CHAPTER 2

QUASIRANDOM NUMBER GENERATION

Quasirandom numbers are akin to random numbers but exhibit much more regularity. This makes them well-suited for numerical evaluation of multidimensional integrals. This chapter discusses the main types of quasirandom sequences, including Halton, Faure, Sobol', and Korobov sequences. ☞ 1

2.1 MULTIDIMENSIONAL INTEGRATION

Recall that the purpose of a uniform random number generator is to produce an unlimited stream of numbers $U_1, U_2, \ldots$ that behave statistically as independent and uniformly distributed random variables on $(0, 1)$. From such a stream it is easy to construct an infinite sequence of independent and uniformly distributed random *vectors* (points) in $(0,1)^d$, by defining $\mathbf{U}_1 = (U_1, \ldots, U_d)$, $\mathbf{U}_2 = (U_{d+1}, \ldots, U_{2d}), \ldots$. For any real-valued function h on $(0,1)^d$ these random vectors can then be used to approximate the d-dimensional integral

$$\ell = \int_{(0,1)^d} h(\mathbf{u})\, \mathrm{d}\mathbf{u} \tag{2.1}$$

via the sample average

$$\widehat{\ell} = \frac{1}{N} \sum_{i=1}^{N} h(\mathbf{U}_i)\,. \tag{2.2}$$

Handbook of Monte Carlo Methods. By D. P. Kroese, T. Taimre, Z. I. Botev

Precise error bounds on the approximation error can be found through simple statistical procedures; see, for example, Algorithm 8.2. In particular, the standard error $\{\mathbb{E}(\widehat{\ell}-\ell)^2\}^{1/2}$ decreases at a rate $\mathcal{O}(N^{-1/2})$. Hence, asymptotically, to decrease the error by a factor 2, one needs 4 times as many samples. This convergence rate can often be improved by constructing **quasirandom** points $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N$ that fill the unit cube in a much more regular way than is achieved via iid random points. In general, the components of such points can be zero, so we assume from now on that quasirandom points lie in the unit cube $[0,1)^d$ rather than $(0,1)^d$. **Quasi Monte Carlo** methods are Monte Carlo methods in which the ordinary uniform random points are replaced by quasirandom points. Quasirandom points are no longer independent, but do have a high degree of uniformity, which is often expressed in terms of their discrepancy (first introduced by Roth [27]). Specifically, let $\mathscr{C}$ be a collection of subsets of $[0,1)^d$ and $\mathcal{P}_N = \{\mathbf{u}_1, \ldots, \mathbf{u}_N\}$ a set of points in $[0,1)^d$. The **discrepancy** of $\mathcal{P}_N$ relative to $\mathscr{C}$ is defined as

☞ 306

$$D_{\mathscr{C}}(\mathcal{P}_N) = \sup_{C\in\mathscr{C}} \left| \frac{1}{N}\sum_{i=1}^{N} \mathrm{I}_{\{\mathbf{u}_i\in C\}} - \int \mathrm{I}_{\{\mathbf{u}\in C\}}\,\mathrm{d}\mathbf{u} \right| . \tag{2.3}$$

Special cases are the **ordinary discrepancy**, where $\mathscr{C}$ is the collection of rectangles $[a_1,b_1)\times\cdots\times[a_d,b_d)$, and the **star discrepancy**, where $\mathscr{C}$ is the collection of rectangles $[0,b_1)\times\cdots\times[0,b_d)$.

The sum in (2.3) is simply the number of points in C, whereas the integral is the d-dimensional volume of C. The integration error for all indicator functions $\mathrm{I}_{\{\mathbf{u}\in C\}}$, $C\in\mathscr{C}$ is thus bounded by the discrepancy of the point set. Similarly, the **Koksma–Hlawka inequality** provides, for a suitable class of functions h in (2.1) and (2.2), a bound $|\widehat{\ell}-\ell| \leqslant D^* k_h$ on the integration error, where D^* is the star discrepancy and k_h is a constant that depends only on the function h; see [24, Page 19]. Discrepancy measures are therefore useful tools for studying convergence rates for multidimensional integration. Note that the star discrepancy may be viewed as the d-dimensional generalization of the Kolmogorov–Smirnov test statistic.

☞ 336

EXAMPLE 2.1 (Regular Grid)

☞ 12

Consider the d-dimensional lattice $\mathbb{Z}^d N^{-1/d}$, where we assume that $N = m^d$ for some strictly positive integer m. By intersecting the lattice with the hypercube $[0,1)^d$ we obtain a regular grid of N points on $[0,1)^d$. The ordinary and star discrepancy for this point set are both $1-(1-m^{-1})^d$, which is of the order $\mathcal{O}(m^{-1}) = \mathcal{O}(N^{-1/d})$.

To see this, take in (2.3) the "worst-case" set $C = [0, 1-m^{-1}]^d = [0, 1-m^{-1}+\varepsilon)^d$ for some infinitesimally small $\varepsilon > 0$. The number of grid points in C is N, while its volume is $(1-m^{-1})^d$. It follows that the integration error for the indicator $\mathrm{I}_{\{\mathbf{u}\in C\}}$ is $1-(1-m^{-1})^d$.

The above example indicates that for $d > 2$ integration with a regular grid is inferior to ordinary Monte Carlo integration. However, it is possible to construct infinite sequences $\mathbf{u}_1, \mathbf{u}_2, \ldots$ of points in $[0,1)^d$ so that any point set $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N\}$ has star discrepancy

$$D^*(\{\mathbf{u}_1, \ldots, \mathbf{u}_N\}) \approx c_d \frac{(\ln N)^d}{N} . \tag{2.4}$$

Note that this is close to $\mathcal{O}(N^{-1})$ for fixed d. Using such **low-discrepancy** sequences instead of ordinary random numbers therefore has the potential of significantly improving the accuracy of the integration.

There are two main classes of low-discrepancy sequences: those based on *van der Corput sequences*, such as the Halton, Faure, and Sobol' point sets; and those based on *lattice methods*, such as the Korobov lattice rule. These are discussed in the sections that follow.

