Remark on Algorithm 659: Implementing Sobol's Quasirandom Sequence Generator

STEPHEN JOE
University of Waikato
and
FRANCES Y. KUO
University of New South Wales

An algorithm to generate Sobol' sequences to approximate integrals in up to 40 dimensions has been previously given by Bratley and Fox in Algorithm 659. Here, we provide more primitive polynomials and "direction numbers" so as to allow the generation of Sobol' sequences to approximate integrals in up to 1111 dimensions. The direction numbers given generate Sobol' sequences that satisfy Sobol's so-called Property A.

Categories and Subject Descriptors: G.1.4 [**Numerical Analysis**]: Quadrature and Numerical Differentiation—*multidimensional quadrature*

General Terms: Algorithms

Additional Key Words and Phrases: Low-discrepancy sequences, primitive polynomials, quasirandom sequences, Sobol' sequences

1. INTRODUCTION

One technique for approximating the d-dimensional integral

$$I_d(f) = \int_{[0,1]^d} f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$$

is to make use of Sobol' sequences. These were proposed by Sobol' [1967] and a computer implementation in Fortran 77 was subsequently given by Bratley and Fox [1988] as Algorithm 659. Other implementations are available as C, Fortran 77, or Fortran 90 routines in the popular Numerical Recipes collection of software (for example, see Press et al. [1992]). However, as given, all these implementations have a fairly heavy restriction on the maximum value

Authors' addresses: S. Joe, Department of Mathematics, University of Waikato, Private Bag 3105, Hamilton, New Zealand; email: stephenj@math.waikato.ac.nz; F. Y. Kuo, School of Mathematics, University of New South Wales, Sydney 2052, Australia; email: fkuo@maths.unsw.edu.au.

of d allowed. For Algorithm 659, Sobol' sequences may be generated to approximate integrals in up to 40 dimensions, while the Numerical Recipes routines allow the generation of Sobol' sequences to approximate integrals in up to six dimensions only. We remark that the FinDer software of Paskov and Traub [1995] provides an implementation of Sobol' sequences up to 370 dimensions, but it is licensed software. More information about FinDer is available at http://www.cs.columbia.edu/~ap/html/information.html.

As computers become more powerful, there is an expectation that it should be possible to approximate integrals in higher and higher dimensions. Integrals in hundreds of variables arise in applications such as mathematical finance (e.g., see Paskov and Traub [1995]). Also, as new methods become available for these integrals, one might wish to compare these new methods with Sobol' sequences. Thus, it would be desirable to extend these existing implementations such as Algorithm 659 so they may be used for higher-dimensional integrals. We remark that Sobol' sequences are now considered to be examples of (t, d)-sequences in base 2. The general theory of these low discrepancy (t, d)-sequences in base b is discussed in detail in Chapter 4 of Niederreiter [1992].

The generation of Sobol' sequences is clearly explained in Bratley and Fox [1988]. We review the main points so as to show what extra data would be required to allow Algorithm 659 to generate Sobol' sequences to approximate integrals in more than 40 dimensions. To generate the jth component of the points in a Sobol' sequence, we need to choose a primitive polynomial of some degree s_j in the field $\mathbb{Z}_2$, that is, a polynomial of the form

$$x^{s_j} + a_{1,j}x^{s_j-1} + \cdots + a_{s_j-1,j}x + 1,$$

where the coefficients $a_{1,j}, \ldots, a_{s_j-1,j}$ are either 0 or 1.

We use these coefficients to define a sequence $\{m_{1,j}, m_{2,j}, \ldots\}$ of positive integers by the recurrence relation

$$\begin{aligned} m_{k,j} \;=\; 2a_{1,j}m_{k-1,j} \oplus 2^2 a_{2,j}m_{k-2,j} \oplus \cdots \oplus 2^{s_j-1}a_{s_j-1,j}m_{k-s_j+1,j} \\ \oplus 2^{s_j} m_{k-s_j,j} \oplus m_{k-s_j,j}, \end{aligned} \tag{1}$$

for $k \geq s_j + 1$, where $\oplus$ is the bit-by-bit exclusive-OR operator. The initial values $m_{1,j}, m_{2,j}, \ldots, m_{s_j,j}$ can be chosen freely provided that each $m_{k,j}$, $1 \leq k \leq s_j$, is odd and less than 2^k. The "direction numbers" $\{v_{1,j}, v_{2,j}, \ldots\}$ are defined by

$$v_{k,j} := \frac{m_{k,j}}{2^k}.$$

Then $x_{i,j}$, the jth component of the ith point in a Sobol' sequence, is given by

$$x_{i,j} = b_1 v_{1,j} \oplus b_2 v_{2,j} \oplus \cdots,$$

where b_ℓ is the ℓth bit from the right when i is written in binary, that is, $(\cdots b_2 b_1)_2$ is the binary representation of i. In practice, a more efficient Gray code implementation proposed by Antonov and Saleev [1979] is used; see this reference or Bratley and Fox [1988] for details.

