The Mathematica® Journal

A Toolbox for Quasirandom Simulation

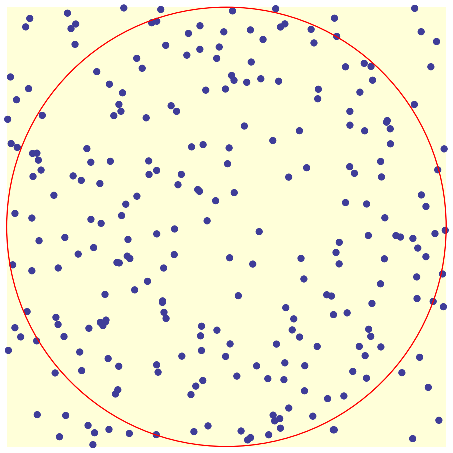
Michael Carter

Quasirandom simulation uses low-discrepancy or quasirandom sequences in place of pseudorandom sequences, producing faster convergence in problems of moderate dimensions. The objective of this article is both pedagogical and practical: to provide an easily understood introduction to the construction of Sobol sequences and a toolkit for constructing and evaluating such sequences.

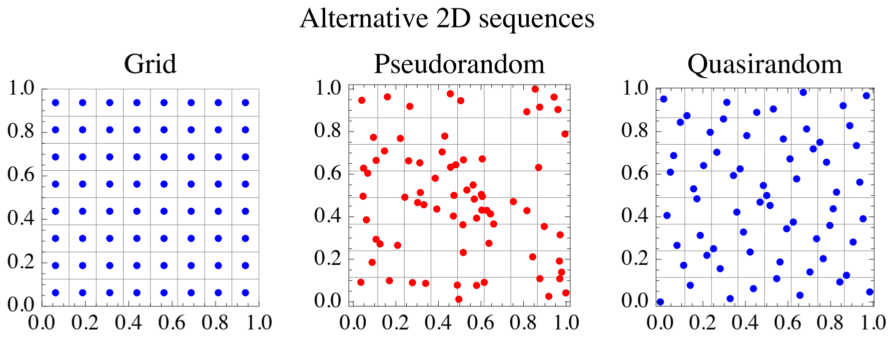
Introduction

In the eighteenth century, Georges-Louis Leclerc, Comte de Buffon, proposed a novel method to estimate π—dropping a needle over and over again onto a wooden floor of parallel planks. The probability of a needle crossing a join in the floor is related to π. By counting the number of crosses, one can estimate this probability, and hence compute a value for π (see [1]). Buffon is said to have tried the method by tossing baguettes over his shoulder. A more direct way of estimating π is to throw darts randomly at a circular target inscribed in a square, and count the proportion that land inside the circle. These are simple examples of numerical integration by simulation.

Estimating π by simulation



Numerical integration requires evaluating a function at a number of distinct points and computing an average value. There are at least three ways in which we could lay out a grid of points on which to evaluate an integral, which are illustrated below. Familiar rules of quadrature, such as Simpson's rule, use a regular grid or lattice. The shortcoming of this approach is that it is difficult to compute a sufficiently dense grid in higher dimensions. For example, a grid of four points in 250 dimensions requires $4^{250} = 3.27339 \times 10^{150}$ points. Furthermore, only a limited number of coordinate values are evaluated. Traditional Monte Carlo simulation uses a pseudorandom grid, so that numerous combinations of points are evaluated. We observe that the points tend to cluster, with many boxes having no points while some boxes have multiple points. Low-discrepancy or quasirandom sequences attempt to combine the best features of both a grid and pseudorandom points while overcoming the disadvantages of each. They are specifically constructed so that they fill the space in a "quasi" random but uniform manner. Observe that each of the boxes in the right-hand graph has one and only one point.



The most common pseudorandom number generator is the *linear congruential generator,* in which successive numbers are generated recursively by

$$x_{n+1} = a\, x_n + c\,(\mathrm{mod}\, m), \tag{1}$$

starting with an initial seed x_0. Dividing by m gives a sequence of fractions $u_i = x_i / m$ in the unit interval $[0, 1)$. By careful choice of a, c, and m, a sequence of period m can be obtained. The resulting sequence of real numbers will appear to be uniformly distributed on $[0, 1)$. For simulation, it is not sufficient to avoid clustering in a single dimension. Most practical problems, from estimating π to valuing an exotic financial derivative, require multidimensional random sequences. In the center of the previous figure, we observe the clustering that is typical of pseudorandom sequences.

Low-discrepancy sequences are known to give superior performance in low-dimensional problems, but their relative advantage erodes as the dimension of the integral increases. One practical area in which quasirandom simulation has shown great promise is finance, where valuation of derivative instruments requires the computation of multidimensional expected values. Such valuations may need to be computed daily, which means that computational efficiency is extremely important. In this application, low-discrepancy sequences have been shown to give improved performance even in very high-dimensional problems.

Low-discrepancy sequences have been proposed by Halton, Faure, Sobol, and Niederreiter. Though not exhibiting the lowest asymptotic discrepancy, Sobol sequences have been found in practice to yield as good or better performance, especially in financial applications. Consequently, this article focuses on Sobol sequences, although the tools are more generally applicable. The objective of this article is both pedagogical and practical: to provide an easily understood introduction to the composition of Sobol sequences and a toolkit for constructing and evaluating such sequences.

■ The Construction of Sobol Sequences

A Sobol sequence is a sequence of points $x_1, x_2, \ldots, x_n$ in the unit hypercube $[0, 1)^d$, where d is the dimension of the problem. In other words, each element x_i of the sequence is a d-dimensional vector whose components are fractions between 0 and 1. A Sobol sequence can be computed by the simple recursion

$$x_{n+1} = v_{j(n)} \oplus x_n. \tag{2}$$

Equation (2) is analogous to the linear congruential generator (1) for pseudorandom numbers. The differences are:

- The coefficients $v_{j(n)}$ vary with n and also with the dimension d (as $v_{j(n)}$ is a vector).
- The operator $\oplus$ is bitwise exclusive or rather than multiplication. It is equivalent to addition modulo 2.

The coefficients $v_{j(n)}$ are known as *direction numbers*. The index $j(n)$ of the appropriate element of the set of direction numbers is the rightmost zero bit in the binary expansion of n. Consequently, to produce a Sobol sequence of length n requires one direction number for each bit in the binary expansion of n, a total of $k = \lceil \log_2 n \rceil$ direction numbers for each dimension. The complicated part of computing a Sobol sequence is computing the direction numbers for each dimension. Once this is done, computing the series using (2) is straightforward and fast. First, we load the accompanying package, assuming that it is in the current directory or a directory in the path.

```
Needs["QRSToolbox`"]
```