2.2 VAN DER CORPUT AND DIGITAL SEQUENCES

Let $b \geqslant 2$ be an integer. Any number $k \in \mathbb{N}$ admits a **b-ary expansion** of the form

$$k = \sum_{i=1}^{r} a_i \, b^{i-1} = a_1 + a_2 b + \cdots + a_r b^{r-1} \,,$$

for some finite r and **digits** $a_1, \ldots, a_r \in \{0, \ldots, b-1\}$. The corresponding b-ary representation of k is written as $(a_r \ldots a_1)_b$, or simply $a_r \ldots a_1$ if the **base** or **radix** b is implicitly understood. For example, the number 12345 in decimal representation ($b = 10$) has binary representation 11000000111001_2 and ternary representation 121221020_3.

Many low-discrepancy sequences are based on the following transformation of natural numbers. Let $k \in \mathbb{N}$ have b-ary representation $a_r \ldots a_2 a_1$. The base-b **radical inverse** of k is the number (in [0,1))

$$\sum_{i=1}^{r} a_i \, b^{-i} = a_1 b^{-1} + a_2 b^{-2} + \cdots + a_r b^{-r} \,,$$

written in b-ary form as $0.a_1 a_2 \ldots a_r$. The radical inverse transformation thus simply reverses the order of the digits and puts a b-ary point in front to obtain a number in the interval $[0, 1)$. As an example, for $b = 2$ the radical inverse of $880 = 1101110000_2$ is $0.0000111011_2 = \frac{59}{2^{10}}$.

The **base-b van der Corput sequence** is the sequence obtained by applying the base-b radical inverse to the numbers $0, 1, 2, \ldots$. The main significance of van der Corput sequences for quasi Monte Carlo is that for each base b the sequence constitutes a one-dimensional low-discrepancy sequence. Another important property of a base-b van der Corput sequence $x_1, x_2, \ldots$ is that it is linearly increasing in subsequent segments of length b; that is, $x_1 < x_2 < \cdots < x_b, \quad x_{b+1} < x_{b+2} < \cdots < x_{2b}$ and so on, with the increments within each segment equal to $1/b$.

■ EXAMPLE 2.2 (Van der Corput Sequence Generation)

The following MATLAB function `vdc.m` calculates the first N elements of the base-b van der Corput sequence. At each iteration the next b-ary representation is determined by the function `nbe.m`. For example, for the binary ($b = 2$) case and starting with 0, subsequent calls to `nbe.m` give the sequence of binary numbers $0, 1, 10, 11, 100, 101, \ldots$ (stored as row vectors). To each of these numbers a reversal of the digits is applied. Finally, each of these "flipped" b-ary numbers is translated back to an ordinary (that is, decimal) representation. The first ten elements of the base-2 van der Corput sequence are thus: $0, 0.5, 0.25, 0.75, 0.125, 0.625, 0.375, 0.875, 0.0625, 0.5625$.

Note that this sequence can be divided into segments of length 2 in which the points are increasing with increment $1/2$.

```
function out = vdc(b,N)
out = zeros(N,1);
numd = ceil(log(N)/log(b)); %maximal number of digits required
bb = 1./b.^(1:numd);
a = [];
out(1)=0;
for i=2:N
    a = nbe(a,b); %next b-ary expansion
    fa = fliplr(a); %flip the digits
    out(i) = sum(fa.*bb(1:numel(a)));
end
```

```
function na = nbe(a,b)
numd = numel(a);
na = a;
carry = true;
for i=numd:-1:1
    if carry
        if a(i) == b-1
            na(i) = 0;
        else
            na(i) = a(i) + 1;
            carry = false;
        end
    end
end
if carry
    na = [1,na];
end
```

A base-b van der Corput sequence can thus be thought of in the following way:

1. *Construct a sequence of vectors representing the reverse b-ary representation of the numbers 0,1,2,...:*

$$\begin{pmatrix}0\\0\\0\\\vdots\end{pmatrix}, \begin{pmatrix}1\\0\\0\\\vdots\end{pmatrix}, \ldots, \begin{pmatrix}b-1\\0\\0\\\vdots\end{pmatrix}, \begin{pmatrix}0\\1\\0\\\vdots\end{pmatrix}, \begin{pmatrix}1\\1\\0\\\vdots\end{pmatrix}, \ldots, \begin{pmatrix}b-1\\1\\0\\\vdots\end{pmatrix}, \begin{pmatrix}0\\2\\0\\\vdots\end{pmatrix}, \ldots \tag{2.5}$$

2. *Premultiply each vector with the row vector* $(b^{-1}, b^{-2}, \ldots)$ *to obtain a number in* $[0, 1)$.

Digital sequences are low-discrepancy sequences in d dimensions constructed from van der Corput sequences with bases $b_1, \ldots, b_d$ for each of the components. Their general construction is as follows.

Algorithm 2.1 (Digital Sequence) *For each component $k = 1, \ldots, d$, execute the following steps.*

1. *Construct the reverse b_k-ary representation vectors of the numbers 0,1,2,..., as in* (2.5).
2. *Premultiply each such vector with some fixed* **generator matrix** G_k.
3. *Premultiply the resulting vectors with the row vector $(b^{-1}, b^{-2}, \ldots)$ to obtain the k-th coordinate of the digital sequence.*

The generator matrices are usually taken to be right-triangular and such that each $r \times r$ submatrix is non-singular. Notice that a van der Corput sequence is a one-dimensional digital sequence where the generator matrix is the identity matrix.

2.3 HALTON SEQUENCES

The simplest way to construct low-discrepancy sequences in d dimensions is by taking van der Corput sequences with different bases $b_1, \ldots, b_d$ for each of the components. The bases must be pairwise relatively prime; that is, each pair does not share a common factor greater than 1. It is customary to take $b_1, \ldots, b_d$ equal to the first b prime numbers greater than 1. In the framework of Algorithm 2.1, the Halton sequence is a digital sequence in which the generator matrices for the components are simply identity matrices. More precisely, let $b_1, \ldots, b_d \geqslant 2$ be relative prime integers. Let $u_{1j}, u_{2j}, \ldots$ be the base-b_j van der Corput sequence, $j = 1, \ldots, d$. The sequence of d-dimensional points $\mathbf{u}_1 = (u_{11}, \ldots, u_{1d}), \mathbf{u}_2 = (u_{21}, \ldots, u_{2d}), \ldots$ is called the **Halton sequence** corresponding to $b_1, \ldots, b_d \geqslant 2$.

