5.4 BETTER RANDOM NUMBERS

Variance reduction methods and antithetic sampling

Monte-Carlo is easy to understand and implement, and adequate in virtually any valuation context, but it is expensive in CPU time and its convergence in the number of paths is slow. A vast amount of research, collectively known as *variance reduction methods*, conducted in the past decades, attempted to accelerate its convergence.

Perhaps the simplest variance reduction technique is *antithetic sampling*. For every path generated with the uniform vector $U = (U_1, ..., U_D)$, generate another one with its antithetic:

$$(1 - U_1, ..., 1 - U_D)$$

Not only does antithetic sampling balance the paths with a desirable symmetry, it also generates two paths for the cost of one random vector.

When the scheme consumes Gaussian numbers, antithetic sampling is even more efficient: for every path generated with the Gaussian vector G, generate another one with $-G$, guaranteeing that the empirical mean of all components in the Gaussian vector is zero. This is a simple case of a family of variance reduction methods called *control variates*, where the simulation enforces that the empirical values of sample statistics such as means, standard deviations, or correlations match theoretical targets. In addition, we get two paths for the cost of one Gaussian vector, saving not only the generation of the uniforms, but also the not-so-cheap Gaussian transformations.

Antithetic sampling is a simple notion and translates into equally simple code. We implement antithetic sampling with mrg32k3a in the next chapter. We will mention other variance reduction methods, but we don't discuss them in detail, referring readers to chapter 4 of [62]. What we *do* cover is a special breed of "random" numbers particularly well suited to Monte-Carlo simulations.

Monte-Carlo as a numerical integration

We reiterate the sequence of steps involved in a Monte-Carlo valuation.[11] We generate and evaluate N paths, each path number i following the steps:

1. Pick a uniform random vector U_i in the *hypercube* in dimension D:

$$U_i = (u_{ik}) \in (0, 1)^D$$

[11] Assuming the simulation scheme consumes Gaussian numbers, as is almost always the case.

2. Turn U_i into a Gaussian vector with the application of the inverse cumulative Gaussian distribution N^{-1} to its components:

$$N_i = (n_{ik}) = G(U_i) = (N^{-1}(u_{ik}))$$

3. Feed the Gaussian vector N_i to the simulation scheme to produce a path X^i for the model's state vector X over the simulation timeline:

$$X^i = (X^i_{T_j})_{0 \le j \le M} = v(N_i)$$

4. Use the model's mapping to turn the path X^i into a collection of samples S^i over the event dates:

$$S^i = (S^i_{T_j})_{1 \le j \le J} = \zeta(X^i)$$

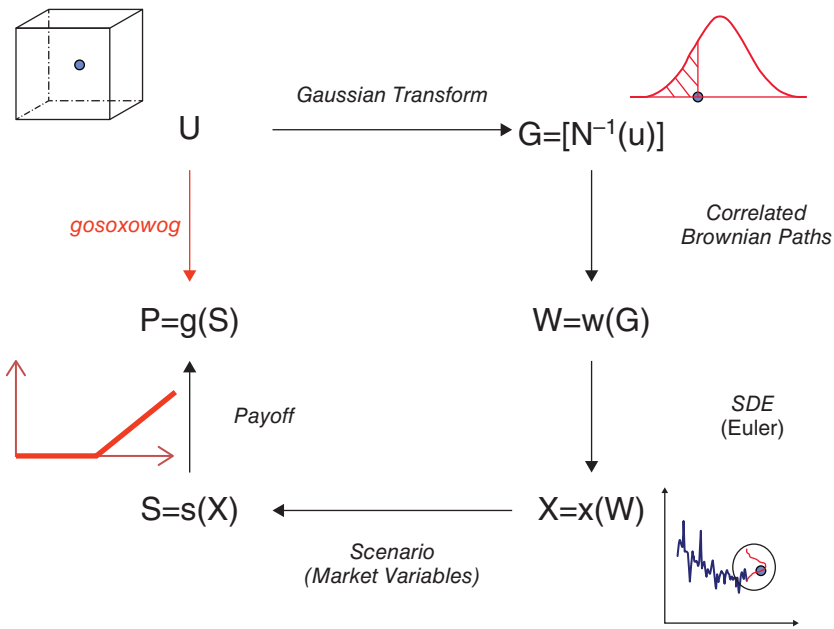
5. Compute the product's payoff g_i on this path:

$$g_i = g(S^i)$$

Hence, g_i is obtained from U_i through a series of successive transformations. Therefore:

$$g_i = h(U_i)$$

where $h \equiv g \circ \zeta \circ v \circ G$:



The MC estimate of the value is:

$$\overline{V_0} = \frac{1}{N}\sum_{i=1}^{N} g_i = \frac{1}{N}\sum_{i=1}^{N} h(U_i)$$

The *theoretical* value (modulo discrete time) is:

$$V_0 = E[h(U)] = \int_{(0,1)^D} h(u)du$$

Hence, Monte-Carlo is nothing more, nothing less, than the numerical approximation of the integral of a (complicated) function $h : (0,1)^D \to \mathbb{R}$ with a sequence $(U_i)_{1\leq i\leq N}$ of random samples drawn in the hypercube.

Koksma-Hhlawka inequality and low-discrepancy sequences

A random sequence of points is not necessarily the optimal choice for numerical integration. Standard numerical integration schemes, covered, for example, in [20], work in low dimension. Dimension is typically high in finance. With Euler's scheme, it is the number of time steps times the number of factors. For a simple weekly monitored 3y barrier in Black and Scholes or Dupire's model, the dimension is 156. Typical dimension for exotics is in the hundreds. For xVA and other regulations, it its generally in the thousands or tens of thousands, due to the large number of factors required to accurately simulate the large number of market variables affecting a netting set.

Numerical integration schemes would not work in such dimension, but there exists one helpful mathematical result known as the *Koksma-Hhlawka inequality*:

$$|\overline{V_0} - V_0| \leq V(h)D[(U_i)_{1\leq i\leq N}]$$

where the value and its estimate are defined above, the left-hand side is the absolute simulation error, V is the variation of h, something we have little control about, and D is the *discrepancy* of the sequence of points $(U_i)_{1\leq i\leq N}$, a measure of how well it fills the hypercube. Formally:

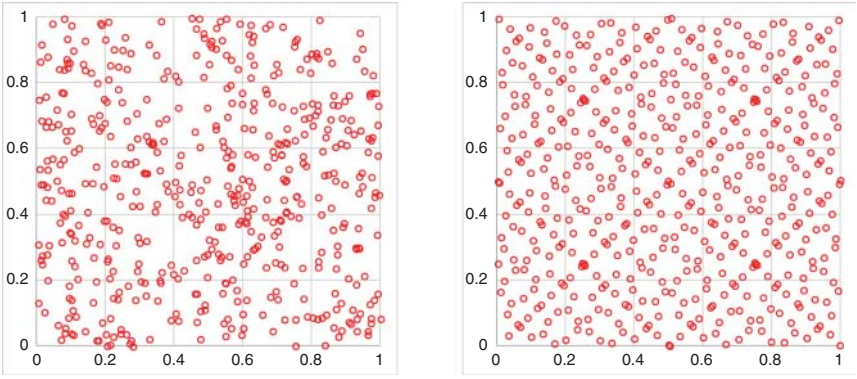
$$D[(u_i)_{1\leq i\leq N}] = \sup_{E=[0,t_1)\times\ldots\times[0,t_D)} \left| \frac{1}{N}\sum_{i=1}^{N} 1_{\{u_i\in E\}} - VOL(E) \right|$$

where $VOL\{E = [0,t_1)\times\ldots\times[0,t_D)\} = \prod_{k=1}^{D} t_k$ is the volume of the box E. Hence, the discrepancy is the maximum error on the estimation of volumes, a measure of how *evenly* the sequence fills the space.