We then see that the implementation in Bratley and Fox [1988] may be used to generate Sobol' sequences to approximate integrals in more than 40 dimensions by providing more data in the form of primitive polynomials and direction

numbers (or equivalently, values of $m_{1,j}, m_{2,j}, \ldots, m_{s_j,j}$). When generating such Sobol' sequences, we need to ensure that the primitive polynomials used to generate each component are different and that the initial values of the $m_{k,j}$'s are chosen differently for any two primitive polynomials of the same degree. The error bounds for Sobol' sequences given in Sobol' [1967] indicate we should use primitive polynomials of as low a degree as possible.

We discuss how additional primitive polynomials may be obtained in the next section. After these primitive polynomials have been found, we need to decide upon the initial values of the $m_{k,j}$ for $1 \leq k \leq s_j$. As explained above, all we require is that they be odd and that $m_{k,j} < 2^k$. Thus, we could just choose them randomly, subject to these two constraints. However, Sobol' [1976] introduced an extra uniformity condition known as Property A. Geometrically, if the cube $[0, 1]^d$ is divided up by the planes $x_j = 1/2$ into 2^d equally sized subcubes, then a sequence of points belonging to $[0, 1]^d$ possesses Property A if, after dividing the sequence into consecutive blocks of 2^d points, each one of the points in any block belongs to a different subcube.

Property A is not that useful to have for large d because of the computational time required to approximate an integral using 2^d points. Also, Property A is not enough to ensure that there are no bad correlations between pairs of dimensions (see Section 7 of Morokoff and Caflisch [1994] for a discussion). Nevertheless, Property A would seem a reasonable criterion to use in deciding upon a choice of the initial $m_{k,j}$. The numerical results for Sobol' sequences given in Section 4 suggest that the direction numbers obtained here are indeed reasonable.

Other ways of obtaining the direction numbers are also possible. For example, in Cheng and Druzdzel [2000], the initial direction numbers are obtained by an interesting technique of minimizing a measure of uniformity in two dimensions. This technique may alleviate the problem of bad correlations between pairs of dimensions that was mentioned above.

Sobol' [1976] showed that a Sobol' sequence used to approximate a d-dimensional integral possesses Property A if and only if

$$\det(V_d) = 1 \,(\mathrm{mod}\, 2),$$

where V_d is the $d \times d$ binary matrix defined by

$$V_d := \begin{bmatrix} v_{1,1,1} & v_{2,1,1} & \cdots & v_{d,1,1} \\ v_{1,2,1} & v_{2,2,1} & \cdots & v_{d,2,1} \\ \vdots & \vdots & \ddots & \vdots \\ v_{1,d,1} & v_{2,d,1} & \cdots & v_{d,d,1} \end{bmatrix}, \tag{2}$$

with $v_{k,j,1}$ denoting the first bit after the binary point of $v_{k,j}$.

The primitive polynomials and direction numbers used in Algorithm 659 are taken from Sobol' and Levitan [1976] and a subset of this data may be found in Sobol' [1976]. Though it is mentioned in Sobol' [1976] that Property A is satisfied for $d \leq 16$, that is, $\det(V_d) = 1 \,(\mathrm{mod}\, 2)$ for all $d \leq 16$, our calculations showed that Property A is actually satisfied for $d \leq 20$. As a result, we change the values of the $m_{k,j}$ for $21 \leq j \leq 40$, but keep the primitive polynomials. For $j \geq 41$, we obtain additional primitive polynomials.

The number of primitive polynomials of degree s is $\phi(2^s - 1)/s$, where ϕ is Euler's totient function. Including the special case for $j = 1$ when all the $m_{k,1}$ are 1, this allows us to approximate integrals in up to dimension $d = 1111$ if we use all the primitive polynomials of degree 13 or less.

We then choose values of the $m_{k,j}$ so that we can generate Sobol' sequences satisfying Property A in dimensions d up to 1111. This is done by generating some values randomly, but these are subsequently modified so that the condition $\det(V_d) = 1 \pmod 2$ is satisfied for all d up to 1111. This process involves evaluating values of the $v_{k,j,1}$'s to obtain the matrix V_d and then evaluating the determinant of V_d. A more detailed discussion of this strategy is given in the next section. It is not difficult to produce values to generate Sobol' points for approximating integrals in even higher dimensions.

