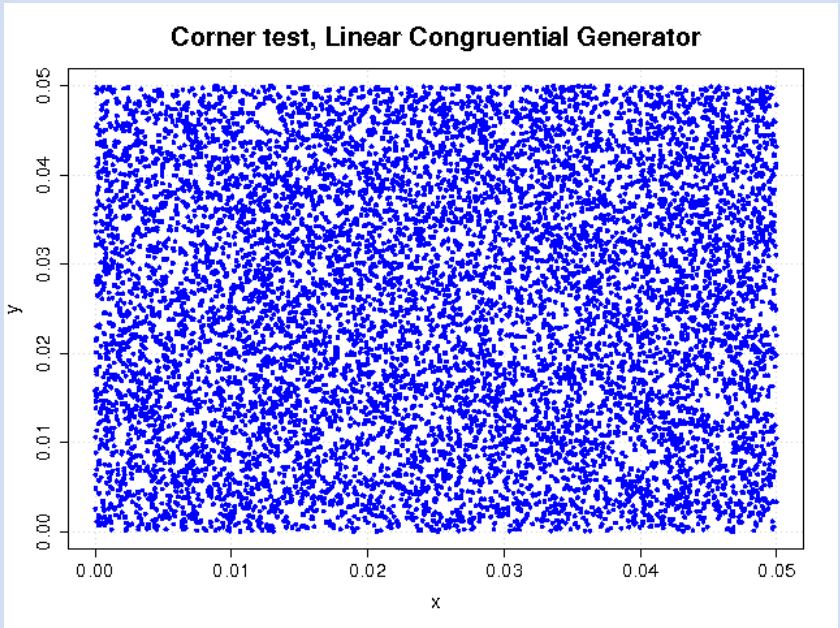
$a = 0.1$, 10^4 realizations

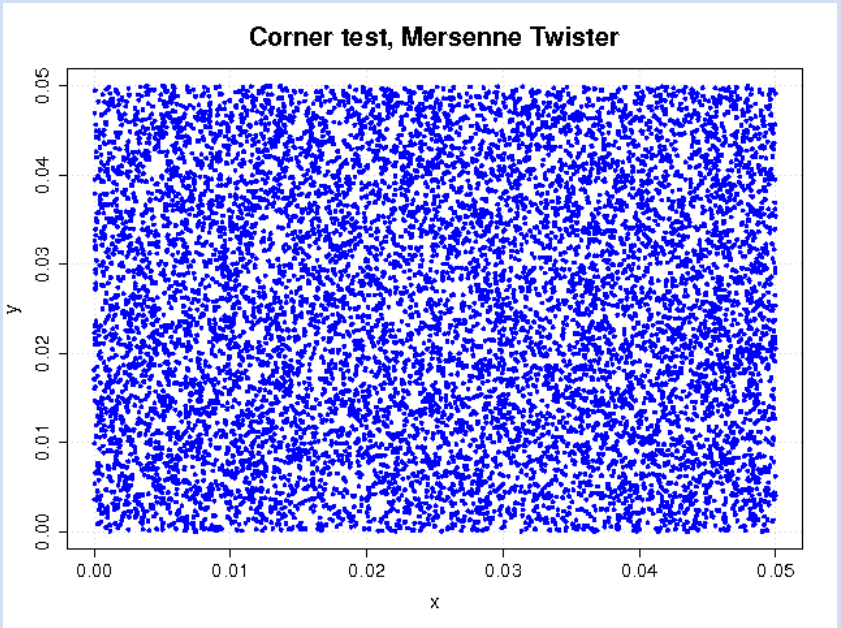




Corner test : realizations in $[0, a]^2$

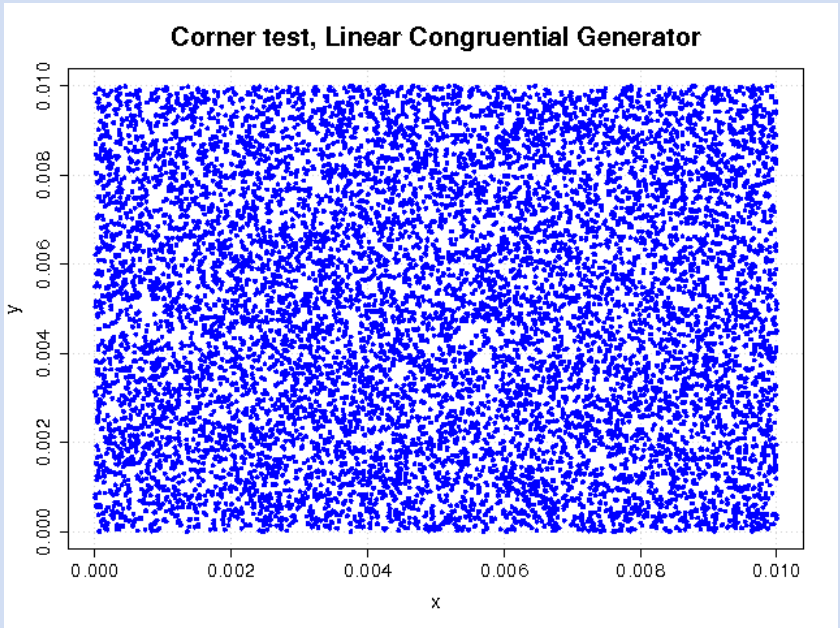
$a = 0.05$, 10^4 realizations

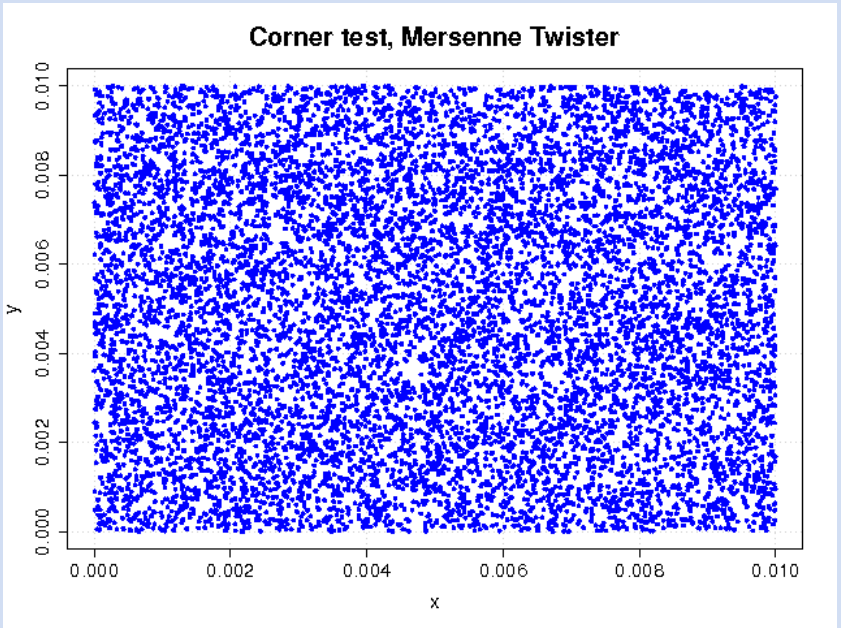




Corner test : realizations in $[0, a]^2$

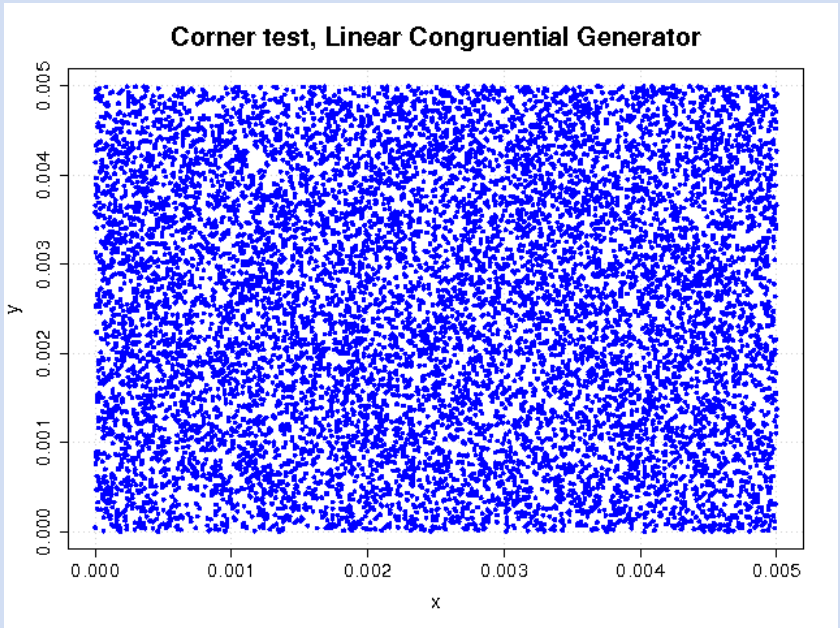
$a = 0.01$, 10^4 realizations

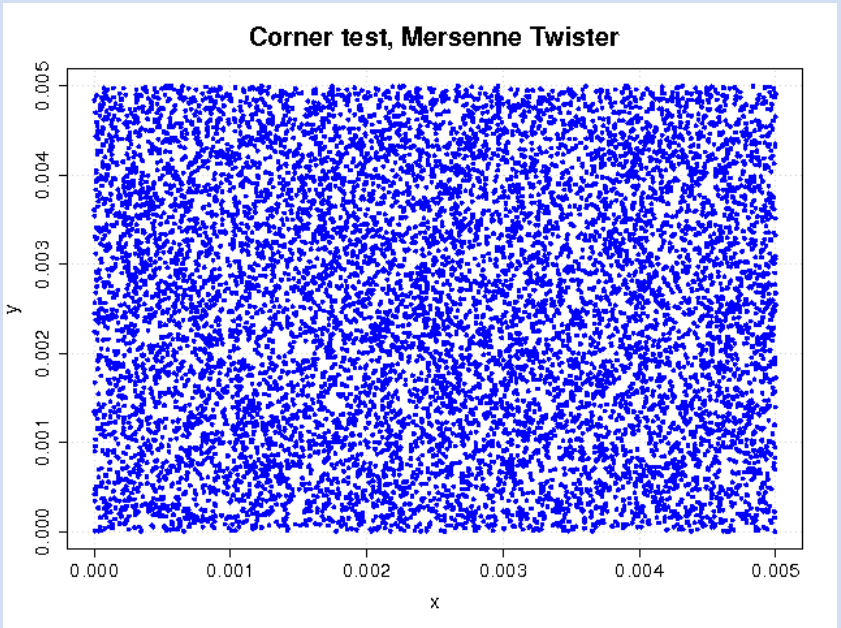




Corner test : realizations in $[0, a]^2$

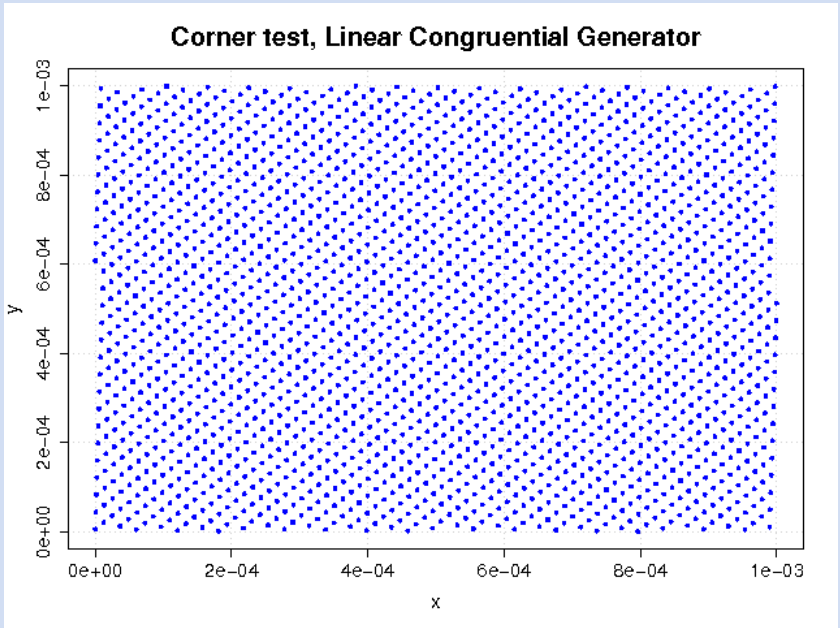
$a = 0.005$, 10^4 realizations

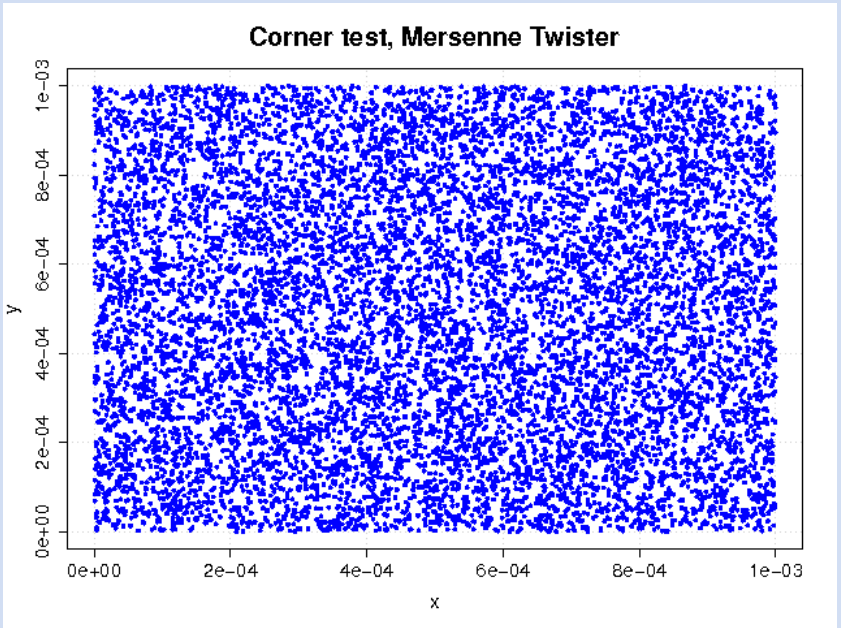




Corner test : realizations in $[0, a]^2$

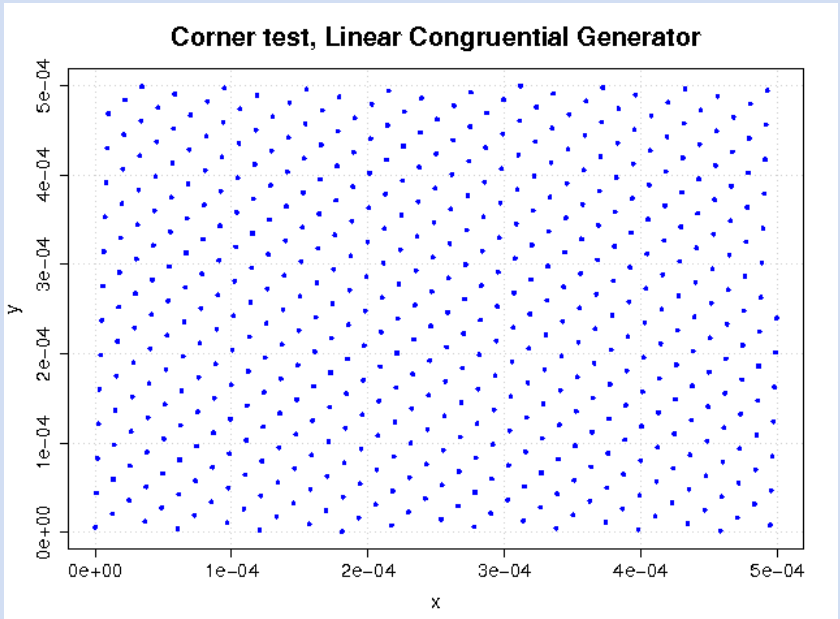
$a = 0.001$, 10^4 realizations