The Koksma-Hhlawka inequality nicely separates the characteristic V of the integrated function and the discrepancy D of the sequence of points for the estimation. It follows that random sampling is *not* optimal. Random sampling may cause clusters and holes in the hypercube, resulting in poor evenness and high discrepancy. It is best to use sequences of points

purposely designed to minimize discrepancy. A number of such sequences were invented by Van Der Corput, Halton, or Faure, but the most successful one, by far, was designed by Sobol in 1967 USSR [71].

The following picture provides a visual intuition of the discrepancy of Sobol's sequence. It compares 512 points drawn randomly in $(0,1)^2$ on the left to the 512 first Sobol points in dimension two on the right:



It is clear to the naked eye that Sobol's sequence effectively avoids the holes and clusters inevitable with a random sequence and fills the hypercube with a better "evenness." The mathematical notion of discrepancy formalizes the notion of "evenness," and the Koksma-Hhlawka inequality demonstrates the intuition that better evenness results in a more accurate integration.

Sobol's sequence

The construction speed of Sobol's sequence was massively improved in 1979 by Antonov and Saleev, whose algorithm generates successive Sobol points extremely fast, with just a few low-level bit-wise operations. It is this implementation that is typically presented in literature, including here. Over the past 20 years, Jaeckel [63] and Joe and Kuo [72], [73] performed considerable work on Sobol's *direction numbers*[12] so that the sequence could be practically applied in the very high dimension familiar to finance, achieving remarkable results. Sobol's sequence (with Antonov and Saleev's optimization and Jaeckel or Joe and Kuo's direction numbers) became a best practice in financial applications, which it remains to this day.

Sobol's sequence is not really a sequence of points in the hypercube $(0,1)^D$, but a collection of D sequences of numbers in $(0,1)$. The sequences of scalar numbers (x_i^d) on each axis d of the sequence is self-contained, and the first D coordinates of a sequence in dimension $D' > D$ exactly correspond to the sequence in dimension D.

[12]Explained shortly.

Sobol generates sequences of *integers* (y_i^d) between 0 and $2^{32} - 1$, so the ith number on the axis d is:

$$x_i^d = \frac{y_i^d}{2^{32}} \in (0, 1)$$

The integers (y_i^d) on a given axis d are produced by recursion: $y_0^d = 0$ (the point 0 is not valid in Sobol; the first valid point is the point number 1) and:

$$y_{i+1}^d = y_i^d \oplus DN_{J_i}^d$$

where $\oplus$ denotes the bit-wise exclusive or (xor), J_i is the rightmost 0 bit in the binary expansion of i, and $\{DN_j^d, 0 \leq j \leq 31\}$ are the 32 *direction numbers* for the sequence number d.

The rightmost bit of every even number is 0; hence, once every two points, Sobol's recursion consists in xor-ing the first direction number DN_0^d to its state variable y^d. The operation xor is associative and has the property that:

$$x \oplus x = 0$$

so DN_0^d flicks in and out of the state y^d every two points. For the same reason, DN_1^d flicks in and out every four points, DN_2^d flicks in and out every eight points, and, more generally, DN_k^d (with $0 \leq k \leq 31$) flicks in and out every 2^{k+1} points. Sobol's sequence is illustrated below:

i	DN0	DN1	DN2	DN3	y
0	0	0	0	0	0
1	1	0	0	0	DN0
2	1	1	0	0	DN0 + DN1
3	0	1	0	0	DN1
4	0	1	1	0	DN1 + DN2
5	1	1	1	0	DN0 + DN1 + DN2
6	1	0	1	0	DN0 + DN2
7	0	0	1	0	DN2
8	0	0	1	1	DN2 + DN3
9	1	0	1	1	DN0 + DN2 + DN3
10	1	1	1	1	DN0 + DN1 + DN2 + DN3
11	0	1	1	1	DN1 + DN2 + DN3
12	0	1	0	1	DN1 + DN3
13	1	1	0	1	DN0 + DN1 + DN3
14	1	0	0	1	DN0 + DN3
15	0	0	0	1	DN3