The sequence $\left(\frac{1}{2}, \frac{3}{4}, \frac{7}{8}, \frac{5}{16}, \frac{7}{32}, \frac{43}{64}\right)$ is a valid set of direction numbers for generating the first 63 elements of a one-dimensional Sobol sequence. Here are the first 16 elements of this sequence.

```
TableForm[
 {Sobol[{16, 1}, InitialValues → {{11, {1, 3, 7}}},
    Offset → 0] // Flatten},
 TableHeadings → {None, Range[0, 15]}]
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{1}{8}$	$\frac{5}{8}$	$\frac{3}{8}$	$\frac{7}{8}$	$\frac{11}{16}$	$\frac{3}{16}$	$\frac{15}{16}$	$\frac{7}{16}$	$\frac{9}{16}$	$\frac{1}{16}$	$\frac{13}{16}$	$\frac{5}{16}$

The seventh element in this sequence is $\frac{7}{8}$. The binary expansion of 7 is 0111; the rightmost zero bit is four from the end. Therefore the appropriate direction number is the fourth, namely $\frac{5}{16}$. From (2), the eighth element is

$$x_8 = v_4 \oplus x_7 = \frac{5}{16} \oplus \frac{7}{8} = \frac{5}{16} \oplus \frac{14}{16} = \frac{0101_2}{16} \oplus \frac{1110_2}{16} = \frac{1011_2}{16} = \frac{11}{16}.$$

We now turn to the computation of direction numbers, the essential ingredients of a Sobol sequence. A degree k polynomial

$$x^k + a_1\, x^{k-1} + a_2\, x^{k-2} + \ldots + a_{k-1}\, x + a_k$$

defines a unique recursive sequence of order k

$$x_j = a_1\, x_{j-1} + a_2\, x_{j-2} + \ldots + a_{k-1}\, x_{j-k+1} + a_k\, x_{j-k}$$

requiring k initial values. Direction numbers for a Sobol sequence are computed by a special recursive sequence

$$v_j = a_1\, v_{j-1} \oplus a_2\, v_{j-2} \oplus \ldots \oplus a_{k-1}\, v_{j-k+1} \oplus a_k\, v_{j-k} \oplus \frac{v_{j-k}}{2^k}$$

such that:

- The class of polynomials is restricted to the *primitive polynomials mod* 2 (defined in the next section).
- Addition (+) is replaced with bitwise or ($\oplus$).
- An extra term is added.

Computationally, it is convenient to use the equivalent recursion, which requires only integer arithmetic:

$$m_j = 2\, a_1\, m_{j-1} \oplus 2^2\, a_2\, m_{j-2} \oplus \ldots \oplus 2^{k-1}\, a_{k-1}\, m_{j-k+1} \oplus 2^k\, a_k\, m_{j-k} \oplus m_{j-k}. \tag{3}$$

Seeded with k non-negative odd integers $m_1, m_2, m_3, \ldots, m_k$ with $m_i < 2^i$, subsequent values generated by (3) are also non-negative odd integers with $m_j < 2^j$ for all j. Direction numbers are obtained by dividing each term by 2^j, that is

$$(v_1, v_2, \ldots, v_M) = \left(\frac{m_1}{2}, \frac{m_2}{4}, \frac{m_3}{8}, \ldots, \frac{m_M}{2^M}\right).$$

To illustrate, the direction numbers used above were generated from the third-degree primitive polynomial $x^3 + x + 1$, in which $a_1 = 0$ and $a_2 = a_3 = 1$. The corresponding recurrence relation is

$$m_j = 2^2\, m_{j-2} \oplus 2^3\, m_{j-3} \oplus m_{j-3}.$$

Starting with the initial values $\{1, 3, 7\}$, this recurrence generates the sequence $\{1, 3, 7, 5, 7, 43\}$, which consists of the numerators of the direction numbers used above.

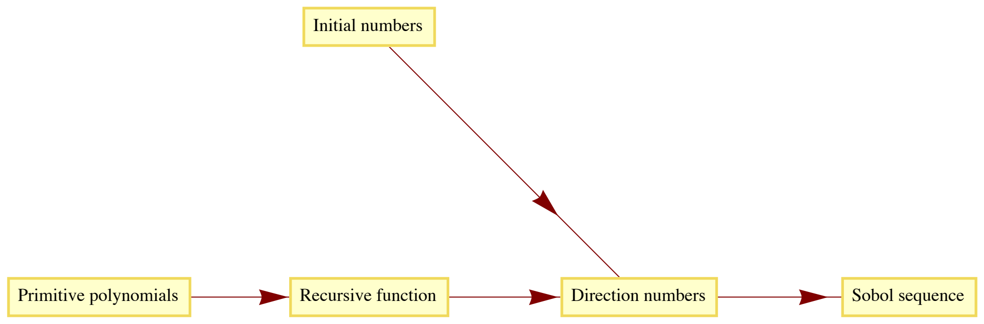
```
CirclePlus[i_Integer, j_Integer] := BitXor[i, j]

f[j_] := 4 f[j - 2] ⊕ (8 f[j - 3] ⊕ f[j - 3])
f[1] = 1; f[2] = 3; f[3] = 7;
Table[f[j], {j, 1, 6}]

{1, 3, 7, 5, 7, 43}
```

Each dimension of the Sobol sequence requires a different primitive polynomial, and each degree k polynomial requires k initial values, odd integers less than $2, 4, \ldots, 2^k$, respectively. The choice of initial values is the one area of discretion left to the modeler. Indeed, the choice of initial values is the most important issue in successful application of Sobol sequences.

Consequently, a toolbox for quasirandom simulation using Sobol sequences needs to provide for the following:

- identifying primitive polynomials modulo 2 to specify the recursions for generating direction numbers
- selecting appropriate initial values to seed the recursions
- computing the direction numbers
- generating the resulting multidimensional Sobol sequence

These ingredients are depicted schematically in the following diagram.



In addition, our toolbox provides functions for computing the discrepancy on any sequence, which can be used to evaluate initial values for computing Sobol sequences and to compare the effectiveness of pseudorandom and quasirandom sequences.

Primitive Polynomials Mod 2

A polynomial modulo 2 is a polynomial

$$x^k + a_1 x^{k-1} + \ldots + a_{k-1} x + a_k$$

whose coefficients $a_1, a_2, \ldots, a_k$ are either zero or one. The highest power is the degree of the polynomial. There are 2^3 third-degree polynomials mod 2, namely

$$x^3, x^3 + 1, x^3 + x^2, x^3 + x^2 + 1, x^3 + x^2 + x, x^3 + x^2 + x + 1, x^3 + x, x^3 + x + 1.$$

The term primitive polynomial has two distinct meanings in algebra (see [2]). For our purposes, a polynomial mod 2 is *primitive* if (in binary arithmetic):

- it is *irreducible* (cannot be factored)
- the smallest power q for which the polynomial divides $x^q + 1$ is $q = 2^k - 1$

Only two of the eight third-degree polynomials are primitive, namely

$$x^3 + x^2 + 1, x^3 + x + 1.$$

Note that a polynomial with no constant term can always be factored as

$$x^k + a_1 x^{k-1} + \ldots + a_{k-1} x = x\left(x^{k-1} + a_1 x^{k-2} + \ldots + a_{k-1}\right).$$

Therefore, every primitive polynomial has a constant term. For the rest of this article, polynomial means polynomial modulo 2.

□ Encoding

Any polynomial mod 2 can be uniquely encoded as an integer by interpreting the coefficients (0 or 1) as bits. For example, the primitive third-degree polynomial $x^3 + x + 1$ can be encoded as binary 1011_b or decimal 11. The *Mathematica* functions `EncodeP` and `DecodeP` encode and decode polynomials.

The two primitive third-degree polynomials are encoded.

```
EncodeP[{x^3 + x + 1, x^3 + x^2 + 1}]
```

```
{11, 13}
```

They can be decoded.

```
DecodeP /@ %
```

```
{1 + x + x^3, 1 + x^2 + x^3}
```

For the remainder of the article, we will call a polynomial's unique decimal code its *p*-number.

□ Polynomials of Degree *k*

A polynomial with *p*-number p is of degree k if and only if $2^k \leq p < 2^{k+1}$. This provides a very straightforward method for generating all the polynomials of a given degree, namely by decoding the consecutive integers between 2^k and 2^{k+1}. For example, here are the third-degree polynomials.

```
DecodeP /@ Range[2^3, 2^(3+1) - 1]
```

```
{x^3, 1 + x^3, x + x^3, 1 + x + x^3,
 x^2 + x^3, 1 + x^2 + x^3, x + x^2 + x^3, 1 + x + x^2 + x^3}
```

We note that a polynomial has a constant term if and only if it has an odd *p*-number. Therefore, to compute primitive polynomials we need consider only those with an odd *p*-number.

