Quasi-random number generator

Ines Meršak

21. 4. 2017

1 Pseudo-random vs. quasi-random

Pseudo-random number:

- a computer-generated number
- appears to be random but is generated by an entirely deterministic process
- a pseudo-random process is easier to produce than a genuinely random one
- the benefit of a pseudo-random process is that it can be used again and again to produce exactly the same numbers, which is useful for testing and fixing software

Quasi-random number:

- also called low-discrepancy numbers
- the discrepancy is a measure of how inhomogeneously a set of d-dimensional vectors are distributed in the unit hypercube
- low-discrepancy means the points are distributed more uniformly, with less clusters and gaps that are typical for pseudo-random numbers
- unlike pseudo-random numbers, low-discrepancy numbers aim not to be serially uncorrelated but instead to take the previous draws into account when determining the next number in the sequence

If we take a uniform random generator on $[0, 1)$ and halve the interval, for each trial there is a probability of $\frac{1}{2}$ that the generated point will be in the left interval and a probability of $\frac{1}{2}$ that the point will be in the right interval. generating a point on each of these subintervals. Therefore, it is possible for first n generated points to coincidentally all lie in the first half of the interval, while the next point still falls within the other of the two halves with probability $\frac{1}{2}$. This is not the case with the quasirandom sequences because of the low-discrepancy requirement that has an effect of points being generated in a highly correlated manner (i.e., the next point "knows" where the previous points are).

2 Quasi-random numbers – usage

Quasi-random numbers are useful in computational problems and are especially popular for financial Monte Carlo calculations. Quasi-Monte Carlo calculations (using sequences of quasi-random numbers to compute the integral) asimptotically converge faster than normal Monte Carlo calculations using pseudo-random numbers, even for large dimensionality of drawn vectors d.

3 Sobol' numbers

- we need a new unique generating integer $\gamma(n)$ for each new draw
- easy choice is $\gamma(n) = n$, another possibility is the Gray code $\gamma(n) = G(n)$, which I will not be discussing
- the generation is carried out on a set of integers in the interval $[1, 2^b - 1]$
- b represents the number of bits in an unsigned integer on the given computer and is typically 32
- denote x_{nk} as the nth draw of Sobol' integer in dimension k
- a set of b *direction integers* for each dimension k, which are the basis of the number generation
- there are some additional constraints on the direction integers which I will not discuss
- for each dimension, we select a primitive polynomial modulo two and calculate the direction integers using the coefficients of the polynomial and binary addition

- from there, we calculate x_{nk}: depending on which bits in the binary representation of $\gamma(n)$ are set, the direction integers are XORed to produce the Sobol' integer x_{nk}

4 Project timeline

Work done so far:

- reading the source material
- getting familiar with C++

Plan for the rest of the project:

- implement Sobol' number generator with Gray code
- test the generator with quasi-Monte Carlo integration
- compare results with the parallel version