Methods of Monte Carlo Simulation

Ulm University
Institute of Stochastics

Lecture Notes
Dr. Tim Brereton
Winter Term 2013/2014

Ulm, February 2014

Contents

Chapter 1

Introduction

Monte Carlo methods are methods that use random numbers to solve problems or gain insight into problems. These problems can be 'probabilistic' in nature or 'deterministic'. Probabilistic applications of Monte Carlo methods include:

- Estimating probabilities and expectations.
- Estimating the sensitivity of random objects to changes in parameters.
- Getting a sense of what random objects 'look like' and how they behave.

Deterministic problems which can be solved using Monte Carlo methods are, for example:

- Estimating solutions to difficult integration problems.
- Approximating or finding solutions to complicated optimization problems.
- Solving mathematical problems by transforming them into 'probabilistic' problems (an example is probabilistic methods for solving partial differential equations).

Monte Carlo techniques are not always the best tools, especially for simple problems. However, they are the best (or only) solutions for a lot of realistic problems.

1.1 Monte Carlo Integration

A simple problem that can solved using Monte Carlo methods is to compute an integral of the form

$$I = \int_0^1 f(x)\,\mathrm{d}x,$$

where f is an arbitrary function. This can be written as

$$I = \int_0^1 f(x)dx = \int_0^1 \frac{1}{1} f(x)\,\mathrm{d}x.$$

Note that $1/1$ is the probability density function (pdf) of the uniform distribution on $(0,1)$. So, we can write

$$I = \int_0^1 f(x)\,\mathrm{d}x = \mathbb{E}\, f(X),$$

where $X \sim \mathcal{U}(0,1)$. Now we can approximate $\mathbb{E}\, f(X)$ by

$$I \approx \hat{I}_N = \frac{1}{N}\sum_{i=1}^{N} f(X_i),$$

where $X_1, X_2, \ldots, X_N$ are independent and identically distributed (iid) copies of X. Then, under suitable technical conditions (e.g. $\mathbb{E}\, f(x)^2 < \infty$), the strong law of large numbers implies that

$$\lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} f(X) \to \mathbb{E}\, f(X_i) \qquad \text{almost surely (a.s.) and in } L^2$$

We can easily establish some important properties of the estimator $\hat{I}_N$.

1.1.1 Expectation, Variance and Central Limit Theorem

We have

$$\mathbb{E}\,\hat{I} = \mathbb{E}\left(\frac{1}{N}\sum_{i=1}^{N} f(X_i)\right) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}\, f(X_i) = \frac{N}{N}\mathbb{E}\, f(x) = I.$$

Therefore, $\hat{I}_N$ is unbiased. Likewise, we have

$$\mathrm{Var}(\hat{I}_N) = \mathrm{Var}\left(\frac{1}{N}\sum_{i=1}^{N} f(X_i)\right) = \frac{1}{N^2}\sum_{i=1}^{N} \mathrm{Var}(f(X_i)) = \frac{1}{N}\mathrm{Var}(f(X)).$$

So the standard deviation is

$$\mathrm{Std}(\hat{I}_N) = \frac{\sqrt{\mathrm{Var}(f(X))}}{\sqrt{N}}.$$

Under suitable technical conditions (e.g. $\mathrm{Var}(f(X)) < \infty$), a central limit theorem holds and we additionally get that

$$\left(\hat{I}_N - I\right) \xrightarrow{D} \mathcal{N}\left(0, \frac{\mathrm{Var}(f(X))}{N}\right) \qquad \text{as } N \to \infty.$$

Example

$$\text{Estimate} \qquad \int_0^1 e^{-x^2}\,\mathrm{d}x$$

Listing 1.1: Matlab Code

```
N = 10^5;
X = rand(N,1);
f_X = exp(-X.^2);
I_hat = mean(f_X)
```

1.1.2 Higher Dimensional Integration Problems

Monte Carlo integration is especially useful for solving higher dimensional integration problems. For example, we can estimate integrals of the form

$$I = \int_0^1 \int_0^1 \cdots \int_0^1 f(x_1, \ldots, x_n)\, \mathrm{d}x_1\, \mathrm{d}x_2 \cdots \mathrm{d}x_n.$$

Similarly to the one-dimensional case, observe that $I = \mathbb{E}\, f(\mathbf{X})$, where $\mathbf{X}$ is a *vector* of iid $\mathcal{U}(0,1)$ random variables. This expected value can be approximated by

$$\hat{I}_N = \frac{1}{N} \sum_{i=1}^{N} f\left(\mathbf{X}_i\right),$$

where the $\{\mathbf{X}_i\}_{i=1}^N$ are iid n-dimensional vectors of iid $\mathcal{U}(0,1)$ random variables.