```
DecodeP /@ Range[2^3 + 1, 2^(3+1) - 1, 2]
```

```
{1 + x^3, 1 + x + x^3, 1 + x^2 + x^3, 1 + x + x^2 + x^3}
```

The function `PolynomialMod2[k]` generates the polynomials of degree k. By default, `PolynomialMod2` gives only polynomials with a constant term, which are candidates for being primitive.

```
PolynomialMod2[3]
```

```
{1 + x^3, 1 + x + x^3, 1 + x^2 + x^3, 1 + x + x^2 + x^3}
```

Setting the option `WithConstantTerm` to `False` generates all polynomials of the specified degree, including those with no constant term.

```
PolynomialMod2[3, WithConstantTerm → False]
```

```
{x^3, 1 + x^3, x + x^3, 1 + x + x^3,
 x^2 + x^3, 1 + x^2 + x^3, x + x^2 + x^3, 1 + x + x^2 + x^3}
```

□ Irreducible Polynomials

Suppose we attempt to factor the third-degree polynomials with constant term.

```
Text@
 TableForm[TraditionalForm /@ {#, Factor[#, Modulus → 2]} & /@
   PolynomialMod2[3],
  TableHeadings → {None, {"Polynomial", "Factorization"}}]
```

Polynomial	Factorization
x^3+1	$(x+1)\left(x^2+x+1\right)$
x^3+x+1	x^3+x+1
x^3+x^2+1	x^3+x^2+1
x^3+x^2+x+1	$(x+1)^3$

We observe that the first and last polynomials can be factored, while the other two are irreducible. Now examine the heads of the expressions after attempted factorization.

```
Head@Factor[#, Modulus → 2] & /@ PolynomialMod2[3]
```

```
{Times, Plus, Plus, Power}
```

The irreducible polynomials have a head of `Plus`, while the other two have heads of `Times` or `Power`. We can use this to identify irreducible polynomials.

```
IrreducibleQ[poly_] := Head[Factor[poly, Modulus → 2]] === Plus
```

Here are the irreducible third-degree polynomials.

```
Select[PolynomialMod2[3], IrreducibleQ]
```

$\{1+x+x^3,\ 1+x^2+x^3\}$

Primitive Polynomials

If $q = 2^k - 1$ is prime, all the irreducible polynomials of degree k are primitive. If not, we must test each of the prime factors of q. For example, $2^4 - 1 = 15$ is not prime. Here are the irreducible polynomials of degree four.

```
irreducible4 = Select[PolynomialMod2[4], IrreducibleQ]
```

$\{1+x+x^4,\ 1+x^3+x^4,\ 1+x+x^2+x^3+x^4\}$

For each prime factor m of $q = 2^k - 1$, we have to test whether the polynomial divides $x^{q/m} + 1$. For $k = 4$ and $q = 15$, the prime factors are 3 and 5, and their cofactors are 5 and 3, respectively. That is, we need to check whether each polynomial divides $x^5 + 1$ or $x^3 + 1$. Clearly, a fourth-degree polynomial cannot divide $x^3 + 1$. Therefore, we need only check those cofactors less than k. That is, we check whether the irreducible polynomials of degree 4 divide $x^5 + 1$.

```
PolynomialRemainder[x^5 + 1, #, x, Modulus → 2] & /@
 irreducible4
```

$\{1+x+x^2,\ x+x^3,\ 0\}$

This reveals that the third irreducible polynomial divides $x^5 + 1$. We conclude that the first and second polynomials are primitive, while the third is not.

All fifth-degree irreducible polynomials are primitive, since $2^5 - 1 = 31$ is prime. Considering the sixth-degree polynomials, the prime factors of $2^6 - 1 = 63$ are 3 and 7, and their cofactors are 21 and 9. Consequently, we need to consider the divisibility of the polynomials $x^{21} + 1$ and $x^9 + 1$.

```
irreducible6 = Select[PolynomialMod2[6], IrreducibleQ];
```

```
PolynomialRemainder[x^21 + 1, #, x, Modulus → 2] & /@
 irreducible6
```

```
{x + x^3 + x^4 + x^5, 1 + x^3, 0, 1 + x + x^2 + x^3,
 x^3 + x^4 + x^5, 1 + x^2 + x^3 + x^5, x + x^2 + x^4, x^3 + x^4, 0}
```

```
PolynomialRemainder[x^9 + 1, #, x, Modulus → 2] & /@
 irreducible6
```

```
{1 + x^3 + x^4, 0, 1 + x + x^2 + x^4, x^2 + x^4 + x^5,
 x + x^2 + x^3 + x^5, x^2 + x^3, 1 + x + x^2, x + x^2 + x^5, x^3}
```

```
Remove[irreducible4, irreducible6]
```

We observe that the third and last polynomial divide $x^{21} + 1$, while the second polynomial divides $x^9 + 1$. The remaining six polynomials are primitive. We summarize this test in the function `PrimitiveQ`.

```
? PrimitiveQ
```

Assuming poly is a polynomial modulo 2, PrimitiveQ[poly]
gives True if poly is primitive, and False otherwise.

The following function generates the primitive polynomials of degree k.

```
PrimitivePolynomialsMod2[k_Integer, x_] :=
 Select[PolynomialMod2[k], PrimitiveQ[#, x] &]
PrimitivePolynomialsMod2[1, x_] := {1 + x}
```

Here are p-numbers of the 52 primitive polynomials up to degree eight.

```
Table[{k, PrimitivePolynomialsMod2[k] // EncodeP}, {k, 8}] //
 Column
```

```
{1, {3}}
{2, {7}}
{3, {11, 13}}
{4, {19, 25, 31}}
{5, {37, 41, 47, 55, 59, 61}}
{6, {67, 73, 87, 91, 97, 103, 109, 115, 117}}
{7, {131, 137, 143, 145, 157, 167, 171, 185,
   191, 193, 203, 211, 213, 229, 239, 241, 247, 253}}
{8, {283, 285, 299, 301, 313, 319, 333, 351, 355, 357,
   361, 369, 375, 379, 391, 395, 397, 415, 419, 425,
   433, 445, 451, 463, 471, 477, 487, 499, 501, 505}}
```

The number N of primitive polynomials of degree k can be easily computed from Euler's totient function ϕ. Specifically,

$$N(k) = \frac{\phi\left(2^k - 1\right)}{k}.$$

There are some published sources for primitive polynomials. *Numerical Recipes* [3] lists all 160 primitive polynomials of degree 10 or less (but note that they use a different encoding). Joe and Kuo [4] provide a list of the 1111 primitive polynomials through degree 13, which they have recently extended to degree 18 [5]. The CD accompanying Jäckel [6] lists all primitive polynomials up to degree 27, a mammoth eight million primitive polynomials.

Implementation

We now give details of the *Mathematica* implementation of these components.

Direction Numbers

Recall that direction numbers are computed by the recursive sequence (3)

$$m_j = 2\, a_1\, m_{j-1} \oplus 2^2\, a_2\, m_{j-2} \oplus \ldots \oplus 2^{k-1}\, a_{k-1}\, m_{j-k+1} \oplus 2^k\, m_{j-k} \oplus m_{j-k},$$

where $a_1, a_2, \ldots, a_{k-1}$ are the coefficients of a degree k primitive polynomial ($a_k = 1$). Each recursion of order k requires k initial values, which are arbitrary odd integers less than $2, 4, \ldots, 2^k$, respectively. To implement this, we extract the binary digits from the p-number of the specified polynomial, use them to define an appropriate recursive function, initialize the function with the vector M of initial values, and then map this function over the integers $1, 2, \ldots, n$. The resulting function is called `DirectionNumbers`.

```
? DirectionNumbers
```

```
DirectionNumbers[{p,M},n] computes n direction numbers for polynomial
    p using the initial values M. DirectionNumbers[p,n] computes
    n direction numbers for polynomial p using unit initial values.
```

Here are the first six direction numbers for the third primitive polynomial $x^3 + x^2 + 1$ with initial values $\{1, 3, 7\}$.

