# Monte Carlo integration

Monte Carlo integration is a quadrature (cubature) where the nodes are chosen randomly. Typically no assumptions are made about the smoothness of the integrand, not even that it is continuous.

Plain Monte Carlo algorithm distributes points uniformly throughout the integration region using either uncorrelated pseudo-random or correlated quasi-random sequences of points.

Adaptive algorithms, such as VEGAS and MISER, distribute points non-uniformly in an attempt to reduce integration error. They use correspondingly *importance* and *stratified* sampling.

## Multi-dimensional integration

One of the problems in multi-dimensional integration is that the integration region $\Omega$ is often quite complicated, with its boundary not easily described by simple functions. However, it is usually much easier to find out whether a given point lies within the integration region or not. Therefore a popular strategy is to create an auxiliary rectangular volume $V$ which contains the integration volume $\Omega$ and an auxiliary function $F$ which coincides with the integrand inside the volume $\Omega$ and is equal zero outside. Then the integral of the auxiliary function over the (simple rectangular) auxiliary volume is equal the original integral.

Unfortunately, the auxiliary function is generally non-continuous at the boundary and thus the ordinary quadratures which assume continuous integrand will fail badly here while the Monte-Carlo quadratures will do just as good (or as bad) as with continuous integrand.

## Plain Monte Carlo sampling

Plain Monte Carlo is a quadrature with random abscissas and equal weights ,

$$\int_V f(\mathbf{x})dV \approx w \sum_{i=1}^{N} f(\mathbf{x}_i) \ , \tag{1}$$

where $\mathbf{x}$ a point in the multi-dimensional integration space. One free parameter, $w$, allows one condition to be satisfied: the quadrature has to integrate exactly a constant function. This gives $w = V/N$,

$$\int_V f(\mathbf{x})dV \approx \frac{V}{N} \sum_{i=1}^{N} f(\mathbf{x}_i) = V\langle f \rangle \ . \tag{2}$$

According to the *central limit theorem* the error estimate $\epsilon$ is close to

$$\epsilon = V\frac{\sigma}{\sqrt{N}} \ , \tag{3}$$

where $\sigma$ is the variance of the sample,

$$\sigma^2 = \langle f^2 \rangle - \langle f \rangle^2. \tag{4}$$

The $1/\sqrt{N}$ convergence of the error, typical for a random process, is quite slow.

## Importance sampling

Suppose that the points are distributed not uniformly but with some density $\rho(x)$ : the number of points $\Delta n$ in the volume $\Delta V$ around point $x$ is given as

$$\Delta n = \frac{N}{V}\rho\Delta V, \tag{5}$$

where $\rho$ is normalised such that $\int_V \rho dV = V$.

The estimate of the integral is then given as

$$\int_V f(\mathbf{x})dV \approx \sum_{i=1}^{N} f(\mathbf{x}_i)\Delta V_i = \sum_{i=1}^{N} f(\mathbf{x}_i)\frac{V}{N\rho(\mathbf{x}_i)} = V\left\langle \frac{f}{\rho} \right\rangle \ , \tag{6}$$

Table 1: Plain Monte Carlo integrator

```
#include <stdlib.h>
#include <cmath>
#define RND ((double)rand()/RAND_MAX)

void randomx(int d, double *a, double *b, double *x)
{
for(int i=0;i<d;i++) x[i]=a[i]+RND*(b[i]-a[i]);
}

double volume(int d, double *a, double *b)
{
double V=1; for(int i=0;i<d;i++) V*=b[i]-a[i];
return V;
}

void plainmc
(int d, double f(double*), double *a, double *b, int N, double *Q, double*err)
{
double V = volume(d,a,b);
double sum=0, sum2=0, x[d];
for(int i=0;i<N;i++)
        {
        randomx(d,a,b,x);
        double fx=f(x);
        sum+=fx; sum2+=fx*fx;
        }
double avr = sum/N, var = sum2/N-avr*avr;
*Q = avr*V; *err = sqrt(var/N)*V;
}
```

where

$$\Delta V_i = \frac{V}{N\rho(x_i)} \tag{7}$$

is the "volume per point" at the point $x_i$.