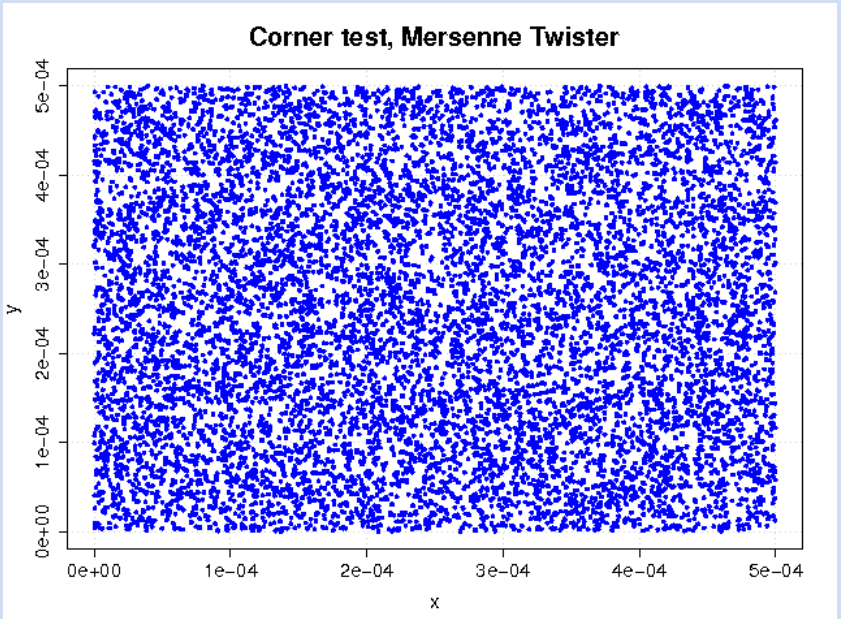




Corner test : realizations in $[0, a]^2$

$a = 0.0005$, 10^4 realizations





From uniform to non-uniform

A lot of methods

- CDF inversion
- Rejection (with or without squeeze)
- Transformation (with the ratio of uniforms as a special case)
- Mix of all of that...

The objective of these methods is to express a random variable X of a given non-uniform distribution as the output of an algorithm that use one or more iid variables $U_1, \ldots, U_N$ uniformly distributed over $[0, 1]$. For a broad overview on these methods, see [6].

CDF inversion

Principle and good properties

If U is uniformly distributed over $[0,1]$ and if F is a CDF, $X = F^{-1}(U)$ is a random variable distributed according to F. Here,