```
DirectionNumbers[{11, {1, 3, 7}}, 6]
```

```
{1, 3, 7, 5, 7, 43}
```

Note that `DirectionNumbers` returns the (integer) numerators of the direction number sequence, which is the form in which they will be used for generating the Sobol sequence. If no direction numbers are specified, the function assumes all initial values are one (*unit initialization*).

□ Recursive Sobol Sequence

It is most efficient to compute all d dimensions of a Sobol sequence in parallel, which is especially convenient in *Mathematica*. To illustrate, we take the primitive polynomials and initial values listed in *Numerical Recipes*, which are sufficient to generate a six-dimensional sequence.

```
                   ⎛ 3        {1}      ⎞
                   ⎜ 7       {1, 1}    ⎟
NRInitialValues =  ⎜11     {1, 3, 7}   ⎟ ;
                   ⎜13     {1, 3, 3}   ⎟
                   ⎜19   {1, 1, 3, 13} ⎟
                   ⎝25   {1, 1, 5, 9}  ⎠

SetOptions[Sobol, InitialValues → NRInitialValues];
```

To generate 10 points requires $k = 4$ sets of direction numbers, each set containing direction numbers for six dimensions.

```
Transpose[DirectionNumbers[#, 4] & /@ NRInitialValues]

{{1, 1, 1, 1, 1, 1}, {3, 1, 3, 3, 1, 1},
 {5, 7, 7, 3, 3, 5}, {15, 11, 5, 15, 13, 9}}
```

To enable integer computation, it is convenient to scale the first set of direction numbers by $2^{k-1} = 2^3 = 8$, the second by $2^{k-2} = 2^2 = 4$, and so on.

```
With[{k = 4},
 scaledDN = Table[2^j, {j, k - 1, 0, -1}] *
   Transpose[DirectionNumbers[#, k] & /@ NRInitialValues]]

{{8, 8, 8, 8, 8, 8}, {12, 4, 12, 12, 4, 4},
 {10, 14, 14, 6, 6, 10}, {15, 11, 5, 15, 13, 9}}
```

The sequence can then be computed efficiently using `FoldList` to implement (2) across all dimensions in parallel, then converted to fractions by multiplying each result by $2^{-k} = 2^4 = 16$. Here are the first 10 points of the six-dimensional Sobol sequence using the initial values listed in *Numerical Recipes*.

```
With[{n = 10, k = 4},
 2^-k FoldList[BitXor, scaledDN[[1]],
   scaledDN[[RightMostZero /@ Range[1, n - 1]]]]]
```

$$\left\{\left\{\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2}\right\},\left\{\frac{1}{4},\frac{3}{4},\frac{1}{4},\frac{1}{4},\frac{3}{4},\frac{3}{4}\right\},\right.$$
$$\left\{\frac{3}{4},\frac{1}{4},\frac{3}{4},\frac{3}{4},\frac{1}{4},\frac{1}{4}\right\},\left\{\frac{3}{8},\frac{5}{8},\frac{1}{8},\frac{5}{8},\frac{1}{8},\frac{7}{8}\right\},$$
$$\left\{\frac{7}{8},\frac{1}{8},\frac{5}{8},\frac{1}{8},\frac{5}{8},\frac{3}{8}\right\},\left\{\frac{1}{8},\frac{3}{8},\frac{3}{8},\frac{7}{8},\frac{7}{8},\frac{1}{8}\right\},$$
$$\left\{\frac{5}{8},\frac{7}{8},\frac{7}{8},\frac{3}{8},\frac{3}{8},\frac{5}{8}\right\},\left\{\frac{5}{16},\frac{5}{16},\frac{11}{16},\frac{9}{16},\frac{11}{16},\frac{3}{16}\right\},$$
$$\left.\left\{\frac{13}{16},\frac{13}{16},\frac{3}{16},\frac{1}{16},\frac{3}{16},\frac{11}{16}\right\},\left\{\frac{1}{16},\frac{9}{16},\frac{15}{16},\frac{13}{16},\frac{7}{16},\frac{15}{16}\right\}\right\}$$

`Sobol[{n, s}]` generates a Sobol sequence of n points of dimension s. We give the arguments (n, s) as a list to conform with the pseudorandom generator `RandomReal`, and also to suggest that the result is an $n \times s$ matrix of real numbers.

We now describe a number of useful variants of the basic function.

□ Nonrecursive Sobol Sequence

The recursive construction of the Sobol sequence (2) was proposed by Antonov and Saleev [7]. The original construction of Sobol is

$$x_n = b_1\, v_1 \oplus b_2\, v_2 \oplus \ldots \oplus b_k\, v_k, \tag{4}$$

where $b_1, b_2, \ldots, b_k$ are the binary digits of n, that is $n = b_1\, 2^0 + b_2\, 2^1 + b_3\, 2^2 + \ldots + b_k\, 2^{k-1}$. The recursive definition (2) is obtained from (4) using a Gray code encoding of n. This alters the order of the sequence without changing the asymptotic discrepancy. The original definition is useful for constructing single elements of the sequence.

`Sobol[n, s]` gives the n^{th} element in the (recursive) Sobol sequence. `Sobol[n, s, GrayCode → False]` gives the n^{th} element in the original Sobol sequence. For example, here is the 10th element in the recursive sequence we computed in the previous subsection.

```
Sobol[10, 6]
```

$$\left\{\frac{1}{16}, \frac{9}{16}, \frac{15}{16}, \frac{13}{16}, \frac{7}{16}, \frac{15}{16}\right\}$$

This is the 15th element in the sequence constructed using (4).

```
Sobol[15, 6, GrayCode → False]
```

$$\left\{\frac{1}{16}, \frac{9}{16}, \frac{15}{16}, \frac{13}{16}, \frac{7}{16}, \frac{15}{16}\right\}$$

Equation (4) is useful for initiating the sequence at an arbitrary starting point.

□ Initial Offset

Note the repetition of coordinates in the initial points of the sequence. For example, in the first 10 points of a three-dimensional sequence, many of the coordinates appear three times.

```
Sobol[{10, 3}] // Flatten // Sort
```

$$\left\{\frac{1}{16}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{3}{16}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{5}{16}, \frac{5}{16}, \frac{3}{8}, \frac{3}{8}, \frac{3}{8}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{9}{16}, \frac{5}{8}, \frac{5}{8}, \frac{5}{8}, \frac{11}{16}, \frac{3}{4}, \frac{3}{4}, \frac{3}{4}, \frac{13}{16}, \frac{13}{16}, \frac{7}{8}, \frac{7}{8}, \frac{7}{8}, \frac{15}{16}\right\}$$

To avoid this repetition, many authors recommend discarding the first m points in a Sobol sequence, although the choice of m is arbitrary. To this end, we modify the definition to allow setting an initial offset as an option. Setting this option `Offset → m` will use (4) to generate the first point of the sequence and then use (2) to generate the remaining points of the sequence.

Strictly speaking, the Sobol sequence begins with the point $(0, 0, \ldots, 0)$, although most implementations exclude this point. A sequence beginning with zero can be obtained by setting the `Offset → 0`.