The corresponding variance is now given by

$$\sigma^2 = \left\langle \left(\frac{f}{\rho}\right)^2 \right\rangle - \left\langle \frac{f}{\rho} \right\rangle^2 . \tag{8}$$

Apparently if the ratio $f/\rho$ is close to a constant, the variance is reduced.

It is tempting to take $\rho = |f|$ and sample directly from the function to be integrated. However in practice it is typically expensive to evaluate the integrand. Therefore a better strategy is to build an approximate density in the product form, $\rho(x, y, \ldots, z) = \rho_x(x)\rho_y(y)\ldots\rho_z(z)$, and then sample from this approximate density. A popular routine of this sort is called VEGAS. The sampling from a given function can be done using the *Metropolis algorithm* which we shall not discuss here.

## Stratified sampling

Stratified sampling is a generalisation of the recursive adaptive integration algorithm to random quadratures in multi-dimensional spaces.

The ordinary "dividing by two" strategy does not work for multi-dimensions as the number of subvolumes grows way too fast to keep track of. Instead one estimates along which dimension a subdivision should bring the most dividends and only subdivides along this dimension. Such strategy is called *recursive stratified sampling*. A simple variant of this algorithm is given in table .

In a stratified sample the points are concentrated in the regions where the variance of the function is largest, as illustrated on figure .

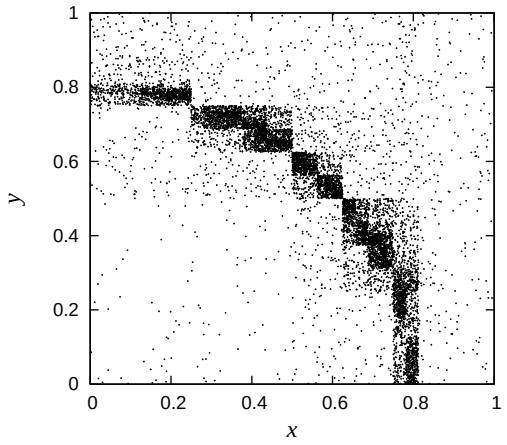


Figure 1: Stratified sample of a discontinuous function $f(x, y) = (x^2 + y^2 < 0.8^2)\ ?\ 1 : 0$

Table 2: Recursive stratified sampling algorithm

```
sample N random points with plain Monte Carlo;
estimate the average and the error;
IF the error is acceptable :
  RETURN the average and the error;
ELSE :
  FOR EACH dimension :
    subdivide the volume in two along the dimension;
    estimate the sub-variances in the two sub-volumes;
  pick the dimension with the largest sub-variance;
  subdivide the volume in two along this dimension;
  dispatch two recursive calls to each of the sub-volumes;
  estimate the grand average and grand error;
  RETURN the grand average and grand error;
```

## Quasi-random (low-discrepancy) sampling

Pseudo-random sampling has high discrepancy[1] – it typically creates regions with hight density of points and other regions with low density of points, as illustrated on Fig. 2. With pseudo-random sampling there is a finite probability that all the $N$ points would fall into one half of the region and none into the other half.

*Quasi-random* sequences avoid this phenomenon by distributing points in a highly correlated manner with a specific requirement of low discrepancy, see Fig. 2 for an example. Quasi-random sampling is like a computation on a grid where the grid constant must not be known in advance as the grid is ever gradually refined and the points are always distributed uniformly over the region. The computation can be stopped at any time.

By placing points more evenly than at random, the quasi-random sequences try to improve the $1/\sqrt{N}$ convergence rate of pseudo-random sampling.

The central limit theorem does not work in this case as the points are not statistically independent. Therefore the variance can not be used as an estimate of the error. The error estimation is actually not trivial. In practice one can employ two different sequences and use their difference as the error estimate.