Example

$$\text{Estimate} \qquad \int_0^1 \int_0^1 e^{x_1} \cos(x_2)\, \mathrm{d}x_1\, \mathrm{d}x_2.$$

Listing 1.2: Matlab Code

```
N = 10^6;
f_X = zeros(N,1);
for i = 1:N
    X = rand(1,2);
    f_X(i) = exp(X(1))*cos(X(2));
end
I_hat = mean(f_X)
```

1.2 Further Reading

Very good reviews of Monte Carlo methods can be found in [1, 2, 4, 6, 9, 10, 11]. If you are interested in mathematical tools for studying Monte Carlo methods, a good book is [5].

Chapter 2

Pseudo Random Numbers

The important ingredient in everything discussed so far is the ability to generate iid $\mathcal{U}(0,1)$ random variables. In fact, almost everything we do in Monte Carlo begins with the assumption that we can generate $\mathcal{U}(0,1)$ random variables. So, how do we generate them?

Computers are not inherently random, so we have two choices:

1. Using a physical device that is 'truly random' (e.g. radioactive decay, coin flipping).
2. Using a sequence of numbers that are not truly random but have properties which make them seem/act like 'random numbers'.

Possible problems with the physical approach are:

1. The phenomenon may not be 'truly random' (e.g., coin flipping may involve bias).
2. Measurement errors.
3. These methods are slow.
4. By their nature, these methods are not reproducible.

A problem with the deterministic approach is, obviously, the numbers are not truly random.

The choice of a suitable random number generation method depends on the application. For example, the required properties of a random number generator used to generate pseudo random numbers for Monte Carlo methods are very different from those of a random number generator used to create pseudo random numbers for gambling or cryptography.

2.1 Requirements for Monte Carlo

In Monte Carlo applications there are many properties we might require of a random number generator (RNGs). The most import are:

1. The random numbers it produces should be uniformly distributed.
2. The random numbers it produces should be independent (or at least they should seem to be independent).
3. It should be fast.
4. It should have a small memory requirement.
5. The random numbers it produces should have a large period. This means that, if we use a deterministic sequence that will repeat, it should take a long time before it starts repeating.

Ideally, the numbers should be **reproducible** and the algorithm to generate them should be **portable** and should **not produce 0 or 1**.

2.2 Pseudo Random Numbers

2.2.1 Abstract Setting

S finite set of states,

f transition function $f : S \to S$,

S_0 a seed,

U output space,

g output function $g : S \to U$.

Algorithm 2.2.1 (General Algorithm)

1. Initialize: Set $X_1 = S_0$. Set $t = 2$.
2. Transition: Set $X_t = f(X_{t-1})$.
3. Output: Set $U_t = g(X_t)$.
4. Set $t = t + 1$. Repeat from 2.

2.2.2 Linear Congruential Generators

The simplest useful pseudo random number generator is a Linear Congruential Generator (LCG).

Algorithm 2.2.2 (Basic LCG)

1. Initialize: Set $X_1 = S_0$. Set $t = 2$.
2. Transition: Set $X_t = f(X_{t-1}) = (aX_{t-1} + c) \mod m$.

3. Output: Set $U_t = g(X_t) = \frac{X_t}{m}$.

4. Set $t = t + 1$ and repeat from step 2.

We call a the **multiplier** and c the **increment**.

Example

Take $a = 6$, $m = 11$, $c = 0$ and $S_0 = 1$.