```
Sobol[{4, 6}, Offset → 0]
```

$$\left\{\{0, 0, 0, 0, 0, 0\}, \left\{\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right\}, \left\{\frac{1}{4}, \frac{3}{4}, \frac{1}{4}, \frac{1}{4}, \frac{3}{4}, \frac{3}{4}\right\}, \left\{\frac{3}{4}, \frac{1}{4}, \frac{3}{4}, \frac{3}{4}, \frac{1}{4}, \frac{1}{4}\right\}\right\}$$

The default value of `Offset` is 1, generating a sequence beginning with $\left(\frac{1}{2}, \frac{1}{2}, \ldots, \frac{1}{2}\right)$. That is, `Offset` specifies the ordinal number of the first element in the sequence.

□ Specifying the Polynomials and Initial Values

We show below that selection of the initial values is important in determining the quality of the Sobol sequence. We next allow for the initial values to be specified as an option. This is useful for selecting particular dimensions, such as portraying 2D projections. Initial values are specified as a list, each element of which is a polynomial p-number followed by a list of initial values equal in number to the degree of the polynomial.

The following is the two-dimensional sequence extracted from the third and fourth coordinates of the *Numerical Recipes* implementation.

$$\texttt{Sobol}\left[\{10, 2\}, \texttt{InitialValues} \to \begin{matrix} 11 & \{1, 3, 7\} \\ 13 & \{1, 3, 3\} \end{matrix}\right]$$

$$\left\{\left\{\frac{1}{2}, \frac{1}{2}\right\}, \left\{\frac{1}{4}, \frac{1}{4}\right\}, \left\{\frac{3}{4}, \frac{3}{4}\right\}, \left\{\frac{1}{8}, \frac{5}{8}\right\}, \left\{\frac{5}{8}, \frac{1}{8}\right\}, \left\{\frac{3}{8}, \frac{7}{8}\right\}, \left\{\frac{7}{8}, \frac{3}{8}\right\}, \left\{\frac{11}{16}, \frac{9}{16}\right\}, \left\{\frac{3}{16}, \frac{1}{16}\right\}, \left\{\frac{15}{16}, \frac{13}{16}\right\}\right\}$$

The alternative initial values proposed by Joe and Kuo [5] give a different sequence.

```
Sobol[{10, 2}, InitialValues → 11  {1, 3, 1}
                               13  {1, 1, 1}]
```

$$\left\{\left\{\frac{1}{2},\frac{1}{2}\right\},\left\{\frac{1}{4},\frac{3}{4}\right\},\left\{\frac{3}{4},\frac{1}{4}\right\},\left\{\frac{7}{8},\frac{3}{8}\right\},\left\{\frac{3}{8},\frac{7}{8}\right\},\left\{\frac{5}{8},\frac{5}{8}\right\},\left\{\frac{1}{8},\frac{1}{8}\right\},\left\{\frac{7}{16},\frac{9}{16}\right\},\left\{\frac{15}{16},\frac{1}{16}\right\},\left\{\frac{3}{16},\frac{5}{16}\right\}\right\}$$

□ Specifying the Polynomials

For some purposes, it will be useful to select only the primitive polynomials and have the initial values supplied by the default value of the `InitialValues` option. The following input gives the third and fourth coordinates of the *Numerical Recipes* sequence.

```
Sobol[{10, 2}, Polynomials → {11, 13}]
```

$$\left\{\left\{\frac{1}{2},\frac{1}{2}\right\},\left\{\frac{1}{4},\frac{1}{4}\right\},\left\{\frac{3}{4},\frac{3}{4}\right\},\left\{\frac{1}{8},\frac{5}{8}\right\},\left\{\frac{5}{8},\frac{1}{8}\right\},\left\{\frac{3}{8},\frac{7}{8}\right\},\left\{\frac{7}{8},\frac{3}{8}\right\},\left\{\frac{11}{16},\frac{9}{16}\right\},\left\{\frac{3}{16},\frac{1}{16}\right\},\left\{\frac{15}{16},\frac{13}{16}\right\}\right\}$$

This gives a two-dimensional sequence based on the same polynomials, but with different starting values.

```
Sobol[{10, 2}, Polynomials → {11, 13},
 InitialValues → 11  {1, 3, 1}
                 13  {1, 1, 1}]
```

$$\left\{\left\{\frac{1}{2},\frac{1}{2}\right\},\left\{\frac{1}{4},\frac{3}{4}\right\},\left\{\frac{3}{4},\frac{1}{4}\right\},\left\{\frac{7}{8},\frac{3}{8}\right\},\left\{\frac{3}{8},\frac{7}{8}\right\},\left\{\frac{5}{8},\frac{5}{8}\right\},\left\{\frac{1}{8},\frac{1}{8}\right\},\left\{\frac{7}{16},\frac{9}{16}\right\},\left\{\frac{15}{16},\frac{1}{16}\right\},\left\{\frac{3}{16},\frac{5}{16}\right\}\right\}$$

◻ Unit Initialization

Recall that k^{th}-order recursion for building direction numbers requires k initial values that are odd integers less than $2, 4, \ldots, 2^k$. The simplest valid choice for the initial values is to set them all equal to one, which we will call *unit initialization*. We add this as an option. We will demonstrate in the next section that unit initialization does not produce sequences with the lowest discrepancy.

```
Sobol[{10, 2}, Polynomials → {11, 13}, InitialValues → Unit]
```

$$\left\{\left\{\frac{1}{2}, \frac{1}{2}\right\}, \left\{\frac{3}{4}, \frac{3}{4}\right\}, \left\{\frac{1}{4}, \frac{1}{4}\right\}, \left\{\frac{3}{8}, \frac{3}{8}\right\}, \left\{\frac{7}{8}, \frac{7}{8}\right\}, \left\{\frac{5}{8}, \frac{5}{8}\right\}, \left\{\frac{1}{8}, \frac{1}{8}\right\}, \left\{\frac{15}{16}, \frac{9}{16}\right\}, \left\{\frac{7}{16}, \frac{1}{16}\right\}, \left\{\frac{3}{16}, \frac{5}{16}\right\}\right\}$$

◻ Summary

`Sobol[{n, s}]` generates a Sobol sequence of n points of dimension s, while `Sobol[n, s]` gives the n^{th} element in the same sequence. The function `Sobol` has the following options, which are listed with their default values:

Option	Default
InitialValues	JK2007InitialValues
Offset	1
Polynomials	Automatic
GrayCode	True

The initial values listed in *Numerical Recipes* [3] are only sufficient to generate a sequence of six dimensions. Consequently, we adopt the initial values provided by Joe and Kuo [5] as the default option for generating Sobol sequences.

Joe and Kuo provide optimized initial values sufficient for 7800 dimensions on their website at web.maths.unsw.edu.au/~fkuo/sobol/index.html. For economy, we have incorporated only the first 100 values into this accompanying package.

```
SetOptions[Sobol, InitialValues → JK2007InitialValues];
```

■ Evaluation

There are at least three ways in which the performance of low-discrepancy sequences can be evaluated:

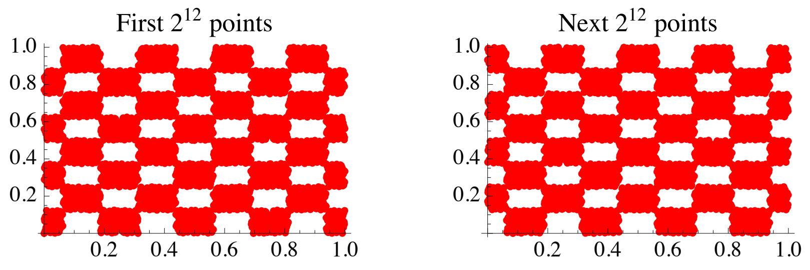
- viewing two-dimensional projections
- measuring discrepancy
- evaluating performance in application

We illustrate each in turn.

□ Projections

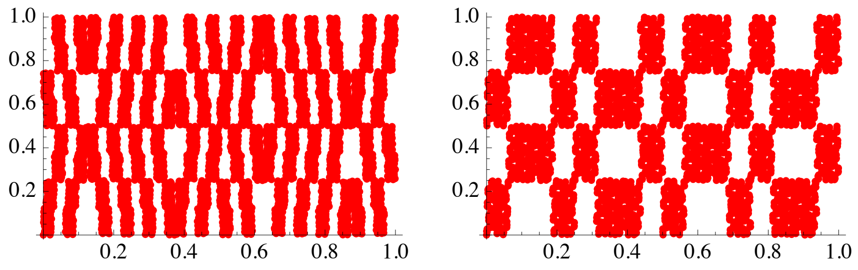
A graphic way to explore the discrepancy of pseudorandom and quasirandom sequences is to plot two-dimensional projections. The following graph shows a striking example of the failure of a Sobol sequence to approximate a uniform distribution in particular dimensions. The graph on the left shows the clustering of the first 4096 points in the sequence. The graph on the right shows that the distribution of the next 4096 points exactly complements the distribution of the first 4096 points.

Dimensions 19 and 28 of a standard Sobol sequence



This particular example, taken from Joe and Kuo [5], uses the initial values given by Bratley and Fox [8] in their well-known implementation. Below, we depict two other examples drawn from [5].

Figures 1 and 4 from Joe and Kuo (2007)



These examples underline the importance of the choice of initial values. Joe and Kuo [5] select the initial values specifically to improve these two-dimensional projections. We restore these as the default.