Let $\mathbf{u}_1, \mathbf{u}_2, \ldots$ be a Halton sequence. The set $\{\mathbf{u}_1, \ldots, \mathbf{u}_N\}$ formed by the first N elements of the Halton sequence is the corresponding **Halton point set.** Halton [10] showed that the star discrepancy of such sets is of the order $\mathcal{O}((\ln N)^d/N)$. Hammersley [11] modified the Halton set by defining points $\mathbf{u}_1^* = (u_{11}^*, \ldots, u_{1d}^*), \ldots, \mathbf{u}_N^* = (u_{N1}^*, \ldots, u_{Nd}^*)$ with $u_{i1}^* = (i-1)/N$ and $u_{ij}^* = u_{i,(j-1)}, j = 2, \ldots, d$, $i = 1, \ldots, N$. The point set thus obtained, called a **Hammersley set**, has a discrepancy of $\mathcal{O}((\ln N)^{d-1}/N)$.

■ EXAMPLE 2.3 (Halton and Hammersley Points)

The following two MATLAB programs produce Halton and Hammersley point sets of size N for a given vector of bases $\mathbf{b} = (b_1, \ldots, b_d)$. The difference in uniformity between the two point sets is illustrated in Figure 2.1 for the two-dimensional case with bases $b_1 = 2$ and $b_2 = 3$.

```
%halton.m
function out = halton(b,N);
```

```
dim = numel(b);
out = zeros(N,dim);
for i=1:dim
    out(:,i) = vdc(b(i),N);
end
```

```
function out = hammersley(b,N)
dim = numel(b);
out = zeros(N,dim);
out(2:N,2:dim) = halton(b(1:dim-1),N-1);
out(:,1) = [0:N-1]/N;
```

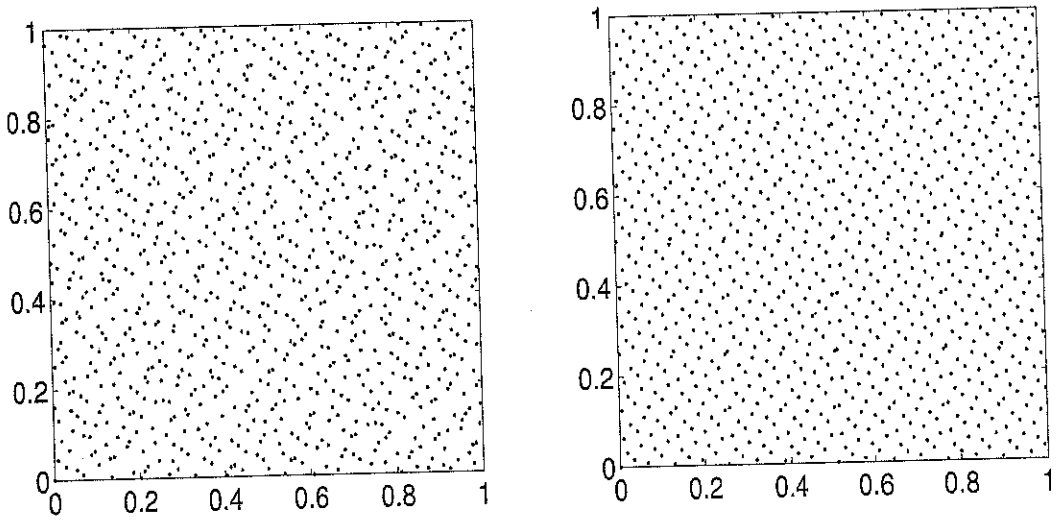

Figure 2.1 The Halton (left) and Hammersley (right) point sets of size $N = 1000$, with bases 2 and 3.

Halton sequences are particularly useful for quasi Monte Carlo integration when d is relatively small. For large d, say $d > 20$, Halton sequences are less effective, because they do not fill the unit hypercube in a uniform way. The reason is that the corresponding van der Corput sequences with large bases produce long linearly increasing segments, as noted in Section 2.2. The situation is illustrated in the following example.

■ EXAMPLE 2.4 (Poor Space-Filling of a Halton Sequence)

Consider a Halton sequence in dimension $d = 40$, where the bases of the van der Corput sequences are chosen to be the first 40 primes. The van der Corput sequences corresponding to the last two coordinates thus form a two-dimensional Halton sequence $(x_1, y_1), (x_2, y_2), \ldots$ with bases 167 and 173. But this sequence does not fill the unit square in an even fashion, as shown in Figure 2.2, where the

first 1000 and 6000 points are plotted. The van der Corput sequences $x_1, x_2, \ldots$, and $y_1, y_2, \ldots$ follow the recursion $x_n = x_{n-1} + 1/167$ and $y_n = y_{n-1} + 1/173$, in each subsequent segment of 167 and 173 points. This results in the unit square being filled in a highly linear fashion.

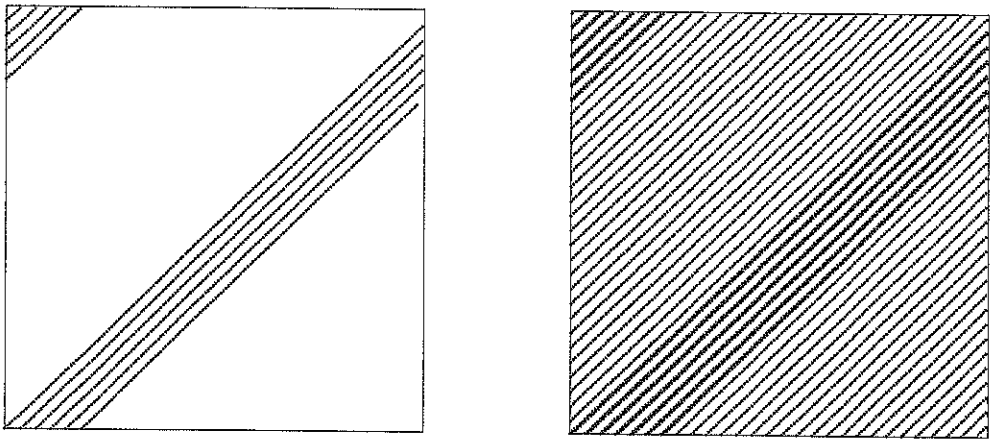

Figure 2.2 Point sets of a two-dimensional Halton sequence with bases 167 and 171 with $N = 1000$ (left) and $N = 6000$ (right) points.