More generally, for $i < 2^k$, the state y_i^d is a combination of the k first direction numbers:

$$y_i^d = \sum_{j=0}^{k-1} a_j^i DN_j^d$$

where the weights a_j^i are either zero or 1: a_0 flicks every 2 points, a_1 every 4 points, a_2 every 8 points, a_j every 2^{j+1} points. It follows, importantly, that the 2^k first points in the sequence $y_i^d, 0 \le i < 2^k$ span *all the* 2^k *possible combinations* of the k first direction numbers, each combination being represented exactly once.

It follows that the Sobol scalar x_i^d in $(0, 1)$ is:

$$x_i^d = \frac{y_i^d}{2^{32}} = \sum_{j=0}^{k-1} a_j \frac{DN_j^d}{2^{32}} = \sum_{j=0}^{k-1} a_j D_j^d$$

where the $D_j^d \equiv \frac{DN_j^d}{2^{32}}$ are the normalized direction numbers.

Direction numbers on the first axis

On the first axis $d = 0$, the direction numbers are simply:

$$DN_k^0 = 2^{31-k}$$

and it follows that the normalized numbers are:

$$D_k^0 = \frac{1}{2^{k+1}}$$

so the sequence of the 32 normalized direction numbers D_k^0 is $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{16}$, etc. Or, bit-wise, the kth direction number DN_k^0 has a bit of 1, k spaces from the right, and 0 everywhere else:

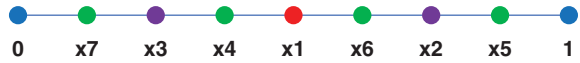
	kth direction number of the first axis							
space	31	30	29	...	**31-k**	..	1	0
bit	0	0	0	0	**1**	0	0	0
k	0	1	2	...	**k**		30	31
DN	2^31	2^30	2^29	...	**2^(31-k)**	...	2	1
D	1/2	1/4	1/8	...	**2^(-k-1)**	...	2^(-31)	2^(-32)

0 everywhere on the left — The kth direction number has bit 1 on space k — 0 everywhere on the right

It follows that the sequence of numbers on the first axis is:

$$\begin{aligned} x_0^0 &= 0 \\ x_1^0 &= \frac{1}{2} \\ x_2^0 &= \frac{1}{2} + \frac{1}{4} = \frac{3}{4} \\ x_3^0 &= \frac{1}{4} \\ x_4^0 &= \frac{1}{4} + \frac{1}{8} = \frac{3}{8} \\ x_5^0 &= \frac{1}{2} + \frac{1}{4} + \frac{1}{8} = \frac{7}{8} \\ &\dots \end{aligned}$$

or, graphically:



where we see that the normalized direction number $1/2^{k+1}$ does not kick in before all the combinations of the $1/2^j$ for $1 \leq j \leq k$ were exhausted. It is visible on the chart that x_1 goes in the middle of the axis, x_2 and x_3 go in the middle of the spaces on the left and right of x_1, and the following four numbers go straight in the middle of the spaces left in the previous sequence. The same pattern carries on indefinitely, progressively and incrementally filling the axis in the middle of the spaces left by the previous numbers.

It follows that the first 2^k Sobol scalars on the first axis ($2^k - 1$ excluding the first and invalid scalar $x_0 = 0$) evenly span the axis $(0, 1)$:

$$\{x_j^0, 1 \leq j < 2^k\} = \left\{ \frac{j}{2^k}, 1 \leq j < 2^k \right\}$$

and we begin to build an intuition for the performance of the sequence: the $2^k - 1$ first "random" numbers are even quantiles of the uniform distribution. When transformed into a Gaussian or another distribution, these numbers sample even quantiles on the target distribution.