```
SetOptions[Sobol, InitialValues -> JK2007InitialValues];
```

◻ Measuring Discrepancy

Discrepancy is a measure of the difference between the actual distribution of points and a uniform distribution in which the number of points in any set A is proportional to its size. Specifically, let $\mathcal{A}$ be a collection of subsets of the hypercube $[0, 1)^d$. The discrepancy of the sequence $x_1, x_2, \ldots, x_n$ relative to $\mathcal{A}$ is

$$D = \sup_{A \in \mathcal{A}} \left| \frac{\sharp\{x_i \in A\}}{n} - \mu(A) \right|, \tag{5}$$

where $\sharp\{x_i \in A\}$ denotes the number of points in A and $\mu(A)$ is the measure of A. Different collections $\mathcal{A}$ of sets give rise to different measures. Taking $\mathcal{A}$ to be the collection of all rectangles $\prod_{j=1}^{d} [u_j, v_j)$ in $[0, 1)^d$ gives the *ordinary* discrepancy; taking $\mathcal{A}$ to be the collection of all rectangles $\prod_{j=1}^{d} [0, v_j)$ in $[0, 1)^d$ gives the *star* discrepancy. The L_∞ norm in (5) is useful in theoretical evaluation, but impractical for measuring the discrepancy of specific sequences. If we substitute the Euclidean L_2 norm, it is possible to derive explicit formulas for both ordinary T_n and the star T_n^* discrepancy of a given sequence [9]:

$$(T_n)^2 = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \prod_{k=1}^{d} \left(1 - \max\left(x_{i,k}, x_{j,k}\right)\right) \cdot \min\left(x_{i,k}, x_{j,k}\right) - \frac{2^{1-d}}{n} \sum_{i=1}^{n} \prod_{k=1}^{d} x_{i,k}(1 - x_{i,k}) + 12^{-d}, \tag{6}$$

$$(T_n^*)^2 = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \prod_{k=1}^{d} \left(1 - \max\left(x_{i,k}, x_{j,k}\right)\right) - \frac{2^{1-d}}{n} \sum_{i=1}^{n} \prod_{k=1}^{d} \left(1 - x_{i,k}{}^2\right) + 3^{-d}.$$

These calculations are implemented in the functions `DiscrepancySqd` and `StarDiscrepancySqd`.

```
? DiscrepancySqd
```

```
DiscrepancySqd[X] computes the discrepancy
   (squared) of X, a sequence of n points in the d dimensional
   unit hypercube [0,1)^d, where n × d are the dimensions of X.
```

We can also calculate the expected value of these quantities for a genuinely random sequence [9]:

$$E\left[T_N{}^2\right] = \frac{1}{N}\, 6^{-d}\left(1 - 2^{-d}\right), E\left[(T_N^*)^2\right] = \frac{1}{N}\left(2^{-d} - 3^{-d}\right). \tag{7}$$

These formulas are implemented in the corresponding functions `EDiscrepancySqd` and `EStarDiscrepancySqd`.

To illustrate, the following computation shows the discrepancy of a three-dimensional Sobol sequence relative to that expected of a purely random sequence. Remember that discrepancy measures the degree of nonuniformity in the distribution of the sequence. By this measure, we observe that the quasirandom sequence is three to six times more uniform than what would be expected of a purely random sequence. However, the advantage of the quasirandom sequence erodes as the number of dimensions is increased. (This input takes a long time to evaluate.)

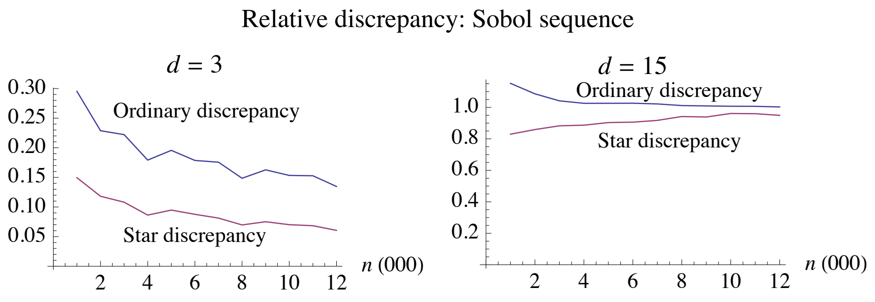
```
With[{X = Sobol[{1024, 3}]},
 {Sqrt[DiscrepancySqd[N@X] / EDiscrepancySqd[X]], Sqrt[StarDiscrepancySqd[N@X] / EStarDiscrepancySqd[X]]}]
```