Then
$$\begin{aligned} X_1 &= 1 & U_1 &= 1/11 \\ X_2 &= (6 \cdot 1) \mod 11 = 6 & U_2 &= 6/11 \\ X_3 &= (6 \cdot 6) \mod 11 = 3 & U_3 &= 3/11 \\ X_4 &= (6 \cdot 3) \mod 11 = 7 & U_4 &= 7/11 \end{aligned}$$

Sequence: $\underbrace{1, 6, 3, 7, 9, 10, 5, 8, 4, 2}_{\text{period } = 10 = (m-1)}, 1, 6, \ldots$

Listing 2.1: Matlab Code

```
N = 10;
a = 6; m = 11; c = 0;
S_0 = 1;
X = zeros(N,1); U = zeros(N,1);
X(1) = S_0;
for i = 2:N
    X(i) = mod(a*X(i-1)+c,m);
    U(i) = X(i)/m;
end
```

What happens if we take $m = 11$, $a = 3$, $c = 0$, $S_0 = 1$?

$$\begin{aligned} X_1 &= 1 \\ X_2 &= (3 \cdot 1) \mod 11 = 3 \\ X_3 &= (3 \cdot 3) \mod 11 = 9 \\ X_4 &= (3 \cdot 9) \mod 11 = 5 \\ X_5 &= (3 \cdot 5) \mod 11 = 4 \\ X_6 &= (3 \cdot 4) \mod 11 = 1 \end{aligned}$$

Sequence: $\underbrace{1, 3, 9, 5, 4}_{\text{period } = 5}, 1, \ldots$

When using an LCG, it is important to have as long a period as possible. The following theorems give conditions for the maximum possible periods to be obtained.

Theorem 2.2.1 *An LCG with $c = 0$ has full period $(m - 1)$ if*

1. $S_0 \neq 0$.

2. $a^{m-1} - 1$ *is a multiple of* m.

3. *a^{j-1} is not a multiple of m for $j = 1, \ldots, m-2$.*

Theorem 2.2.2 *An LCG with $c \neq 0$ has full period (m) if and only if*

1. *c and m are relatively prime (Their only common divisor is 1).*
2. *Every prime number that divides m divides $a-1$.*
3. *$a-1$ is divisible by 4 if m is.*

The conditions are broken for the examples above (with $c \neq 0$) because 11 divides $m = 11$ but not $a = 3$ or $a = 6$. However, we know that the Theorem 2.2.2 is satisfied if c is odd, m is a power of 2 and $a = 4n + 1$.

Many LCGs have periods of $2^{31} - 1 \approx 2.1 \times 10^9$

Example Minimal Standard LCG: $a = 7^5 = 16807$, $c = 0$, $m = 2^{31}$

In many modern Monte Carlo applications, samples sizes of $N = 10^{10}$ or bigger are necessary. A rough rule of thumb is that the period should be around N^2 or N^3. Thus, for $N \approx 10^{10}$ a period bigger than 10^{20} or even 10^{30} is required.

2.2.3 Extensions of LCGs

Multiple Recursive Generators

Algorithm 2.2.3 (Multiple Recursive Generator (MRG) of order k)

1. Initialize: Set $X_1 = S_0^1, \ldots, X_k = S_0^k$. Set $t = k + 1$.
2. Transition: $X_t = (a_1 X_{t-1} + \cdots + a_k X_{t-k}) \mod m$.
3. Output: $U_t = \frac{X_t}{m}$
4. Set $t = t + 1$. Repeat from step 2.

Usually, most of the $\{a_i\}_{i=1}^k$ are 0. Clearly, an LCG (with $c = 0$) is a special case of an MRG. Note that the state space is now $\{0, \ldots, m-1\}^k$ and the maximum period is now $m^k - 1$. It is obtained, e.g., if:

a) m is prime

b) The polynomial $p(z) = z^k - \sum_{i=1}^{k-1} a_i z^{k-i}$ is prime using modulo m arithmetic.

Obviously, $m^k - 1$ can be much bigger than m or $m - 1$.

Combined Generators

An example of a combined random number generator is the Wichmann-Hill pseudo random number generator.

Algorithm 2.2.4 (Wichmann-Hill)

1. Set $X_0 = S_0^X, Y_0 = S_0^Y, Z_0 = S_0^Z$. Set $t = 1$.
2. Set $X_t = a_1 X_{t-1} \mod m_1$
3. $Y_t = a_2 Y_{t-1} \mod m_2$
4. $Z_t = a_3 Y_{t-1} \mod m_3$
5. $U_t = \left(\frac{X_t}{m_1} + \frac{Y_t}{m_2} + \frac{Z_t}{m_3}\right) \mod 1$
6. Set $t = t + 1$ and repeat from step 2.