2.4 FAURE SEQUENCES

Faure [5] developed low-discrepancy sequences that, like the sequences of Halton, are based on van der Corput sequences. But, unlike Halton sequences, Faure sequences have a *common* base. This base needs to be a prime number at least as large as the dimension d.

The construction of a d-dimensional Faure sequence is based on d permutations (one for each component) of a base-b van der Corput sequence. More precisely, the Faure sequence is a digital sequence in which the generator matrix for component $k = 1, \ldots, d$ is given by G^{k-1}, where G is the upper-triangular **Pascal matrix** in base b; that is, the matrix with (i, j)-th element

$$G_{ij} = \binom{j-1}{i-1} \mod b, \quad i = 1, \ldots, j, \quad j = 1, 2, \ldots .$$

For each $r = 0, 1, 2, \ldots$ and $k = 1, \ldots, d$ the submatrix formed by the first b^r rows and columns of G^{k-1} is invertible. As a consequence, premultiplication by G^{k-1} in Step 2 of Algorithm 2.1 leads to a permutation of the first b^r vectors obtained in Step 1, and results (in Step 3) in a permutation of the first b^r numbers in the original van der Corput sequence. To obtain a sequence of length N it suffices to truncate G to the r-th row and column with $r = \lfloor \ln N / \ln b \rfloor + 1$, as no more than r digits will be needed. This leads to the following algorithm.

Algorithm 2.2 (Faure Sequence With Dimension d and Base b)

1. *Set* $r = \lfloor \ln N / \ln b \rfloor + 1$. *Define* $\mathbf{b} = (b^{-1}, b^{-2}, \ldots, b^{-r})$ *and let* $n = 0$.
2. *Calculate the* b*-ary representation* $(a_r \ldots a_1)_b$ *of* n. *Store it in the vector* $\mathbf{a} = (a_1, a_2, \ldots, a_r)^\top$.
3. *For* $k = 1, \ldots, d$ *compute* $\widetilde{\mathbf{a}}_k = C^{k-1}\mathbf{a}$.
4. *For* $k = 1, \ldots, d$ *compute* $x_{nk} = \mathbf{b}\,\widetilde{\mathbf{a}}_k$ *and let* $\mathbf{x}_n = (x_{n1}, \ldots, x_{nd})^\top$ *be the* n*-th Faure point.*
5. *If* $n = N$ *stop; otherwise, set* $n = n + 1$ *and go to Step 2.*

In contrast to Halton sequences, the discrepancy of a Faure sequence does not deteriorate when d grows large. Although both sequences satisfy the low discrepancy growth (2.4), the constant of proportionality c_d grows rapidly in d for the Halton sequence, whereas it decreases to 0 for the Faure sequence; see [5]. Even though the Faure sequence has better discrepancy and uniformity properties than the Halton sequence, for small N and large d (and hence large b) it can still exhibit poor space-filling behavior similar to the Halton sequence illustrated in Figure 2.2.

The amount of uniformity in a Faure sequence is further demonstrated by the fact that certain subsets of points form a $(0, m, d)$-net. More precisely, a set of b^m points in $[0,1)^d$ is said to be **(t, m, d)-net** in base b if every elementary rectangle of volume b^{t-m} contains exactly b^t points. Here an **elementary rectangle** means a d-dimensional rectangle of the form

$$\prod_{i=1}^{d} \left[\frac{a_i}{b^{k_i}}, \frac{a_i + 1}{b^{k_i}} \right),$$

with $a_i \in \{0, \ldots, b-1\}$ and $k_i \in \{1, 2, \ldots\}$ for $i = 1, \ldots, n$. The volume of this elementary rectangle is $1/\prod_{i=1}^{d} b^{k_i}$. In terms of (2.3), the discrepancy of a (t, m, d)-net relative to each elementary rectangle of volume b^{t-m} is 0. Note that a (t, m, d)-net is automatically a $(t+1, m, d)$-net.

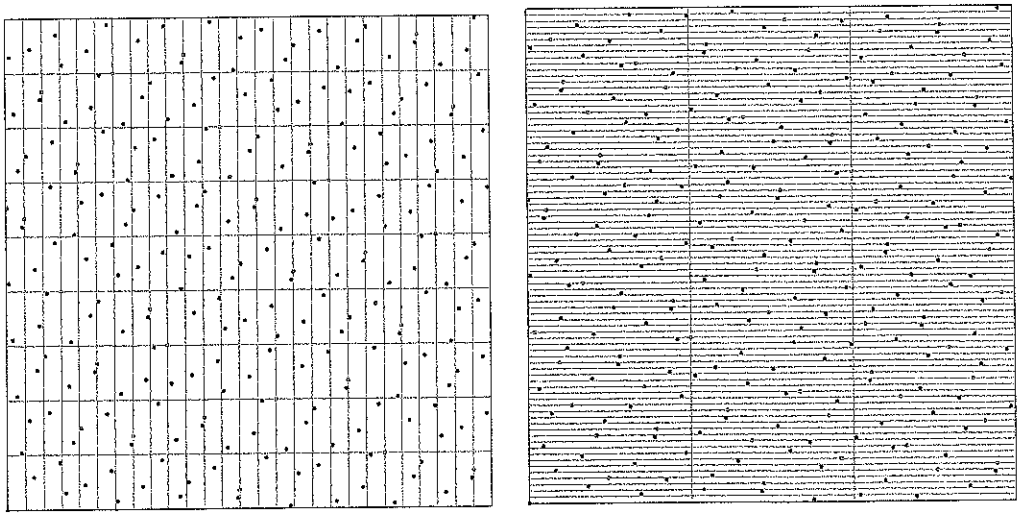

Figure 2.3 The first 3^5 points of a two-dimensional Faure sequence with base $b = 3$ are plotted in each unit square. Both squares are divided into elementary boxes of volume $1/b^5$. Each such box has exactly one point, because the Faure points form a $(0, 5, 2)$-net.