$F^{-1}(u) = \inf_{t \in \mathbb{R}}\{F(t) \geq u\}$ is the generalized inverse of F.
This well-known method has some interesting properties :

- It needs only one uniform realization per non-uniform realization ;
- Same dependence structure as the underlying stream of uniform realizations ;
- Same gap test value as the underlying stream of uniform realizations ;

CDF inversion

Drawbacks

- The resulting generator is rather slow for most of the usual distributions (normal, gamma etc.) ;
- The realizations are truncated to a rather small interval due to the finite precision of the computer arithmetic : $[-5.17, 5.17]$ for floats, $[-8.13, 8.13]$ for doubles.

Rejection with squeeze

Principle

Let f be the PDF of the distribution we want to sample from, and suppose that there exist a PDF g (the rejection distribution) which is easy to sample and fast to evaluate, a function h (the squeeze function) very cheap to evaluate and a real $\lambda > 0$ such that :

$$\forall x \in \text{supp}(f), h(x) \geq f(x) \; geq\lambda g(x) \tag{8}$$

Then, the algorithm is :

1. Generate Y according to g ;
2. Generate U uniformly over $[0, \lambda g(Y)$;
3. If $U < h(Y)$, accept Y as a realization of f ;
4. If $U < f(Y)$, accept Y as a realization of f ;
5. else go back to step 1 ;

Rejection with squeeze

Properties

This algorithm uses at least two realizations of the underlying uniform PRNG, and its efficiency is measured in term of the mean number of iterations needed to produce one non-uniform realization. The squeeze function is here only to postpone the potentially expansive test in step 4.

The attainable range is extended compared to the CDF inversion method, as one have to compare functions that takes values close to 0 instead of 1. For the normal distribution generated using exponential tails, one can attain the range $[-13.47, 13.47]$ for floats and $[-37.62, 37.62]$ for doubles.

The gap test can be severely degraded if the rejection distribution is far from f : this method change the frequency of the realizations of g, not their values, so if they are not well distributed with respect to f it can lead to significant gaps.