Pierre L'Ecuyer combines multiple recursive generators.

2.2.4 The Lattice Structure of Linear Congruential Generators

LCGs have what is known as a lattice structure.

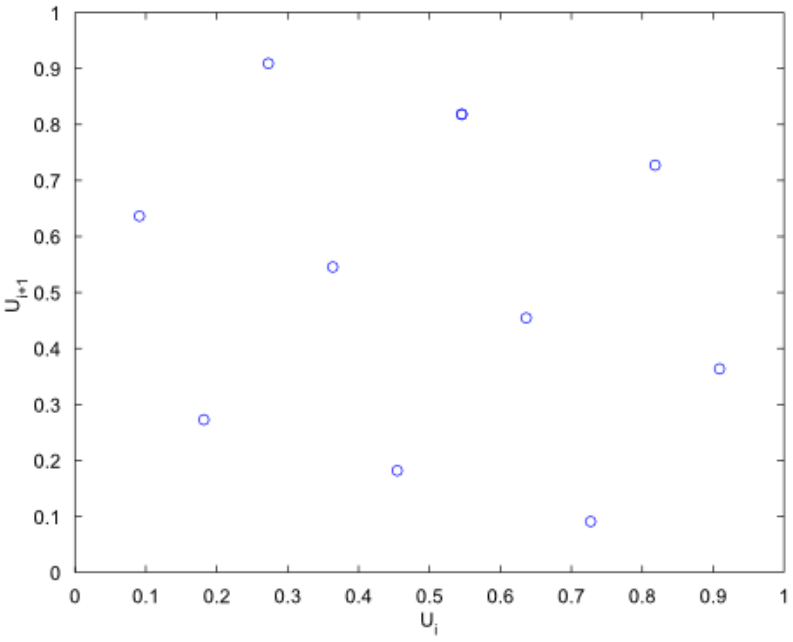


Figure 2.1: Points generated by LCG with $a = 6$, $c = 0$ and $m = 11$.

In higher dimensions, d-tuples – e.g., $(X_1, X_2), (X_2, X_3), (X_3, X_4) \ldots$ – lie on hyperplanes. It can be shown that points created by linear congruential generation with modulus m lie on at most $(d!m)^{\frac{1}{d}}$ hyperplanes in the d-dimensional unit cube. For $m = 2^{31} - 1$ and $d = 2$, they lie on at most ≈ 65536 hyperplanes. For $d = 10$ they lie on at most only ≈ 39 hyperplanes.

One way to asses the quality of a random number generator is to look at the "spectral gap" which is the maximal distance between two hyperplanes.

2.2.5 Linear Feedback Shift Register-type algorithms

Each X_i can take values in $\{0, 1\}$. We update the X_i using

$$X_i = (a_1 X_{i-1} + \cdots + a_k X_{i-k}) \mod 2.$$

and output

$$U_i = \sum_{j=1}^{L} X_{i\nu+j-1} 2^{-j},$$

where ν is the step size and L is the word length (usually 32 or 64). Using this approach, we get a period of $2^k - 1$. If we write the above in the form $\mathbf{X}_i = a_1 \mathbf{X}_{i-1} + \cdots a_k \mathbf{X}_{i-k}$, where $\mathbf{X}_i = (X_{i,1}, \ldots, X_{i,k})$, we have a vector version of this generator. Then, the output is

$$U_i = \sum_{j=1}^{L} X_{i,j} 2^{-j}.$$

The Mersenne Twister does a few additional things but is roughly of this form.

2.3 Testing Pseudo Random Number Generators

It is not enough for a random number generator to have a large period. It is even more important that the resulting numbers appear to be independent and identically distributed. Of course, we know that the numbers are not really random. However, a good random number generator should be able to fool as many statistical (and other) tests as possible into thinking that the numbers it produces are iid uniforms.

2.3.1 Testing the Sizes of the Gaps between Hyperplanes

This is the only non-statistical test we will discuss. Remember, that many random number generators have a lattice structure.