The first b^m points of a d-dimensional Faure sequence form a $(0, m, d)$-net. As an illustration, Figure 2.3 shows various elementary boxes of volume $1/b^m$, for the case where $b = 3$, $m = 5$, and $d = 2$. Each elementary box of volume $1/b^m$ contains exactly one of the $b^m = 243$ points.

■ EXAMPLE 2.5 (Faure Sequence Generation)

The following MATLAB function produces a d-dimensional Faure point set of size N, using a base b. The latter must be chosen to be prime and greater than or equal to d. Note that for convenience we work below with the transpose of the generator matrix G and the vector $\mathbf{a}$.

```
function p = faure(b,d,N)
r = floor(log(N)/log(b))+1; %largest number of digits
bb=repmat(1./b.^(1:r),N+1,1); %rows (1/b, 1/b^2,...)
p = zeros(N+1,d);
G = zeros(r,r);
for j=1:r
    for i=1:j
        G(i,j) = mod(nchoosek(j-1,i-1),b);
    end
end
G=G';
a=repmat((0:N)',1,r);
for i=1:r-1
    a(:,i)=mod(a(:,i),b);
    a(:,(i+1):r)=floor(a(:,(i+1):r)/b);
    % a now contains the b-ary expansion of 0:N
end
p(:,1)=sum(bb.*a,2);
for k=2:d
    a=mod(a*G,b); %permuted b-ary expansion of 0:N
    p(:,k)=sum(bb.*a,2);
end
```

2.5 SOBOL' SEQUENCES

A d-dimensional Sobol' sequence [29] is a digital sequence (see Algorithm 2.1) where each component has the *same* base 2, and where the $r \times r$ generator matrices (r is the maximal number of digits, that is, $r = \lfloor \ln N / \ln b \rfloor + 1$ if N is the total number of points required) are chosen in the following way. Each generator matrix G (a different one for each component) is defined by r **direction numbers**, $g_1, \ldots, g_r$, whose binary representations form the columns of the generator matrix. The j-th direction number is of the form

$$g_j = \frac{m_j}{2^j}, \quad j = 1, \ldots, r\,,$$

in which $m_1, m_2, \ldots, m_r$ satisfy the recursion ($\oplus$ denotes binary addition via the XOR operation)

$$m_i = \left(\bigoplus_{j=1}^{q} 2^j \, c_j \, m_{i-j} \right) \oplus m_{i-q} \, , \tag{2.6}$$

where the $\{c_i\}$ are the (binary) coefficients and q is the degree of a primitive polynomial in binary arithmetic

$$x^q + c_1 \, x^{q-1} + \cdots + c_{q-1} \, x + 1 \, .$$

Each such polynomial can be represented by a **polynomial number**, whose binary representation corresponds to the coefficients. For example, the polynomial number $37 = 100101_2$ corresponds to the primitive polynomial $x^5 + x^2 + 1$. Each generator matrix is thus completely specified by

1. a primitive polynomial, or the corresponding polynomial number, and
2. the initial values $m_1, \ldots, m_q$ for the recursion (2.6).

Lists of primitive polynomial numbers and starting values may be found in [15]; see also the MATLAB implementation in Example 2.6 below. It is standard to take the d lowest polynomial numbers for a d-dimensional problem, starting with the 0-degree polynomial 1, so that the first component is generated according to the base-2 van der Corput sequence. Because Sobol' sequences use binary arithmetic they can be implemented very efficiently on a computer. A benchmark implementation may be found in [1]; see also [16].

Similar to the Faure sequence, the first 2^m points of a d-dimensional Sobol' sequence form a (t, m, d)-net in base 2, for some $t \leqslant 1 - d + \sum_{i=1}^{d} q_i$, where the $\{q_i\}$ are the degrees of the primitive polynomials for the first d components [29]. For example, for a two-dimensional Sobol' sequence $q_1 = 0$ and $q_2 = 1$, so that $t = 0$. Figure 2.4 illustrates that the corresponding Sobol' point set of size $N = 2^{10} = 1024$ forms a $(0, 10, 2)$-net.

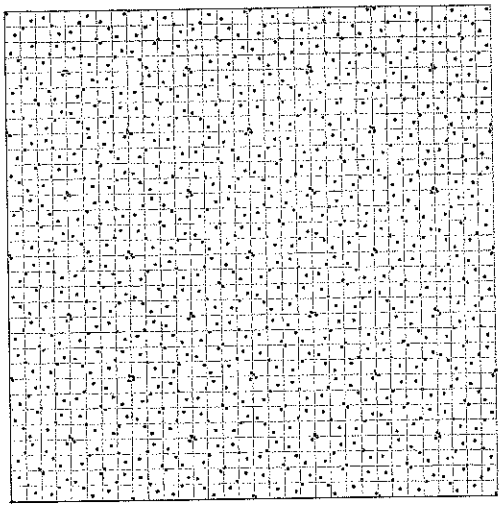

Figure 2.4 The first 2^{10} points of a two-dimensional Sobol' sequence form a $(0, 10, 2)$-net: each elementary box of volume $1/2^{10}$ has exactly one point.