Transformation method

Principle

A random variable X one want to sample can sometimes be expressed as a function of one or several other independent random variables $\Xi_1, \ldots, \Xi_N$ not necessarily identically distributed, but that can all be easily sampled :

$$X = \phi(\Xi_1, \ldots, \Xi_N) \tag{9}$$

The simulation method is then straitforward : sample each of the Ξ_i, then compute a realization of X using ϕ.

Transformation method

Properties

- Many possibilities to express X as a function of Ξ_i ;
- Straightforward implementation once the decomposition has been found ;
- Can be expansive in term of calls to the uniform PRNG for one realization of X, even if it is sometimes possible to get several iid realizations of X for one realization of the Ξ_i, using differente functions.
- The effective range of the resulting PRNG can be limited. For example, for the normal distribution, the Box-Muller transformation leads to the range $[-5.64, 5.64]$ for floats and $[-8.49, 8.49]$ for doubles.

The gap test can be severely degraded if the uderlying uniform PRNG has a specific structure that interact with the structure of ϕ.

LCGs and the ratio of uniforms method

The ratio of uniforms method

This method is a mix between the transformation method and the rejection method, that leads to fast and generic non-uniform PRNGs.

Let (U, V) be uniformly distributed over $C = \{(u, v) \mid 0 \leq u \leq \sqrt{c.f(v/u)}\}$, then $X = V/U$ has density f. If f has sub-quadratic tails, the area of C is bounded and can be enclosed in a rectangle $[0, u^+] \times [v^-, v^+]$. It is then possible to sample (U, V) by the rejection method, and it is very often possible to derive efficient squeeze functions.

LCGs and the ratio of uniforms method

Gap default when combining LCGs and the ratio of uniforms method, from [7]

If (U, V) is obtain using consecutive output of a LCG(m, a, c), as these points will be contains within a lattice, the possible values for the ratio V/U will inherit a very special structure. It is then possible to show that :

Th. 1 If $C = [0, u^+] \times [-v^+, v^+]$, then there exist a gap g such that $\mu(g) \geq \frac{1}{\sqrt{m}} \sqrt{\frac{\sqrt{3}}{32}}$

Th. 2 If $C = [0, u^+] \times [v^-, v^+]$, then there exist a gap g such that $\mu(g) \geq \frac{1}{\sqrt{m}} \sqrt{\frac{\sqrt{3}}{16} \frac{q_2}{q_2^2+1}}$ where $q_2 = |e_1|/|e_2|$ is the Beyer quotient of the lattice, (e_1, e_2) being the Minkowski-reduced basis of the lattice.

These lower bounds are much worse than the expected $\mathcal{O}(1/m)$!

LCGs and the ratio of uniforms method

Gap for usual distributions

We consider the Cauchy, normal and exponential distributions for LCGs with $m \simeq 2^{32}$ and q_2 varying from 0.1 to 1. We compute the lower bound of $\mu(g)$ scaled by 10^6 :

q_2	Cauchy	Normal	Exponential
0.1	4.52	2.08	2.98
0.5	4.52	4.17	5.58
1.0	4.52	4.65	6.12

If you are dealing with (not so) rare events, don't use the combination LCG/ratio of uniforms !

Fast and high-quality normal PRNG

The ziggurat method (Marsaglia, Doornik, see [8])

The ziggurat method is an innovative use of the rejection method with squeeze. Its key characteristics are :

- allow to sample symmetric univariate continuous random variables
- generate fast and high quality random deviates

The main idea is to use rejection with squeeze with respect to horizontal histograms, with a specific treatment of the tail of the distribution.