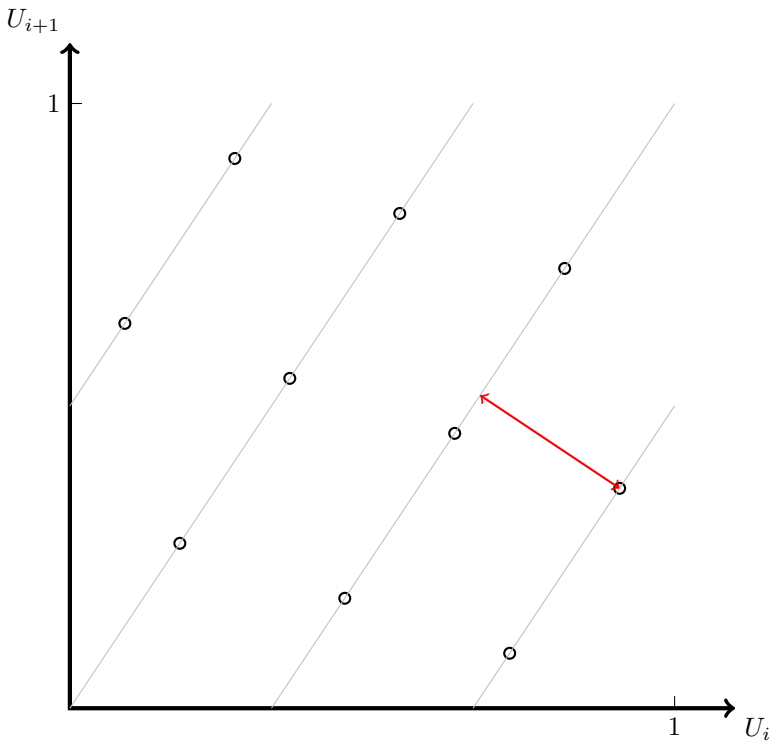


Figure 2.2: Lattice Structure of a Random Number Generator

In general, a lattice structure is bad, because it means numbers are not sufficiently independent of one another. We cannot avoid having a lattice structure, but we would like to have as many hyperplanes as possible (this means that patterns of association between random variables are less strong). 'SpectralÂ´ tests measure the gaps between hyperplanes. The mathematics involved in these tests is quite complicated.

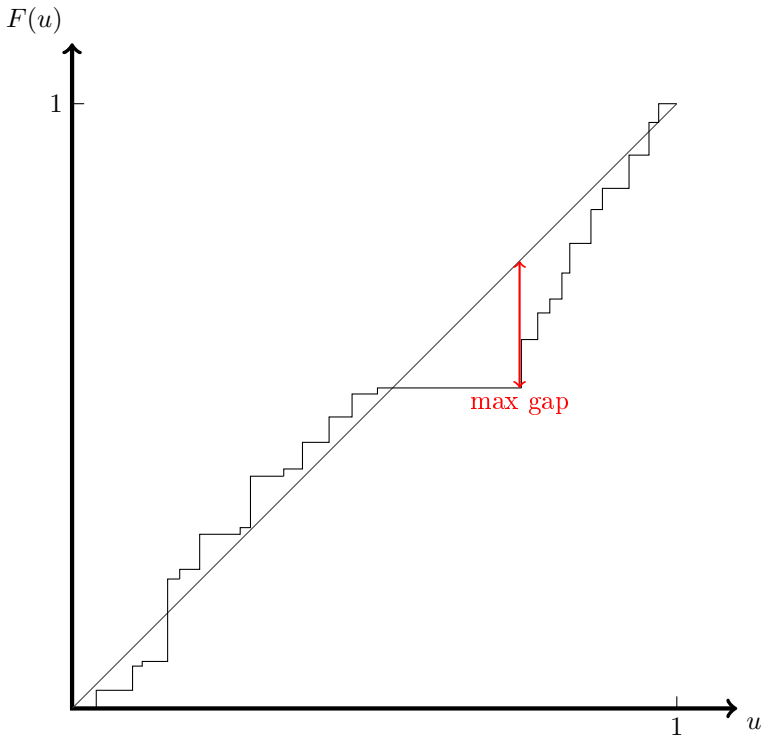
2.3.2 The Kolmogorov-Smirnov Test

This test checks if the output of a RNG is close enough to the uniform distribution. The idea of the Kolmogorov-Smirnov test is to compare the estimated cumulative density function (cdf) of the output of a random number generator against the cdf of the uniform (0,1) distribution. If the two cdfs are too different from one another, we say that the RNG does not produce uniform random variables. The test statistic is

$$D_n := \sup_{u \in (0,1)} |F_n(u) - F(u)|,$$

where F_n is an estimate of the cdf given the data (often called the empirical cdf). This statistic shouldn't be too big.