2. GENERATING THE PRIMITIVE POLYNOMIALS AND DIRECTION NUMBERS

2.1 Obtaining Primitive Polynomials

Recall that we are interested in the primitive polynomials of the form

$$x^{s_j} + a_{1,j}x^{s_j-1} + \cdots + a_{s_j-1,j}x + 1,$$

where the coefficients $a_{1,j}, \ldots, a_{s_j-1,j}$ are either 0 or 1. We represent such a polynomial by $P_{s_j,a_j}(x)$, where a_j is the decimal value of the binary number $(a_{1,j}a_{2,j}\cdots a_{s_j-1,j})_2$, that is,

$$a_j = \sum_{k=1}^{s_j-1} a_{k,j}2^{s_j-k-1}.$$

Note that though this representation of the primitive polynomial using a_j is also used in Bratley and Fox [1988], the Fortran 77 routines associated with this paper use $\bar{a}_j$ instead, where

$$\bar{a}_j = 2^{s_j} + 2a_j + 1$$

is the decimal value of the binary number $(1a_{1,j}a_{2,j}\cdots a_{s_j-1,j}1)_2$.

Finding out whether a given polynomial in $\mathbb{Z}_2$ is a primitive polynomial is not a trivial task. Fortunately, there are computer programs available on the Internet that will compute all the possible primitive polynomials of specified degree. We obtain the coefficients using a computer program downloaded from ftp://helsbreth.org/pub/helsbret/random/lfsr_s.c. In order to check that the primitive polynomials generated from this program were correct, they were compared with those generated by using a different computer program, which we downloaded from http://www.theory.csc.uvic.ca/~cos/dis/distribute.pl.cgi?package=poly.c.

For $d \le 40$, we keep the primitive polynomials as they are in Algorithm 659. For $d \ge 41$, we adopt a systematic approach in which we arrange the primitive polynomials of the same degree in increasing order of the a_j. In Algorithm 659 all the primitive polynomials up to degree 7 were used plus three out of the 16 primitive polynomials of degree 8. The remaining 13 are used for dimensions

$d = 41$ to $d = 53$. The 48 primitive polynomials of degree 9 are used in the same way for d from 54 to 101. There are 60 primitive polynomials of degree 10 that are used for $102 \le d \le 161$, 176 of degree 11 used for $162 \le d \le 337$, 144 of degree 12 used for $338 \le d \le 481$, and 630 polynomials of degree 13 used for d from 482 to 1111.

2.2 Evaluation of the $v_{k,j,1}$

In order to test for Property A, we need to form the matrix V_d (see (2)), which contains entries consisting of the first bit after the binary point of the direction numbers. Thus, we do not need to evaluate the directions numbers fully, but only need the first bit of each direction number. Since the initial direction numbers $v_{k,j}$ are given by $m_{k,j}/2^k$, for $1 \le k \le s_j$, then it is clear that $v_{k,j,1} = 1$ if $m_{k,j}/2^k \ge 1/2$ and $v_{k,j,1} = 0$ if $m_{k,j}/2^k < 1/2$. Such considerations lead to the following lemma.

LEMMA 1. *For $j = 1$, assume we have the special case in which all the $m_{k,1}$ have the value 1. Then*

$$v_{1,1,1} = 1 \quad \textit{and} \quad v_{k,1,1} = 0 \text{ for } k \ge 2.$$

Given choices of primitive polynomials and initial values $m_{k,j}$ for $1 \le k \le s_j$, then

$$v_{k,j,1} = \begin{cases} 0, & \text{if } m_{k,j} < 2^{k-1}, \\ 1, & \text{if } m_{k,j} \ge 2^{k-1}, \end{cases} \quad 1 \le k \le s_j, \quad j \ge 2.$$

To form V_d, we also need values of the $v_{k,j,1}$ for $k > s_j$. The required result is given in the next lemma.

LEMMA 2. *For $j \ge 2$ and $k > s_j$, we have*

$$v_{k,j,1} = a_{1,j} v_{k-1,j,1} \oplus a_{2,j} v_{k-2,j,1} \oplus \cdots \oplus a_{s_j-1,j} v_{k-s_j+1,j,1} \oplus v_{k-s_j,j,1}.$$

PROOF. Because $v_{k,j} = m_{k,j}/2^k$, it follows from the recurrence relation (1) that

$$v_{k,j} = a_{1,j} v_{k-1,j} \oplus a_{2,j} v_{k-2,j} \oplus \cdots \oplus a_{s_j-1,j} v_{k-s_j+1,j} \oplus v_{k-s_j,j} \oplus \frac{v_{k-s_j,j}}{2^{s_j}}.$$

Since $\oplus$ is a bit-by-bit operator and $v_{k-s_j,j}/2^{s_j} < 1/2$, the required result follows. □

2.3 Finding Initial Values of the $m_{k,j}$

Recall that for $j \le 20$, we use the existing $m_{k,j}$ in Algorithm 659 as Property A is already satisfied. For j between 21 and 40, we start with the existing $m_{k,j}$ in Algorithm 659 and for j successively taking the values $41 \ldots 1111$, we randomly generate $m_{1,j}, \ldots, m_{s_j,j}$ such that they are odd and satisfy $m_{k,j} < 2^k$. We can then use Lemmas 1 and 2 to form V_j for each j from 21 to 1111 and test whether $\det(V_j) = 1 \pmod 2$ is satisfied. If it is, we can then proceed to the next value of j.

