Math 525: Lecture 25

April 17, 2018

Last class, we saw that the identity

$$\int_{[0,1]^D} f \equiv \int_0^1 \cdots \int_0^1 f(x_1, \ldots, x_D) dx_1 \cdots dx_D = \mathbb{E}\left[f(X_1, \ldots, X_D)\right]$$

where $X_1, \ldots, X_D \sim U[0,1]$ independent and f is Borel measurable

yields a method (called the *Monte Carlo method*) for computing integrals: sample the function N times to obtain observations $Y_1, \ldots, Y_N \sim f(X_1, \ldots, X_D)$. Then, compute

$$\frac{S_N}{N} = \frac{Y_1 + \cdots + Y_N}{N}$$

which, by the law of large numbers, approximates $\mathbb{E}Y$.

1 Computational effort of Monte Carlo

Instead of Monte Carlo, consider approximating the integral using Riemann sums:

$$\int_0^1 \cdots \int_0^1 f(x_1, \ldots, x_D) dx_1 \cdots dx_D \approx \sum_{1 \le i_1, \ldots, i_D \le N} f(i_1 \Delta x, \ldots, i_D \Delta x)(\Delta x)^D \text{ where } \Delta x = N^{-1}.$$

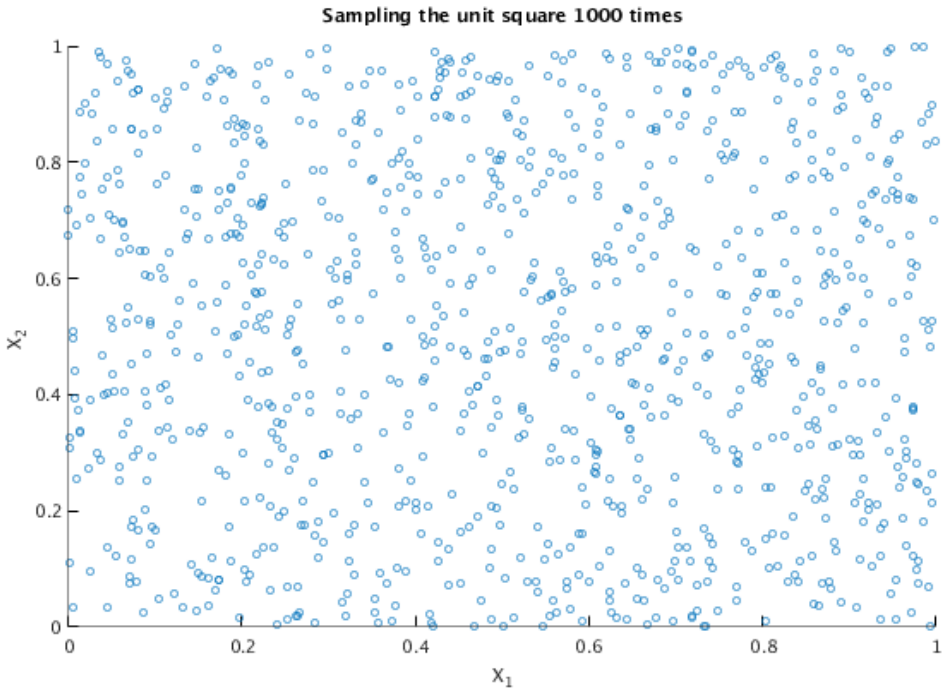
Note, in particular, that the function f has to be evaluated N^D times. This is referred to as the *curse of dimensionality*: for a given accuracy, we must increase the number of function evaluations exponentially as the dimension increases. The same is true of other "quadrature methods" (e.g., trapezoid rule, Simpson's rule, etc.), it does not apply only to Riemann sums.

The appeal of Monte Carlo is that unlike quadrature rules, it requires only N function evaluations to obtain $O(1/\sqrt{N})$ error (by the CLT). Therefore, while Monte Carlo might be ill-suited for computing low-dimensional integrals (e.g., $D = 1$ or $D = 2$), it is well-suited to computing high-dimensional ones.

2 Quasi-Monte Carlo

Discrepancy

Consider the two-dimensional ($D = 2$) case. Sampling $Z \sim U([0,1] \times [0,1])$ using a uniform distribution $N = 1000$ times, we obtain the following scatter plot:



In particular, there are regions of high density (lots of points) and regions of low density (few points). In hopes of obtaining a better error than $O(1/\sqrt{N})$, quasi-Monte Carlo uses instead a deterministic sequence $(z^n)_n$ in $[0,1]^D$. This sequence is chosen to have "good density properties". Then, the integral $\int_{[0,1]^D} f$ is approximated by

$$\frac{1}{N}\sum_{n=1}^{N} f(z^n) \equiv \frac{1}{N}\sum_{n=1}^{N} f(z_1^n, \ldots, z_D^n).$$

What does it mean for a sequence to have "good density properties"? One way to characterize this is through by discrepancy:

Definition 2.1. The *discrepancy* of a nonempty set $P = \{z_1, \ldots, z_N\} \subset [0,1]^D$ is

$$d(P) = \sup_{\substack{0 \leqslant a_i < b_i \leqslant 1 \\ i=1,\ldots,D}} \left| \prod_{i=1}^{D} (b_i - a_i) - \frac{\left| P \cap \prod_{i=1}^{D} [a_i, b_i) \right|}{N} \right|.$$

This definition looks scarier than it is. Let's go over it carefully. For some choice of endpoints $0 \leq a_i < b_i \leq 1$ $(i = 1, \ldots D)$, let $A = \prod_{i=1}^{D} [a_i, b_i)$. Note that A is just a "hyper-rectangle" in the hyper-cube $[0,1]^D$. Therefore, the product $\prod_{i=1}^{D}(b_i - a_i)$ is nothing other than the volume of A. Moreover, the quantity $|P \cap A|$ is the number of points that are in A. Therefore, the discrepancy corresponds to picking a hyper-rectangle that has the "worst density" with respect to the points $P = \{z_1, \ldots, z_n\}$.

Definition 2.2. The *star discrepancy* of a nonempty set $P = \{z_1, \ldots, z_N\} \subset [0,1]^D$ is

$$d^*(P) = \sup_{\substack{0 < u_i \leqslant 1 \\ i=1,\ldots,D}} \left| \prod_{i=1}^{D} u_i - \frac{\left| P \cap \prod_{i=1}^{D} [0, u_i) \right|}{N} \right|.$$

The star discrepancy is almost identical to the discrepancy: all that differs is that we fix our attention to hyper-rectangles of the form $\prod_{i=1}^{D}[0, u_i)$.

Proposition 2.3. *For any nonempty finite set $P \subset [0,1]^D$,*

$$d^*(P) \leq d(P) \leq 2^D d^*(P).$$

Proof. The leftmost inequality is trivial. As for the rightmost inequality, let's first consider the one-dimensional case ($D = 1$). Note that

$$[a, b) = [0, b) \setminus [0, a).$$

Therefore,

$$|P \cap [a, b)| = |P \cap ([0, b) \setminus [0, a))| = |P \cap [0, b)| - |P \cap [0, a)|.$$

Putting this all together,

$$\begin{aligned} d(P) &= \sup_{0 \leq a < b \leq 1} \left| (b - a) - \frac{|P \cap [a, b)|}{N} \right| = \sup_{0 \leq a < b \leq 1} \left| \left(b - \frac{|P \cap [0, b)|}{N} \right) - \left(a - \frac{|P \cap [0, a)|}{N} \right) \right| \\ &\leq \sup_{0 < b \leq 1} \left| b - \frac{|P \cap [0, b)|}{N} \right| + \sup_{0 \leq a < 1} \left| a - \frac{|P \cap [0, a)|}{N} \right| = d^*(P) + d^*(P) = 2d^*(P). \end{aligned}$$

The same idea extends to higher dimensions ($D > 1$). □

Koksma–Hlawka inequality

Proposition 2.4 (Koksma–Hlawka)**.** *Let $P = \{z_1, \ldots, z_N\}$ be a nonempty subset of $[0,1]^D$. Then,*

$$\left| \frac{1}{N} \sum_{n=1}^{N} f(z^n) - \int_{[0,1]^D} f \right| \leq V_{HK}(f) d^*(P)$$

where $V_{HK}(f)$ is the Hardy–Krause variation of f.

The proof of this result (and even the definition of $V_{HK}(\cdot)$) is far out of the scope of this class. However, we can interpret this result as telling us that for any "well-behaved" function f, the error in quasi-Monte Carlo using points $P = \{z_1, \ldots, z_N\}$ is at most

$$C_f d^*(P)$$

where C_f is some positive constant that depends only on f.

In other words, the quality of quasi-Monte Carlo depends only on the (star) discrepancy of the sequence we choose. In particular, we would like to pick the sequence to have the lowest (star) discrepancy possible. We should mention at this point that it is not known what the lowest possible (star) discrepancy is:

Conjecture 2.5. *There is a constant C_D depending only on the dimension D such that*

$$d^*(P) \geq C_D \frac{(\ln N)^{D-1}}{N}$$

for any nonempty set $P = \{z_1, \ldots, z_N\} \subset [0,1]^D$.

The conjecture has been verified for $D = 1$ ($C_1 = 1/2$) and $D = 2$ ($C_2 = \frac{1}{8} e^{W_{-1}(-\frac{1}{2e})+1} = 0.0233352886063546\ldots$) but remains open for $D \geq 3$.