Example (The Kolmogov-Smirnov statistic)



2.3.3 Chi-Squared Test

The Chi-Squared test is another test to check that the output of a random number generator has a uniform $(0, 1)$ distribution. We can test if a given sample $\{U_n\}_{n=1}^N$ is uniform by dividing it into equally spaced intervals.

Example (An example subdivision)

$$\begin{aligned} Q_1 = &\quad \text{number of points in } \{U_n\}_{n=1}^N \quad \text{between 0 and 0.1} \\ Q_2 = &\quad \text{number of points in } \{U_n\}_{n=1}^N \quad \text{between 0.1 and 0.2} \\ &\vdots \\ Q_{10} = &\quad \text{number of points in } \{U_n\}_{n=1}^N \quad \text{between 0.9 and 1} \end{aligned}$$

If the given sample is uniformly distributed, the expected number of points in the ith interval (E_i) is the length times the number of points in the sample ($|Q_i|N$). In the example, $E_1 = E_2 = \cdots = E_{10} = \frac{N}{10}$.

The Chi-Squared Test statistic is

$$\chi^2_{\text{stat}} = \sum_{i=1}^{L} \frac{(Q_i - E_i)^2}{E_i},$$

where L is the number of segments (in the example, $L = 10$). If χ^2_{stat} is too big, the random number generator does not appear to be producing uniform random variables.

2.3.4 Permutation Test

For iid random variables, all orderings should be equally likely.

For example, if X_1, X_2, X_3 are iid R.V.s, then

$$\mathbb{P}(X_1 \leq X_2 \leq X_3) = \mathbb{P}(X_2 \leq X_1 \leq X_3) = \mathbb{P}(X_3 \leq X_2 \leq X_1) = \dots$$

Let Π_d be the indices of an ordering of d iid uniformly distributed random variables. For example, if $X_1 \leq X_2 \leq X_3$, $\Pi_3 = \{1, 2, 3\}$.

It's not hard to see that

$$\mathbb{P}(\Pi = \pi) = \frac{1}{d!}. \tag{2.1}$$

This suggests we can test a random number generator by generating N runs of d random variables and then doing a Chi-Squared test on them to check whether their ordering indices are distributed according to (2.1).

2.4 Quasi Monte Carlo

So far we have considered too much structure in a random number generator as a negative thing (that is, we have wanted things to be as random as possible). However, another approach to Monte Carlo – called Quasi-Monte Carlo (QMC) – tries to use structure / non-independence to improve the performance of Monte Carlo methods.

2.4.1 Numerical Integration and Problems in High Dimensions

Monte Carlo integration is not very efficient in low dimensions. Instead of Monte Carlo integration, one can use Newton Cotes methods. These methods evaluate the function at equally spaced points. There are also fancier numerical methods like Gaussian quadrature.

Using Newton Cotes methods like the rectangle method or the trapezoidal method, one needs

$$\begin{array}{lcl} \text{in 2 D} & n \times n & \text{points} \\ \text{in 3 D} & n^3 & \text{points} \\ & \vdots & \\ \text{in } d \text{ D} & n^d & \text{points.} \end{array}$$

An exponential growth of points is required. This is an example of the "curse of dimensionality".

For most numerical integration methods, the error of the approximated integral is roughly proportional to $n^{-c/d}$ for some constant c. As d gets larger, the error decays more slowly. In comparison, the error of Monte Carlo integration (measured by standard deviation) is proportional to $n^{-1/2}$. In high enough dimensions, this will be smaller than the error of numerical integration.

2.4.2 The Basic Idea of QMC

Quasi Monte Carlo methods generate deterministic sequences that get rates of error decay close to n^{-1} (as compared to $n^{-1/2}$ for standard Monte Carlo methods). They perform best in reasonably low dimensions (say about 5 to 50). One disadvantage is that they need a fixed dimension (some Monte Carlo methods do not work in a fixed dimension). This means it is not always possible to use them.