EXAMPLE 2.6 (Sobol' Generator)

The following MATLAB function `sobol.m` produces d-dimensional Sobol' point sets of size N. The generator matrices are computed via the function `sobmat.m`. This function in turn uses the function `cbe.m` which returns the b-ary representation of a number.

```
function p = sobol(d,N)
b=2; %always base 2
r =  floor(log(N)/log(b))+1;
bb = 1./b.^(1:r);
bbb = repmat(bb,N+1,1);
p = zeros(N+1,d);
G=zeros(r,r);
GG=sobmat(d,r);
a=repmat((0:N)',1,r);
for i=1:r
    a(:,i) = mod(a(:,i),b);
    a(:,(i+1):r) = floor(a(:,(i+1):end)./b);
    % a contains now the b-ary expansion of 0:N
end
for i=1:d
    G(:,:)=GG(:,:,i);
    p(:,i) = sum(bbb.*mod(a*G',b),2);
end
```

```
function G = sobmat(d,r)
polys=[1,3,7,11,13,19,25,37,59,47,61,55,41,67,97,91,109,103,...
      115,131,193,137,145,143,241,157,185,167,229,171,213,191,...
      253,203,211,239,247,285,369,299,425,301,361,333,357,351,...
      501,355,397,391,451,463,487];
ivals=[1 1 1 1 1 1 1 1;          %1
       1 3 5 15 17 51 85 255;    %2
       1 1 7 11 13 61 67 79;     %3
       1 3 7 5 7 43 49 147;      %4
       1 1 5 3 15 51 125 141;    %5
       1 3 1 1 9 59 25 89;       %6
       1 1 3 7 31 47 109 173;    %7
       1 3 3 9 9 57 43 43;       %8
       1 3 7 13 3 35 89 9;       %9
       1 1 5 11 27 53 69 25;     %10
       1 3 5 1 15 19 113 115;    %11
       1 1 7 3 29 51 47 97;      %12
       1 3 7 7 21 61 55 19;      %13
       1 1 1 9 23 39 97 97;      %14
       1 3 3 5 19 33 3 197;      %15
       1 1 3 13 11 7 37 101;     %16
```

```
      1 1 7 13 25 5 83 255;  %17
      1 3 5 11 7 11 103 29;  %18
      1 1 1 3 13 39 27 203;  %19
      1 3 1 15 17 63 13 65]; %20

m = zeros(1,r);
G = zeros(r,r,d);
G(:,:,1) = eye(r);

for k=2:d
    ppn=polys(k); %polynomial number
    m=ivals(k,:); %initial values
    c= cbe(ppn,2);
    c = c(2:end); % coefficients of the primitive polynomial
    deg = numel(c); %degree of the polynomial
    for i=9:r %first 8 values already given
        s = 0;
        for j = 1:deg
            s = bitxor(s,2^j*c(j)*m(i-j));
        end
    m(i)= bitxor(s,m(i-deg));
    end
    for j=1:r
        h = cbe(m(j),2); % binary representation
        numdigs = numel(h);
        G(j- numdigs+1:j,j,k)= h';
    end
end
```

```
function a = cbe(k,b) % coefficients of base-b expansion of k
numd = max(0,floor(log(k)/log(b))) + 1; %number of digits
a = zeros(1,numd);
q = b^(numd-1);
for i = 1:numd
    a(i) = floor(k/q);
    k = k - q*a(i);
    q = q/b;
end
```

2.6 LATTICE METHODS

Not all quasirandom point sets are constructed via van der Corput sequences. An important alternative employs the theory of lattices. The most common construction is Korobov's **method of good lattice points** [17], which defines a set of N points in $[0,1)^d$ of the form

$$\left\{ \frac{i}{N}(1, a, a^2, \ldots, a^{d-1}) \bmod 1, \quad i = 0, \ldots, N-1 \right\}, \tag{2.7}$$

depending on the choice of an integer a. The points can be related to a *linear congruential generator* (LCG) with modulus N, multiplier a, and increment $c = 0$: ☞ 4

$$x_t = a\,x_{t-1} \bmod N, \quad t = 1, 2 \ldots, N-1\,. \tag{2.8}$$

Namely, let $\mathcal{P}_{x_0} = \{(x_t, x_{t+1}, \ldots, x_{t+d-1})/N,\ t = 0, 1, \ldots\}$ be the point set of d successive values of the LCG (divided by N), starting from some x_0. Then, the union $\cup_{x_0 \in \{0,1,\ldots,N-1\}} \mathcal{P}_{x_0}$ coincides with the Korobov point set (2.7). A consequence of this viewpoint is that the coordinates of the Korobov point set can also be generated "on the fly", without specifying the dimension in advance; see also [22, Page 205].

By replacing the vector $(1, a, \ldots, a^{d-1})$ in (2.7) with a general integer vector $\mathbf{v}$ or with linear combinations of r such vectors, one obtains **rank-1** and **rank-r lattice rules**, respectively [28]. Another class of lattice rules is defined through more algebraic means, using polynomial rings and formal Laurent series. Details and references on such **polynomial lattices** may be found in [3] and [22].

The selection of good multipliers and generating vectors is discussed, for example, in [4, 12, 20]. To obtain a full period of length $N - 1$ in each coordinate for any a with the Korobov lattice, N should be chosen to be a prime number. If N is a power of 2 instead, a should be an odd number to ensure a full period.

One drawback of the standard Korobov lattice is that the point set is fixed. The introduction of **extensible** lattice rules [13] alleviates this difficulty. The idea, for the Korobov lattice, is to replace (2.7) with the infinite set

$$\left\{\psi_b(i)(1, a, a^2, \ldots, a^{d-1}) \mod 1, \quad i = 0, 1, 2, \ldots\right\}\,, \tag{2.9}$$

where $\{\psi_b(k)\}$ is the van der Corput sequence with base b. Hickernell et al. [13] recommend the choice $b = 2$ and $a = 17797$ or $a = 1267$; see also [8].

■ EXAMPLE 2.7 (Korobov Lattice Generation)

The following MATLAB functions implement the Korobov and extensible Korobov lattice rules, using (2.7) rather than the recursion (2.8). The last function uses the van der Corput function vdc in Example 2.2 and assumes by default a base $b = 2$. Figure 2.5 shows the 10-th and 21-st coordinates of a 30-dimensional Korobov point set of size $N = 2^{10}$ with $a = 17797$.