Van der Corput (and Halton) sequences

The *Van der Corput sequence* is a simple example of a *low discrepancy sequence*. It is a sequence in one dimension ($D = 1$), constructed as follows. Fix a base b. For a positive integer n, let $n = (\cdots n_1 n_0)_b$ denote its base b expansion (i.e., $0 \leq n_j < b$ is the j-th digit of n in base b). Then,

$$n = n_0 + n_1 b + n_2 b^2 + \cdots$$

The n-th element of the Van der Corput sequence is then given by

$$\varphi_b(n) = n_0 b^{-1} + n_1 b^{-2} + n_2 b^{-3} + \cdots$$

In other words, the Van der Corput sequence "reverses" the base b expansion of the natural numbers.

Example 2.6. In $b = 2$, note that $0 = (0)_2$, $1 = (1)_2$, $2 = (10)_2$, $3 = (11)_2$, $4 = (100)_2$, etc. Therefore, the Van der Corput sequence is

$$\{(0)_2, (0.1)_2, (0.01)_2, (0.11)_2, (0.001)_2, \ldots\}.$$

Converting these numbers back to base 10,

$$\{0, 0.5, 0.25, 0.75, 0.125, \ldots\}.$$

Remark 2.7. The Halton sequences generalize the Van der Corput sequences to $D > 1$.

The following is a slight generalization of Van der Corput's original result (1935) due to S. Harber (1966):

Proposition 2.8. *Let $P = \{\varphi_2(0), \ldots, \varphi_b(N-1)\}$ be the first $N > 1$ elements of the Van der Corput sequence in base $b = 2$. Then,*

$$Nd^*(P) \leq \frac{1}{3} \log_2 N + C$$

for some $C > 0$.

Sobol sequences

Probably the most popular low discrepancy sequence is the Sobol sequence. I mention it here only so that you are aware of it by name. Figure 1 shows the first 1000 points of the Sobol sequence (compare with the 1000 uniform samples at the beginning of this lecture).

Algorithm 1 gives some C++ code to compute π with a two-dimensional Sobol sequence $(z^n)_{n \geq 1}$. Setting N to be 10 million, we obtain $\pi \approx 3.1415952$ (much more accurate than the plain Monte Carlo approach we saw last class).

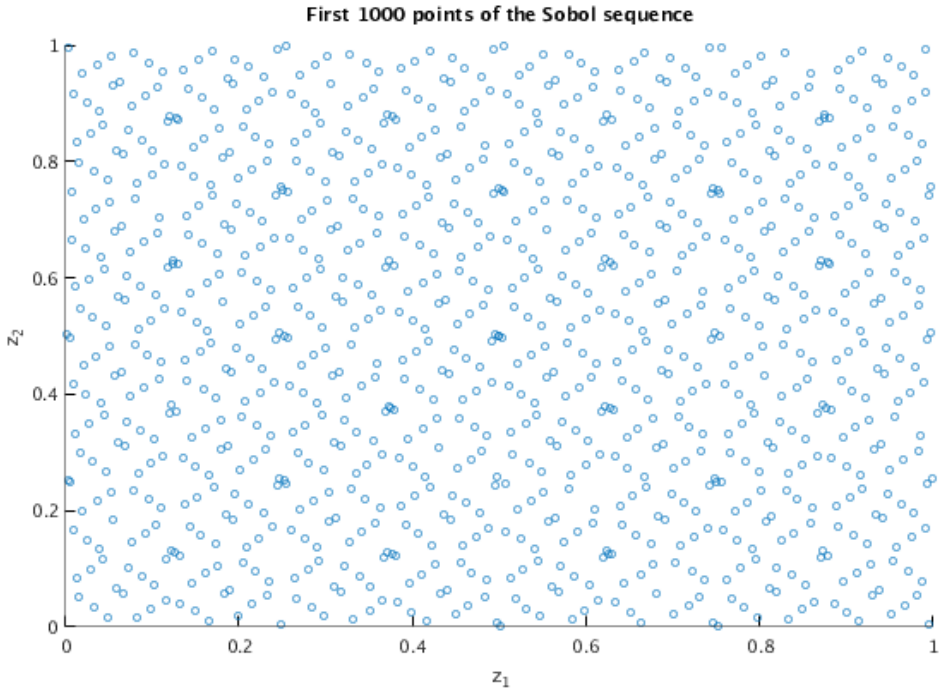


Figure 1: 2d Sobol sequence

Algorithm 1 quasi-Monte Carlo method to approximate π

```
#include <iomanip>
#include <iostream>

// http://people.sc.fsu.edu/~jburkardt/cpp_src/sobol/sobol.html
#include "sobol.hpp"

int main()
{
        const int N          = 10000000;
        long long int seed   = 1;
        long long int count  = 0;

        double r[2];
        for(int n = 0; n < N; ++n)
        {
                i8_sobol(2, &seed, r);
                const bool inside_circle = r[0] * r[0] + r[1] * r[1] <= 1.;
                if(inside_circle) { ++count; }
        }

        const double pi_approximate = 4. * ((double) count) / N;
        std::cout << std::setprecision(16) << pi_approximate << std::endl;

        return 0;
}
```