A grid is a bad way to evaluate high dimensional integrals. The idea of QMC is that the points should be spread out more efficiently. A good spread means here a low 'discrepancy'.

Definition 2.4.1
Given a collection $\mathcal{A}$ of (Lebesgue measurable) subsets of $[0,1)^d$, the discrepancy relative to $\mathcal{A}$ is

$$D(x_1,\ldots,x_n;\mathcal{A}) = \sup_{A\in\mathcal{A}} \left| \frac{\#\{x_i \in A\}}{n} - \mathrm{vol}(A) \right|.$$

Basically, discrepancy measures the difference between the number of points that should be in each of the sets if the points are evenly spaced, and the number of points that are actually in these sets.

Example 'Ordinary discrepancyÂ´', is based on sets of the form

$$\prod_{j=1}^{d} [u_j, c_j) \qquad 0 \le u_j < v_j \le 1.$$

2.4.3 Van der Corput Sequences

A number of QMC methods are based on the Van der Corput sequences (which have low discrepancy). The idea is to write numbers $1, 2, \ldots$ in base b and then 'flip them' and reflect them over decimal points.

Example The van der Corput sequence with $b = 2$

$$\begin{aligned} 1 = 001.0 \quad &\Rightarrow 0.100 && \left(= \frac{1}{2}\right) \\ 2 = 010.0 \quad &\Rightarrow 0.010 && \left(= \frac{1}{4}\right) \\ 3 = 011.0 \quad &\Rightarrow 0.110 && \left(= \frac{3}{4}\right) \\ 4 = 100.0 \quad &\Rightarrow 0.001 && \left(= \frac{1}{8}\right) \\ &\vdots \end{aligned}$$

Example The van der Corput sequence with $b = 3$

$$\begin{aligned} 1 = 001.0 \quad &\Rightarrow 0.100 && \left(= \frac{1}{3}\right) \\ 2 = 002.0 \quad &\Rightarrow 0.020 && \left(= \frac{2}{3}\right) \\ 3 = 010.0 \quad &\Rightarrow 0.010 && \left(= \frac{1}{9}\right) \\ 4 = 011.0 \quad &\Rightarrow 0.110 && \left(= \frac{4}{9}\right) \end{aligned}$$

2.4.4 Halton Sequences

Probably the simplest QMC method is called the Halton sequence. It fills a d dimension cube with points whose coordinates follow Van Der Corput sequences. For example, we can sample points $(x_1, y_1), (x_2, y_2), \ldots$, where the x coordinates follow a Van der Corput sequence with base b_1 and the y coordinates follow a Van der Corput sequence with base b_2. The b_i are chosen to be relatively prime, which means that they have no common divisors.

2.5 Furter Reading

Important books on random number generation and QMC include [3, 7, 8]

Bibliography

[1] S. Asmussen and P. W. Glynn. *Stochastic Simulation: Algorithms and Analysis*. Springer-Verlag, New York, 2007.

[2] G. S. Fishman. *Monte Carlo: Concepts, Algorithms and Applications*. Springer-Verlag, New York, 1996.

[3] J. E. Gentle. *Random Number Generation and Monte Carlo Methods*. Springer-Verlag, New York, second edition, 2003.

[4] P. Glasserman. *Monte Carlo Methods in Financial Engineering*. Springer-Verlag, New York, 2004.

[5] C. Graham and D. Talay. *Stochastic Simulation and Monte Carlo Methods: Mathematical Foundations of Stochastic Simulation*. Springer, Heidelberg, 2013.

[6] D. P. Kroese, T. Taimre, and Z. I. Botev. *Handbook of Monte Carlo Methods*. John Wiley & Sons, New York, 2011.

[7] C. Lemieux. *Monte Carlo and Quasi-Monte Carlo Sampling*. Springer, New York, 2009.

[8] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. SIAM, Philadelphia, 1992.

[9] C. P. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer-Verlag, New York, second edition, 2004.

[10] S. Ross. *Simulation*. Academic Press, San Diego, fifth edition, 2013.

[11] R. Y. Rubinstein and D. P. Kroese. *Simulation and the Monte Carlo Method*. John Wiley & Sons, New York, second edition, 2007.