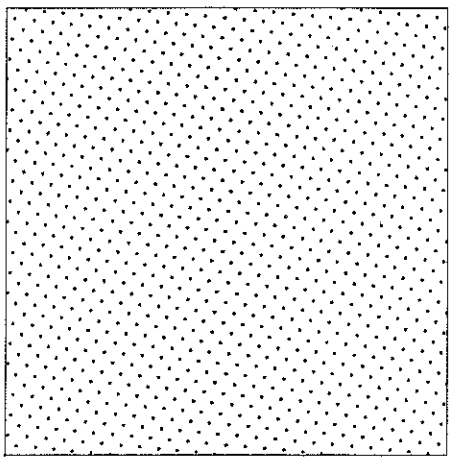

Figure 2.5 The projections onto coordinates 10 and 21 of a 30-dimensional Korobov lattice.

```
function P = korobov(a,d,N)
z(1) = 1;
for i=2:d
    z(i) = mod(z(i-1)*a,N);
end
Z = repmat(z,N,1);
B = repmat((1:N)',1,d);
P = mod(B.*Z/N, 1);
```

```
function P = extkorobov(a,d,N)
b = 2;
z(1) = 1;
for i=2:d
    z(i) = mod(z(i-1)*a,N);
end
Z = repmat(z,N,1);
v = vdc(b,N);
B = repmat(v,1,d);
P = mod(B.*Z, 1);
```

2.7 RANDOMIZATION AND SCRAMBLING

One of the appealing features of ordinary Monte Carlo integration is that an assessment of the error in the sample average approximation (2.2) of the integral (2.1) is readily available in the form of standard errors and confidence intervals. For quasi Monte Carlo integration this is no longer the case, as the points, $\{\mathbf{u}_i\}$ say, are deterministic and not $\mathsf{U}[0,1)^d$ distributed.

☞ 301

However, the situation can be remedied by simply adding a fixed random vector $\mathbf{Z} \sim \mathsf{U}[0,1)^d$ to each point and then taking the fractional part of the resulting point. It is easy to see that each point $\widetilde{\mathbf{U}}_i = (\mathbf{u}_i + \mathbf{Z}) \mod 1$ is $\mathsf{U}[0,1)^d$ distributed. This procedure is called **random shifting** and was first proposed by Cranley and Patterson [2]. Using a random shift renders the quasi Monte Carlo approximation

$$\widetilde{\ell} = \frac{1}{N} \sum_{i=1}^{N} h(\widetilde{\mathbf{U}}_i) \tag{2.10}$$

a random variable with expectation ℓ in (2.1). By repeating the quasi Monte Carlo procedure independently with K different shift vectors one obtains K independent copies of $\widetilde{\ell}$, to which one can apply the standard statistical techniques for evaluating confidence intervals and standard error, as described in Algorithm 8.2. See Algorithm 9.11 for a more detailed description.

☞ 306
☞ 376

For digital sequences a random shift can also be applied directly to the digits. Specifically, suppose $\mathbf{a}_k = (a_{k1}, a_{k2}, \ldots)^\top$ is the infinite-dimensional vector that

corresponds to the b-ary expansion of the k-th coordinate of a point $\mathbf{u}$; thus, the k-th coordinate of $\mathbf{u}$ is given by

$$u_k = \sum_{i=1}^{\infty} a_{ki}\, b^{-i} .$$

Let $\mathbf{W} = (W_1, W_2, \ldots)^\top$ be an infinite-dimensional random vector in which the $\{W_i\}$ are independent and discrete uniformly distributed on $\{0, 1, \ldots, b-1\}$. In other words, $\mathbf{W}$ is the vector representing the b-ary expansion of a U[0, 1)-distributed random number. Next, let $\mathbf{W}_1, \ldots, \mathbf{W}_d$ be independent copies of $\mathbf{W}$. By adding $\mathbf{W}_k$ to $\mathbf{a}_k$ modulo b, for $k = 1, \ldots, d$, one obtains vectors $\widetilde{\mathbf{a}}_1, \ldots, \widetilde{\mathbf{a}}_d$ representing the b-ary expansion of $(\mathbf{u} + \mathbf{Z}) \bmod 1$, where $\mathbf{Z} \sim \mathrm{U}[0,1)^d$. By adding the same $\{\mathbf{W}_k\}$ to all the points in the quasi Monte Carlo point set, this **digital shift** procedure yields a point set that has exactly the same distribution as one obtained using the original random shift method.

Digital sequences such as the Halton, Faure, and Sobol' sequences are sometimes "shuffled" with the aim of improving their uniformity and convergence properties. A general procedure, called **nested permutation scrambling**, introduced by Owen (see, for example, [25] and [26]) is to permute the digits in the b_k-ary expansion between Steps 2 and 3 of Algorithm 2.1 for each component $k = 1, \ldots, d$. This can be done in a deterministic or random way. A convenient subset of such procedures is obtained by premultiplying the digital vectors obtained after Step 2 in Algorithm 2.1 with a random lower-triangular matrix L_k with elements in $\{0, 1, \ldots, b_k\}$, for each dimension $k = 1, \ldots, d$. There exist several variants of this procedure (see [23] and [22, Page 207]), but the most common approach is to choose the lower off-diagonal elements of L_k independently and uniformly from $\{0, 1, \ldots, b_k\}$ and the diagonal elements independently and uniformly from $\{1, \ldots, b_k\}$. This leads to the following modification of Algorithm 2.1. We assume for simplicity that the bases for the d components are all equal to b. All matrix operations are carried out modulo b.