```
{0.295287, 0.14947}
```

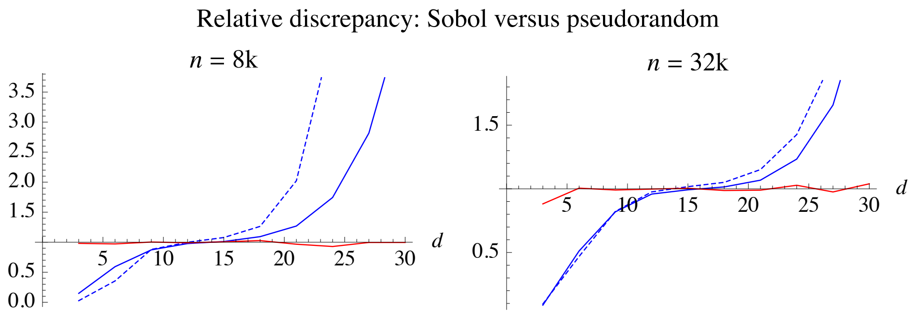
Since computing discrepancy using (6) requires combining every pair of random vectors x_i, x_j, the time required increases linearly with the dimension s but quadratically with n. Computation of (6) in *Mathematica* will run significantly faster if we ensure that floating-point rather than exact arithmetic is used, which is why we apply `N` to the arguments of `DiscrepancySqd` and `StarDiscrepancySqd` in the previous calculation.

Our implementation of `DiscrepancySqd` and `StarDiscrepancySqd` is a straightforward translation of the expressions in (6). These functions can be written more efficiently and then compiled to improve their execution speed by an order of magnitude. (I am grateful to the referee for demonstrating this.) However, the compiled functions are still too slow to be practical for large values of n. Consequently, the following graphs were produced using discrepancy functions implemented in C++ and accessed through *MathLink*. The source code and the *MathLink* executable together with instructions for its use are available from the author on request.

The following graph compares the discrepancy of Sobol sequences of varying length relative to the expected discrepancy of a purely random sequence, using the two measures of discrepancy. We observe that the two measures are broadly consistent, both indicating that the relative advantage of the quasirandom sequence dissipates with increasing dimension. (The horizontal axis is scaled in binary thousands, that is 1024.) Star discrepancy has generally been favored as a measure because it can be used to bound integration error (see [9, 10]).

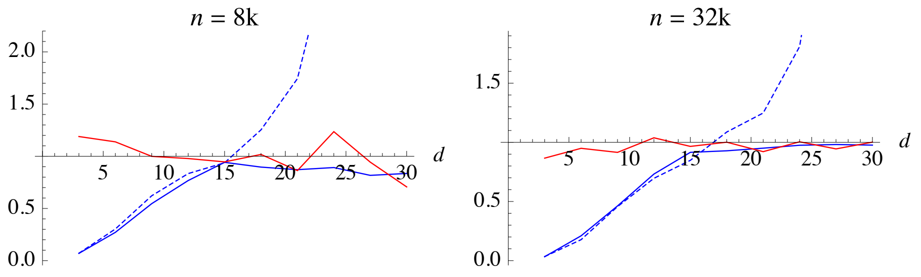


The following graph compares the discrepancy of pseudorandom and quasirandom sequences relative to the expected discrepancy of a purely random sequence as we vary the dimension d. This confirms the conventional wisdom that low-discrepancy sequences (blue) outperform pseudorandom sequences (red) for low dimensions, but that their advantage is eroded as the dimension is increased, with the crossover around $d = 12$ ([11, p. 75]). The measured discrepancy of the pseudorandom sequence (red) hovers around its expected value, so its relative discrepancy is close to the horizontal axis located at 1. The dashed curve shows the discrepancy of a Sobol sequence constructed using unit initialization.



The next graph makes the same comparison using the star discrepancy. Under this measure, the quasirandom sequence (blue) appears to remain competitive with the pseudorandom sequence (red) for higher dimensions. We conclude that the evaluation of the relative performance of quasirandom versus pseudorandom sequences at higher dimensions depends upon the measure of discrepancy used. This graph emphasizes the importance of initial values. The dashed curve shows the relative star discrepancy of a Sobol sequence constructed with unit initialization deteriorates dramatically at higher dimensions.

Relative star discrepancy: Sobol versus pseudorandom

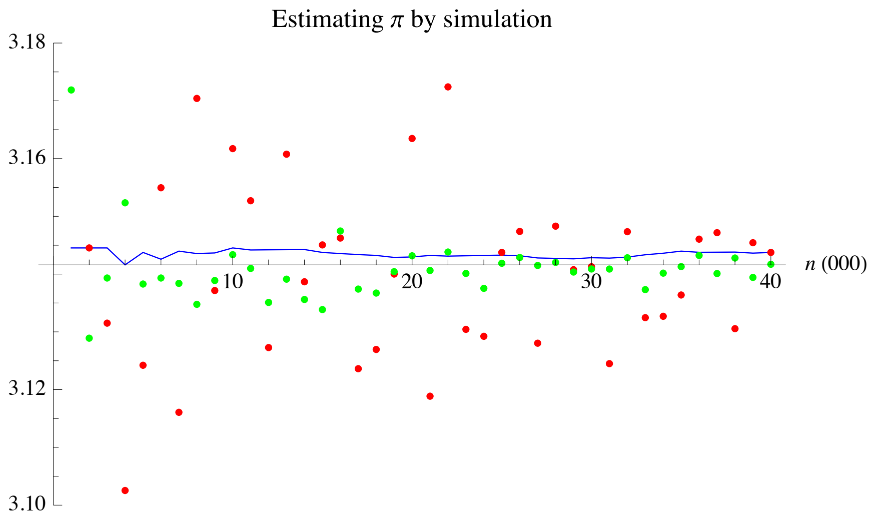


Jäckel [6] includes a more extensive range of comparisons, covering a greater variety of dimensions and other low-discrepancy sequences.

□ Performance in Practice

The following graph compares three methods of estimating π by simulation. The blue line shows the estimate of π from a low-discrepancy sequence of varying sizes, bracketed by the estimates from a rectangular grid (green) and a pseudorandom sequence (red) of identical size. We observe that the low-discrepancy sequence generally provides a more accurate estimate than the rectangular grid, and a significantly better estimate than the pseudorandom sequence. (This input takes a long time to evaluate.)

```
RadiusSqr[{x_, y_}] := (x - 1 / 2) ^ 2 + (y - 1 / 2) ^ 2
PRandom[{n_, s_}] := RandomReal[1, {n, 2}]
EstimatePi[method_, n_] :=
 4 Length[Select[method[{n, 2}], RadiusSqr[#] < 1 / 4 &]] / n