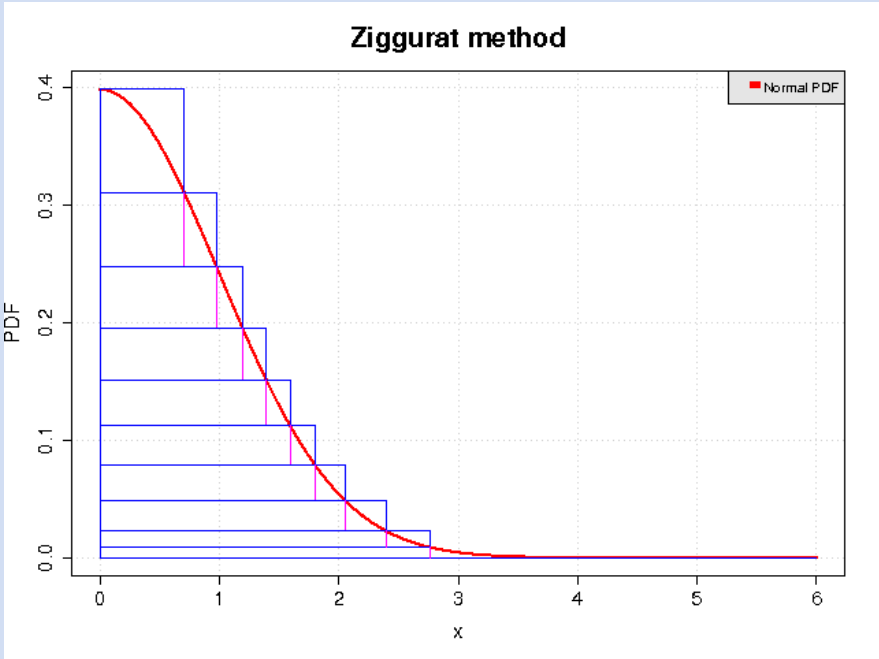
We detail this method in the case of the standard normal distribution.

Fast and high-quality normal PRNG

The ziggurat method (Marsaglia)

How to build the ziggurat ?

- Restrict the PDF f to $x \geq 0$;
- Cover its graph with horizontal rectangular bands
- The bottom band contains also the tail of PDF ;
- All these bands have the same area V ;
- Number these bands from 1 (top) to N (bottom) ;
- Consider the inner horizontal histogram as an inner squeeze function.

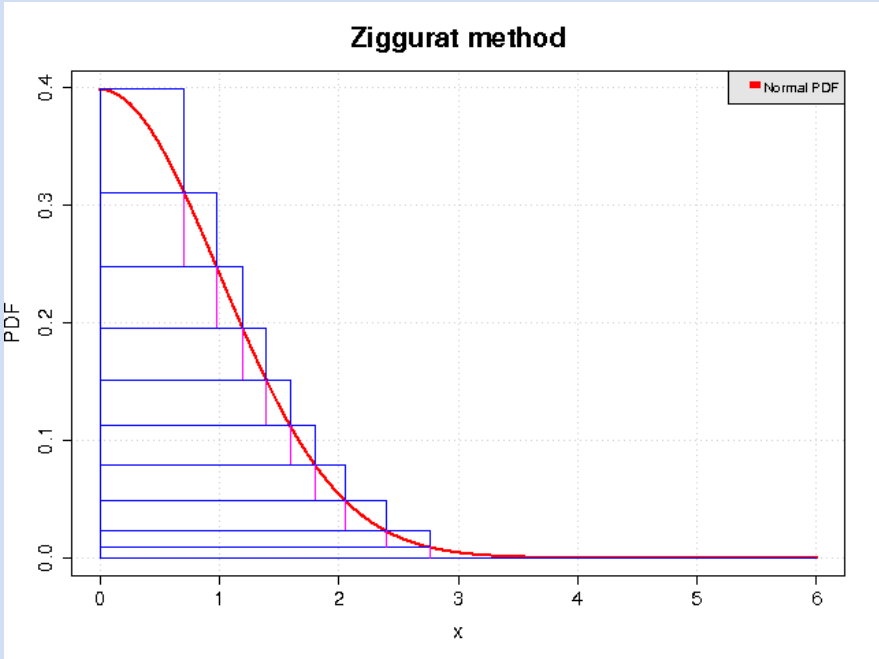


Fast and high-quality normal PRNG

The ziggurat method (Marsaglia)

How to use it ?

1. Generate a band number i uniformly distributed over $\{1, \ldots, N\}$;
2. If $i < N$ generate u_i uniformly in $[0, x_i]$ where x_i is the length of the covering band, else generate u_i uniformly in $[0, V/f(x_N)]$.
3. If $u_i \leq x_{i-1}$ (with $x_0 = 0$), accept u_i
4. Else
    1. if $i = N$, generate u_i in the tail e.g. using Marsaglia's method ;
    2. else use simple rejection in the wedge.