Latin hypercube

This property, where the $2^k - 1$ first numbers are evenly spaced by $1/2^k$ on the axis $(0, 1)$, holds for all the axes. *Sobol samples the same numbers on all the axes, but in a different order.*

The direction numbers DN_k^d on all the axes have the following bit-wise property:

	kth direction number of *any* axis							
space	31	30	29	...	**31-k**	..	1	0
bit	**1**	0	**1**	0	**1**	0	0	0

The kth bit is 1

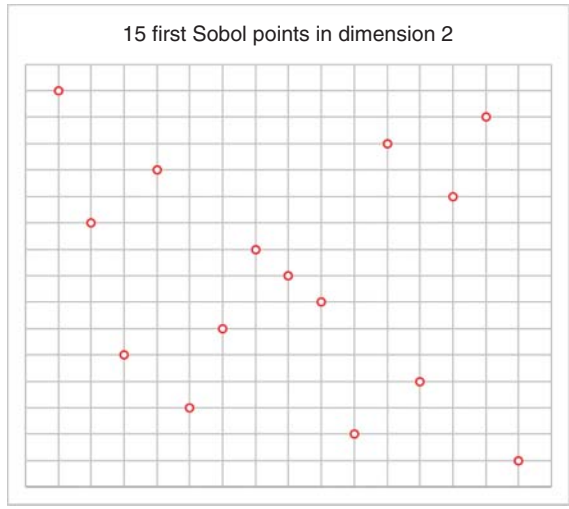
Bits on the left are 0 or 1 depending on the axis d

All the right bits are 0

The kth bit is still one, so 1/2 still flicks in and out every two numbers, 1/4 every four numbers, 1/8 every eight numbers and so forth. The bits on the right of k are still 0, so 1/16 will not kick in before all the combinations of 1/2, 1/4, and 1/8 are exhausted. It follows that the $2^k - 1$ first numbers are still the quantiles $\{j/2^k, 1 \leq j < 2^k\}$.

But the bits on the left of k are no longer all zero. Some are zero, some are one, depending on the axis, and, crucially, they are different on different axes. When the direction number DN_k^d flicks in or out every 2^{k+1} points in the sequence, it doesn't only flick $1/2^{k+1}$. Its bits on the left of k also flick some $1/2^j$ for $j \leq k$, shuffling their order of flickering.

It follows that the same numbers are sampled on all the axes, but in a different order. In addition, these numbers are even quantiles. The chart below shows the first 15 Sobol points in dimension 2, where it is visible that the points sample the 1/16 quantiles on both axes, but in a different order.



This property where N points x_i sample the hypercube $(0,1)^D$ in such a way that for every coordinate $d \in [0, D-1]$:

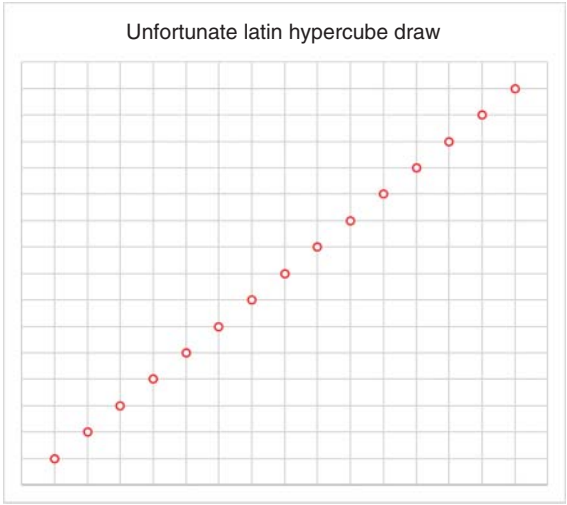
$$\{x_i^d, 0 \le i \le N-1\} = \left\{ \frac{i+1}{N+1}, 0 \le i \le N-1 \right\}$$

is not unique to Sobol and well known under the name *latin hypercube*. Latin hypercube samples the coordinates in a balanced manner. It is a form of control variate, since the N points correctly sample the $1/(N+1)$th quantiles of the marginal distributions by construction.