Algorithm 2.3 (Uniform Linear Scrambling) *Let $r = \lfloor \ln N / \ln b \rfloor + 1$. For each component $k = 1, \ldots, d$, execute the following steps.*

1. *Create a random $r \times r$ matrix L_k as follows:*
   (a) *For $i = 1, \ldots, r$ and $j = 1, \ldots, i-1$ draw the (i, j)-th element of L_k uniformly and independently from $\{0, 1, \ldots, b\}$.*
   (b) *For $i = 1, \ldots, r$ draw the (i, i)-th element of L_k uniformly and independently from $\{1, \ldots, b\}$.*
   (c) *Set the remaining elements to 0.*
2. *Construct the reverse b_k-ary representation vectors of the numbers $0, 1, \ldots, N$ as in (2.5).*
3. *Premultiply each such vector with some fixed* generator matrix *G_k.*
4. *Premultiply the resulting vectors with L_k.*
5. *Premultiply the resulting vectors with the row vector $(b^{-1}, b^{-2}, \ldots)$ to obtain the k-th coordinates of the digital sequence $\mathbf{u}_1, \ldots, \mathbf{u}_N$.*

Finally, generate $\mathbf{Z} \sim \mathrm{U}[0,1)^d$ and return $\{(\mathbf{u}_i + \mathbf{Z}) \bmod 1, i = 1, \ldots, N\}$ as the scrambled set.

Further Reading

A concise introduction to quasi Monte Carlo methods is given in Glasserman [9, Chapter 5], whereas more comprehensive treatments may be found in Niederreiter [24] and Lemieux [22]. An extensive state-of-the-art survey on the theory of quasi Monte Carlo and randomized quasi Monte Carlo is given in [18]. Fox [7] focuses on using quasi Monte Carlo techniques in practice, and Jäckel [15] provides a useful resource for applications in finance. Randomized quasi Monte Carlo is surveyed in [21], and various useful scrambling procedures are discussed in [6, 14, 20, 23]. Efficient methods to implement scrambled digital sequences are discussed in Hong and Hickernell [14]. Related scrambling procedures can be found in [6, 21, 23]. For a collection of research papers on quasi Monte Carlo see [19].

REFERENCES

1. P. Bratley and B. L. Fox. Algorithm 659: Implementing Sobol's quasirandom sequence generator. *ACM Transactions on Mathematical Software*, 14(1):88–100, 1988.
2. R. Cranley and T. N. L. Patterson. Randomisation of number theoretic methods for multiple integration. *SIAM Journal on Numerical Analalysis*, 13(6):904–914, 1976.
3. J. Dick, F. Y. Kuo, F. Pillichshammer, and I. H. Sloan. Construction algorithms for polynomial lattice rules for multivariate integration. *Mathematics of Computation*, 74(252):1895–1921, 2005.
4. K.-T. Fang and Y. Wang. *Number-Theoretic Methods in Statistics*. Chapman & Hall, London, 1994.
5. H. Faure. Discrépance de suites associées à un système de numération. *Comptes Rendus Mathematique*, 286-A:293–296, 1978.
6. H. Faure and S. Tezuka. Another random scrambing of digital (t, s)-sequences. In K. T. Fang, F. J. Hickernell, and H. Niederreiter, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2000*, pages 242–256. Springer-Verlag, Berlin, 2002.
7. B. L. Fox. *Strategies for Quasi-Monte Carlo*. Kluwer Academic Publishers, Norwell, MA, 1999.
8. H. S. Gill and C. Lemieux. Searching for extensible Korobov rules. *Journal of Complexity*, 23(4-6):603–613, 2007.
9. P. Glasserman. *Monte Carlo Methods in Financial Engineering*. Springer-Verlag, New York, 2004.
10. J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integral. *Numerische Mathematik*, 2(1):84–90, 1960.
11. J. M. Hammersley. Monte Carlo methods for solving multivariable problems. *Annals of the New York Academy of Sciences*, 86(3):844–874, 1960.
12. P. Hellekalek. On the assessment of random and quasi-random point sets. In *Random and Quasi-Random Point Sets*, volume 1:38 of *Lecture Notes in Statistics*, pages 49–108. Springer-Verlag, New York, 1998.
13. F. J. Hickernell, H. S. Hong, P. L'Ecuyer, and C. Lemieux. Extensible lattice sequences for quasi-Monte Carlo quadrature. *SIAM Journal on Scientific Computing*, 22(3):1117–1138, 2000.
14. H. S. Hong and F. J. Hickernell. Algorithm 823: Implementing scrambled digital sequences. *ACM Transactions on Mathematical Software*, 29(2):95–109, 2003.

15. P. Jäckel. *Monte Carlo Methods in Finance.* John Wiley & Sons, New York, 2002.

16. S. Joe and F. Y. Kuo. Remark on algorithm 659: Implementing Sobol's quasirandom sequence generator. *ACM Transactions on Mathematical Software*, 29(1):49–57, 2003.

17. N. M. Korobov. The approximate calculation of multiple integrals using number theoretic methods. *Doklady Academii Nauk SSSR*, 115:1062–1065, 1957.

18. P. L'Ecuyer. Quasi-Monte Carlo methods with applications in finance. *Finance and Stochastics*, 13(3):307–349, 2009.

19. P. L'Ecuyer and A. B. Owen (editors). *Monte Carlo and Quasi-Monte Carlo Methods.* Springer-Verlag, New York, 2010.

20. P. L'Ecuyer and C. Lemieux. Variance reduction via lattice rules. *Management Science*, 46(9):1214–1235, 2000.

21. P. L'Ecuyer and C. Lemieux. *Recent Advances in Randomized Quasi-Monte Carlo Methods*, pages 419–474. Kluwer Academic Publishers, Boston, 2002.

22. C. Lemieux. *Monte Carlo and Quasi-Monte Carlo Sampling.* Springer-Verlag, New York, 2009.

23. J. Matoušek. On the $\mathcal{L}_2$-discrepancy for anchored boxes. *Journal of Complexity*, 14(4):527–556, 1998.

24. H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods.* SIAM, Philadelphia, PA, 1992.

25. A. B. Owen. Randomly permuted (t, m, s)-nets and (t, s)-sequences. In H. Niederreiter and J.-S. Shiue, editors, *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, pages 299–317. Springer-Verlag, New York, 1995.

26. A. B. Owen. Scrambled net variance for integrals of smooth functions. *Annals of Statistics*, 25(4):1541–1562, 1997.

27. K. F. Roth. On irregularities of distribution. *Mathematika*, 1(2):73–79, 1954.

28. I. H. Sloan and S. Joe. *Lattice Methods for Multiple Integration.* Clarendon Press, Oxford, 1994.

29. I. M. Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112, 1967.