SobolPi = Table[EstimatePi[Sobol, 1024 n], {n, 1, 40}];
PRandomPi = BlockRandom[SeedRandom[123];
   Table[EstimatePi[PRandom, 1024 n], {n, 1, 40}]];
GridPi = Table[EstimatePi[RGrid, 1024 n], {n, 1, 40}];
Show[
 ListLinePlot[SobolPi, PlotStyle → Blue,
  PlotRange → {3.10, 3.18}],
 ListPlot[PRandomPi, PlotStyle → Red],
 ListPlot[GridPi, PlotStyle → Green],
 AxesOrigin → {0, π},
 AxesLabel → {Row[{Style["n", Italic], " (000)"}], None},
 PlotLabel → "Estimating π by simulation",
 PlotRange → {3.10, 3.18}]
```



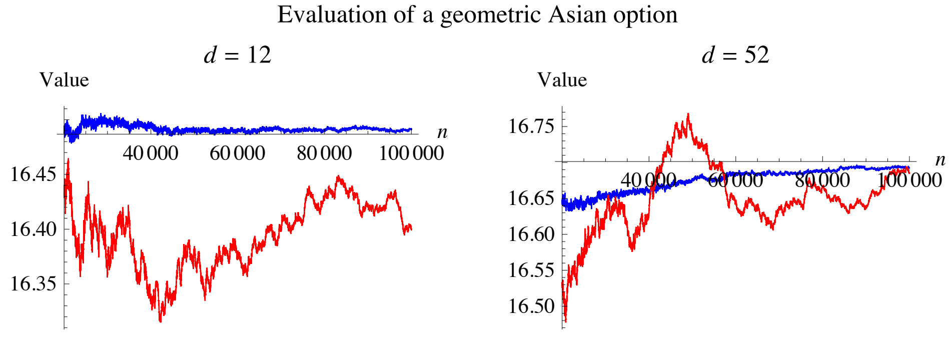
The estimation of π is a one-dimensional problem, where the advantage of low-discrepancy sequences is most pronounced. The world of finance provides a host of multidimensional problems of immense practical importance. A derivative is a financial instrument whose value depends upon the evolution of the price of some underlying asset. Estimating its current value by simulation requires calculating its hypothetical value for each realization, averaging and discounting back to the current time. In the case of a vanilla European option, the payoff of the derivative depends only on the price of the underlying asset at the maturity of the option. For an Asian option, in contrast, the payoff of the derivative depends upon the average price over the term of the option. In effect, estimating the value of an Asian option by simulation amounts to computing a multidimensional integral, with the number of dimensions equal to the number of prices included in the average.

Specifically, the payoff of an Asian option depends upon the average price $\overline{S}$ of the underlying asset during the life of the option. For example, the payoff at maturity of an average price call option is $\max(\overline{S} - K, 0)$, where K is the strike price. There are two ways of calculating the average $\overline{S}$—arithmetic or geometric:

$$A = \frac{1}{m+1}\,(S_0 + S_2 + \ldots + S_m) \geq G = (S_0 \times S_1 \times \ldots \times S_m)^{\frac{1}{m+1}}.$$

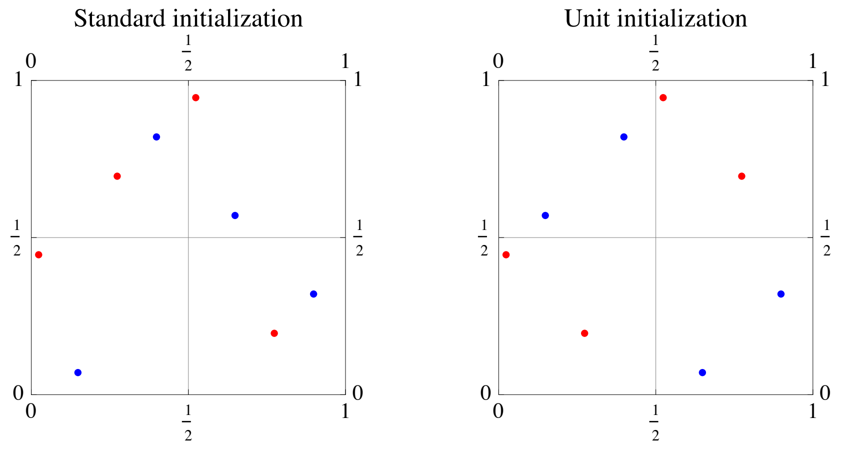
Arithmetic averaging is almost universal in practice. However, geometric averaging is more tractable; indeed, the value of a geometric Asian option is given by an analytical formula. One of the most successful techniques for valuing arithmetic Asian options is simulation, using the known value of a corresponding geometric average option to reduce the simulation error. Consequently, geometric Asian options provide a useful financial test bed for evaluating different simulation methods.

The following graph compares pseudorandom and quasirandom simulations of a one-year geometric average Asian option for varying sample sizes. In the left-hand graphic, the number of dimensions $d = 12$, corresponding to monthly averaging. In the right-hand graphic, $d = 52$, simulating weekly averaging. The axes are drawn at the true value. The law of large numbers ensures that simulated value will converge to the true value eventually. However, as we observe in this diagram, this convergence may be very slow. In both cases, the quasirandom simulation (blue) converges faster than the pseudorandom simulation (red), though the advantage erodes as the number of dimensions increases. We find that the superiority of Sobol sequences in practical applications extends to higher dimensions than might be suggested by considering discrepancy alone. In this example, quasirandom simulation based on a Sobol sequence shows markedly better convergence than pseudorandom simulation, even in a problem of 52 dimensions. Galanti and Jung [11] report extensive comparisons of pseudorandom and quasirandom simulation in financial applications. They observe that quasirandom simulation remains competitive with pseudorandom simulation up to at least 250 dimensions, which would correspond to daily averaging in a one-year option. (This input takes a long time to evaluate.)



■ Selection of Initial Values

The preceding section emphasized that initial values must be selected cautiously. Sobol proposed property A to guide the selection of initial values. To understand property A, consider the following diagram, which shows the unit square divided into four subsquares. The blue points are four successive points in a Sobol sequence (starting at $n = 64$), and the red points are the next four points. In the left-hand graphic, we observe that the blue points belong to different subsquares, as do the red points. This conforms with property A. This is not the case in the graphic on the right, which is constructed from analogous points in a sequence with unit initialization. The second sequence does not satisfy property A.



Generalizing to d dimensions, divide the hypercube $[0, 1)^d$ into 2^d equally sized subcubes and partition a sequence of points in $[0, 1)^d$ into blocks of 2^d points. The sequence satisfies property A if each of the points in any block belongs to a different subcube. An analogous property A' applies when each dimension is divided into quarters. These properties can be verified by evaluating determinants.

Bratley and Fox [8] give initial values for 40 dimensions that are claimed to satisfy property A; Joe and Kuo [4] extend this to 1111 dimensions. However, property A is of limited value in high dimensions, since it is computationally infeasible to use sufficient points to reap an advantage. To benefit from property A in a problem with 250 dimensions would require using $2^{250} = 10^{75}$ points of the sequence, which is larger than the estimated number of particles in the known universe!

Joe and Kuo [5] provide a set of initial values designed to minimize bad projections between pairs of variables. They provide initial values for all polynomials up to degree 18 (21200 dimensions), and claim that they satisfy property A up to 1111 dimensions. The implementation of Lemieux, Cieslak, and Luttmer [12] gives values for 360 dimensions selected on the basis of an optimization. The British–Russian Offshore Development Agency (BRODA)—with which Sobol is affiliated—sells proprietary software to generate sequences with up to 1024 dimensions. Alternatively, Jäckel [6] advocates a randomization procedure to select initial values in higher dimensions.

Starting with Version 6, *Mathematica* includes an option for producing Sobol and Niederreiter sequences as part of its random number generation facility. This comes courtesy of the Intel MKL libraries, which are available for Microsoft Windows (32-bit, 64-bit), Linux x86 (32-bit, 64-bit), and Linux Itanium systems. Specifics of the implementation, such as choice of initial values, are not documented. Evaluation of this implementation in terms of discrepancy, projections, and performance in application remains for future work.

■ Acknowledgments

I gratefully acknowledge the very helpful comments of two anonymous referees.

■ References

[1] E. Siniksaran, "Throwing Buffon's Needle with *Mathematica,*" *The Mathematica Journal*, **11**(1), 2008 pp. 71–90. www.mathematica-journal.com/2009/01/throwing-buffons-needle-with-mathematica.

[2] D. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 2nd ed., Reading, MA: Addison-Wesley, 1981.

[3] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed., New York: Cambridge University Press, 1992.

[4] S. Joe and F. Y. Kuo, "Remark on Algorithm 659: Implementing Sobol's Quasirandom Sequence Generator," *ACM Transactions on Mathematical Software*, **29**(1), 2003 pp. 49–57. doi:10.1145/641876.641879.

[5] S. Joe and F. Y. Kuo, "Constructing Sobol Sequences with Better Two-Dimensional Projections," *SIAM Journal on Scientific Computing*, **30**(5), 2007 pp. 2635–2654. doi:10.1137/070709359.

[6] P. Jäckel, *Monte Carlo Methods in Finance,* Chichester, England: Wiley, 2002.

[7] I. A. Antonov and V. M. Saleev, "An Economic Method of Computing LP_τ Sequences," *USSR Journal of Computational Mathematics and Mathematical Physics*, **19**(1), 1979 pp. 252–256 (English translation). doi:10.1016/0041-5553(79)90085-5.

[8] P. Bratley and B. L. Fox, "Algorithm 659: Implementing Sobol's Quasirandom Sequence Generator," *ACM Transactions on Mathematical Software* **14**(1), 1988 pp. 88–100. doi:10.1145/42288.214372.

[9] W. J. Morokoff and R. E. Caflisch, "Quasi-Random Sequences and Their Discrepancies," *SIAM Journal on Scientific Computing* **15**(6), 1994 pp. 1251–1279. doi:10.1137/0915077.

[10] P. Glasserman, *Monte Carlo Methods in Financial Engineering*, New York: Springer, 2003.

[11] S. Galanti and A. Jung, "Low-Discrepancy Sequences: Monte Carlo Simulation of Option Prices," *Journal of Derivatives* **5**(1), 1997 pp. 63–83. doi:10.3905/jod.1997.407985.

[12] C. Lemieux, M. Cieslak, and K. Luttmer, *RandQMC User's Guide: A Package for Randomized Quasi-Monte Carlo Methods in C*, Technical report 2002-712-15, Department of Computer Science, University of Calgary, 2002. hdl.handle.net/1880/46569.

M. Carter, "A Toolbox for Quasirandom Simulation," *The Mathematica Journal*, 2011. dx.doi.org/doi:10.3888/tmj.13-21.

About the Author

Michael Carter (Ph.D. in economics, Stanford University) has taught in Asia, Europe, and the U.S. His research interests include computational finance, industrial and mathematical economics, and game theory. He has previously published articles in *The Mathematica Journal* on game theory and optimization.

Michael Carter
Adjunct Professor of Finance
Indian Institute of Management, Ahmedabad
carter@iimahd.ernet.in or carter@uni-hohenheim.de