Latin hypercube may be implemented directly with a sampling strategy known as *balanced sampling*: to produce N points in the hypercube, sample the first axis with the sequence $x_i^0 = i/(N+1)$ for $1 \le i \le N$ and all the other axes by random permutations of the same values. Balanced sampling improves the convergence of Monte-Carlo simulations over random sampling, sometimes noticeably, and its construction is extremely efficient. Instead of randomly shuffling uniform samples before applying the Gaussian transformation, we can randomly shuffle the Gaussian quantiles $G_i^0 = N^{-1}(i/(N+1))$, so that the rather expensive Gaussian transformation is only applied on the first axis N times instead of the usual ND. Balanced sampling may be further optimized, both in speed and quality, in combination with antithetic sampling.

One catch is that balanced sampling is not an *incremental* sampling strategy. We cannot sample N points and then P additional points. All the points are generated together, coordinate by coordinate and not point by point. To apply balance sampling in a path-wise simulation, the N random vectors must be stored in memory and picked one by one, imposing a heavy load on RAM and cache. Sobol, on the contrary, is an incremental variation of the latin hypercube, whereby the points are delivered in a sequence, one D-dimensional point at a time.

Latin hypercube evenly samples the *marginal* distributions, but offers no guarantee for the *joint* distributions. An unfortunate draw may very well sample two coordinates in a similar order, as illustrated below with 15 points in dimension 2:



While the marginal distributions are still sampled evenly, it is clear that the empirical *joint* distribution is wrong: the two random numbers, supposedly independent, are 100% dependent in the sample. Seen from the discrepancy perspective, it is visibly poor, the points being clustered around the diagonal, leaving the rest of the space empty. Because the marginal distributions are correctly sampled, a simulation produced with the sample still correctly values all sets of European cash-flows, but the values of exotics would be heavily biased due to the poor sampling of the copula (see Section 4.3).

It therefore appears that, while the latin hypercube property of random numbers is a highly desirable one in the context of Monte-Carlo simulations, it is not in itself sufficient to guarantee a correct representation of joint distributions or a low discrepancy. Additional mechanisms should be in place so that the different axes are sampled in a dissimilar and independent *order*. Sobol's sequence achieves such "order independence" with the definition of its direction numbers.

Direction numbers on all axes

We did not, in our short presentation, specify the direction numbers applied in Sobol's sequence. We did define the 32 direction numbers of the first axis, and introduce a general property of all direction numbers, whereby the kth bit of the kth direction number is always one and the bits on the right of k are always zero. This is what guarantees the latin hypercube property. But the sampling order depends on the bits on the left of k. These bits are zero on the first axis, and specified in a different manner on all other axes, the specification on a given axis determining the sampling order on that axis.

The quality of the sequence, its ability to sample independent uniform numbers on each axis, resulting in a low discrepancy over the hypercube, therefore depends on the specification of the (left bits of the) direction numbers. Sobol [71] delivered a recursive mechanism for the construction of the direction numbers, but it so happens that the starting values for the recursion, called *initializers*, massively affect the quality of the sequence in high dimension. Jaeckel [63] and Joe and Kuo [72], [73] found sets of direction numbers that result in a high-quality sequence in high dimension. Without such sets of high-quality direction numbers, Sobol's sequence is practically unusable in finance, the resulting Monte-Carlo estimates being heavily biased in high dimension. It is only after the construction of direction numbers was resolved that Sobol became best practice in finance.

The construction of the direction numbers is out of our scope. We refer the interested readers to chapter 8 of [63]. Joe and Kuo's dedicated page, http://web.maths.unsw.edu.au/~fkuo/sobol/, collects many resources, including papers, synthetic notes, various sets of direction numbers in dimension up to 21,201, and demonstration code in C++ for the construction of the direction numbers and the points in Sobol's sequence. Our implementation of the next chapter uses their direction numbers in dimension up to 1,111 derived in their 2003 paper [72].