Quasi-random sequences can be roughly divided into *lattice rules* and *digital nets*.

### Van der Corput and Halton sequences

A *van der Corput sequence* is a low-discrepancy sequence over the unit interval. It is constructed by reversing the base-$b$ representation of the sequence of natural numbers $(1, 2, 3, \dots)$. For example, the

[1] discrepancy is a measure of how unevenly the points are distributed over the region.

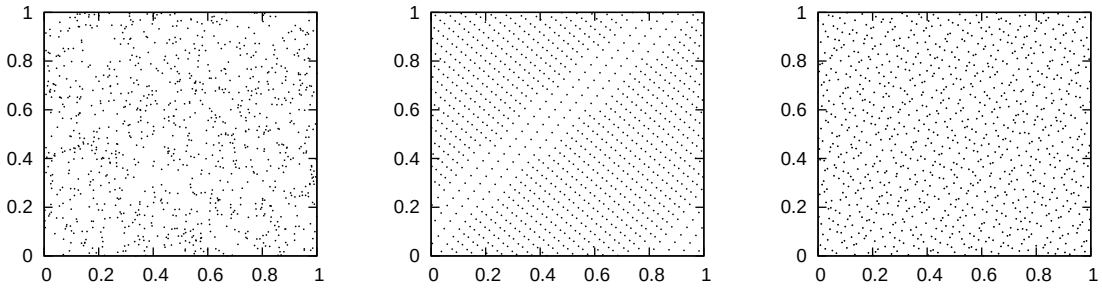

Figure 2: Typical distributions of pseudo-random points (left), and quasi-random (low-discrepancy) points: lattice (center), and base-2/3 Halton (right) sequences.

decimal van der Corput sequence begins as

$$0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.01, 0.11, 0.21, \ldots . \tag{9}$$

In a base-$b$ representation a natural number $n$ with $s$ digits $\{d_i \mid i = 1 \ldots s,\ 0 \le d_i < b\}$ is given as

$$n = \sum_{k=1}^{s} d_k b^{k-1} . \tag{10}$$

The corresponding base-$b$ van der Corput number $q_b(n)$ is given as

$$q_b(n) = \sum_{k=1}^{s} d_k b^{-k} . \tag{11}$$

Table 3 lists a C implementation of this algorithm.

Table 3: van der Corput quasi-random number $q_b(n)$: a C implementation

```
double corput(int n, int base){
   double q=0, bk=(double)1/base;
   while(n>0){ q += (n % base)*bk; n /= base; bk /= base; }
   return q;}
```

The van der Corput numbers in any base are uniformly distributed over the unit interval. They also form a dense set in the unit interval: there exists a subsequence of the van der Corput sequence which converges to any given real number in $[0, 1]$.

The Halton sequence is a generalization of the van der Corput sequence to $d$-dimensional spaces. One chooses a set of coprime bases $b_1, \ldots, b_d$ and then for each dimension $i$ generates a van der Corput sequence with its own base $b_i$. The $n$-th Halton $d$-dimentional point $\mathbf{x}$ (in the unit volume) is then given as

$$\mathbf{x}(n) = \{q_{b_1}(n), \ldots, q_{b_d}(n)\} . \tag{12}$$

**Lattice rules**

In the simplest incarnation a lattice rule can be defined as follows. Let $\alpha_i$, $i = 1, \ldots, d$, (where $d$ is the dimension of the integration space) be a set of cleverly chosen irrational numbers, like square roots of prime numbers. Then the $n$-th point (in the unit volume) of the sequence is given as

$$\mathbf{x}(n) = \{\mathrm{frac}(n\alpha_1), \ldots, \mathrm{frac}(n\alpha_d)\} , \tag{13}$$

where $\mathrm{frac}(x)$ is the fractional part of $x$.

A problem with this method is that a high accuracy arithmetics (e.g. `long double`) might be needed in order to generate a reasonable amount of quasi-random numbers.