Fast and high-quality normal PRNG

Generating realizations from the tail of the normal distribution, Marsaglia's method

To generate a random variate X with density proportional to $\exp(-x^2/2)1_{\{x \geq a\}}$, do :

1. Generate U_1 and U_2, two iid random variables uniform on $]0, 1]$;
2. Define $X = \frac{1}{a}\log(U_1)$ and $Y = \log(U_2)$;
3. If $x^2 < -2y$ return $a - X$. Note : $X < 0$

The joint density of the pairs (X, Y) obtained after a successful step 3 has a joint density of the form :

$$p_{XY}(x, y) = \frac{K}{a}\exp(x/a)1_{\{x<0\}}\exp(y)1_{\{y<0\}}1_{\{x^2+2y<0\}} \qquad (10)$$

from which we deduce the marginal density of X and can check that $a + X$ has the correct density.

Fast and high-quality normal PRNG

The ziggurat method (Marsaglia)

- The acceptation ratio is $\rho = \frac{\sqrt{2\pi}}{2NV}$, so we use large values for N, such as $N = 128$, for which $\rho \simeq 98.78\%$.
- With this setting, $a \simeq 3.4451$ and the aceeptation ratio of MArsaglia's method for generating tail normal random deviate is 98.05%

Further material

Does a bad PRNG matter ?

- There is no need to have a perfect PRNG to obtain the convergence, most of the time it suffice to have low discrepancy ;
- The problem is more on the justification of the convergence of the algorithms and the quantification of the confidence interval (e.g. no TCL with low discrepancy sequences)
- The only very serious default is the lack of space filling in high dimension : one can completely miss a region of interrest and thus severely underestimate the estimation of a quantity of interrest.

References

1. Jerry Banks (ed.), "Handbook of Simulation : Modelling, Estimation and Control", Wiley, 1998.
2. Donald E. Knuth, "The Art of Computer Programming, Vol. 2 : Seminumerical algorithms, 3rd Ed.", Addison-Wesley, 1998.
3. George Marsaglia, "Random numbers fall mainly in the planes", Proc N.A.S, 1968.
4. G. H. Hardy, E. M. Wright, "An introduction to the theory of numbers, 4th ed.", Oxford University Press, 1975.
5. Mutsuo Saito, Makoto Matsumoto, "SIMD-oriented Fast Mersenne Twister : a 128-bit Pseudorandom Number Generator", research report, Dep. of Math, Hiroshima University, 2006.
6. Luc Devroye, "Non-Uniform Random Variate Generation", Springer, 1986.
7. Wolfgang Hörmann, "A Note on the Quality of Random Variates Generated by the Ratio of Uniform Method", ACM Transaction on Modelling and Computer Simulation, Vol 4, 1.
8. Jurgen A. Doornik, "An Improved Ziggurat Method to Generate Normal Random Samples", working paper, http ://www.doornik.com/research.html.

Conclusion

Numerous communities in numerical applied probabilities

- The world of uniform PRNGs makers (those in the state of sin) : arithmetically inclined, know CPU details, IEEE 754 norm ;
- The world of non-uniform PRNGs makers (they believe on truthful uniform PRNGs provided by the first community) : elementary probabilities, analysis background, sometimes also members of the first community but not so frequently.
- The world of applied probabilities problem solvers (you !) : high probability knowledge, in link with the previous community, but when it comes to implement an algorithm, sometime believe that the basic `rand()` function is thruthful ;-) !

Conclusion

What should be remembered ?

- Never use `rand()` for something else than initializing the position of monsters in a Dungeon and Dragon video game !
- Check that the uniform PRNG has a sound theoretical background and a successful record track in real-life applications.
- Check that there is no bad interaction between the uniform PRNG and the non-uniform PRNG !
- Adapt the precision of your uniform PRNG to the precision of your computation : it is nonsens to use double precision in the high-level algorithms if the uniform PRNG gives only float output !

Many thanks for your attention !

Your questions and comments are welcome !