If the determinant is $0 \pmod 2$, then we need to modify the initial values of $m_{k,j}$'s so that $\det(V_j) = 1 \pmod 2$ is satisfied. Since $v_{k,j,1}$ is 0 when

$m_{k,j} \in [1, 2^{k-1})$ and 1 when $m_{k,j} \in [2^{k-1}, 2^k)$, then $v_{k,j,1}$ may be changed by replacing $m_{k,j}$ by $(m_{k,j} + 2^{k-1}) \pmod{2^k}$.

Starting with $m_{2,j}$, we replace it by $(m_{2,j} + 2) \pmod 4$ (which, in effect, changes the value of $v_{2,j,1}$) and then re-evaluate the determinant. If the determinant is still $0 \pmod 2$, we change this new value of $m_{2,j}$ back to its original value and then replace $m_{3,j}$ by $(m_{3,j} + 4) \pmod 8$ and re-evaluate the determinant. We repeat the same process for $m_{4,j}, m_{5,j}, \ldots$ until the determinant is $1 \pmod 2$ or until $m_{s_j,j}$ is reached. If this latter stage is reached, then we generate another random set of $m_{1,j}, \ldots, m_{s_j,j}$ and repeat the process. As it turned out in our calculations, this was never the case.

2.4 Evaluation of the Determinant

The evaluation of the determinant can be simplified since when working in $\mathbb{Z}_2$:

(i) Swapping rows does not change the determinant.

(ii) Adding a row to another (and likewise subtracting a row from another) is equivalent to applying an exclusive-OR operation. This also leaves the determinant unchanged.

Because bit operations like the exclusive-OR operation are very quick in a programming language such as C++, the determinant of V_d may be found quite quickly by attempting to row-reduce V_d to an upper triangular matrix with 1's all the way down its main diagonal. If we succeed, the determinant is 1. If, at some stage, we end up with a row of zeros, then the determinant is 0.

This process may be speeded up by making use of the fact that once we have the $m_{k,j}$ for $1 \le k \le s_j$ and $1 \le j \le d$, then V_d is fixed and so are the first d rows of V_j for $j > d$. We first row-reduce the first 20 rows of V_{1111} so that we have 1's down the main diagonal and 0's below the 1's. (This is possible because the determinants of $V_1, \ldots, V_{20}$ are all 1.) Then to evaluate the determinant of V_j for each j from 21 to 1111, we only need to row-reduce one extra row each time and thus avoid the repetitions in row operations.

3. MODIFICATIONS TO ALGORITHM 659

A table containing values of j, s_j, a_j, $\bar{a}_j$, and $m_{k,j}$, $k = 1, \ldots, s_j$, for j going from 2 to 1111 is available as an ascii text file which may be downloaded from http://www.math.waikato.ac.nz/~stephenj/soboltab.txt. In the case $j = 1$, the Bratley and Fox implementation uses the special case of $m_{k,1} = 1$ for $k \ge 1$.

For the generation of Sobol' sequences, Algorithm 659 essentially consists of the block-data routine BDSOBL, an initialization subroutine INSOBL, and the subroutine GOSOBL which does the actual generation of the Sobol' sequence. Because computations now tend to be done in double precision rather than single precision, the declaration of all the floating point variables in those routines was changed from REAL to DOUBLE PRECISION.

The routine BDSOBL was modified to include the new values of the $m_{k,j}$, generated as described in the previous section. The integer arrays POLY and VINIT which contain the $\bar{a}_j$ and $m_{k,j}$ values, respectively, were changed from their original declaration of POLY(2:40) and VINIT(2:40,8) to

POLY(2:1111),VINIT(2:1111,13) and the new values of $\bar{a}_j$ and $m_{k,j}$ were put in. This then allows the generation of Sobol' points to approximate integrals in up to 1111 dimensions using primitive polynomials of degree up to 13 (instead of up to 40 dimensions using primitive polynomials of degree up to 8 in the original implementation). In INSOBL, the check that the dimension did not exceed 40 was changed to a check that the dimension did not exceed 1111 with a corresponding change to the comment.

In the subroutines INSOBL and GOSOBL, the integer array V was changed from its original declaration of V(40,30) to V(1111,30) to reflect the change in VINIT. (As in the original implementation, we still assume that we are working on a computer with word length at least 31 bits excluding sign.) Similar changes were made to the declaration of LASTQ and QUASI. The final change was in the declaration of the logical array INCLUD from INCLUD(8) to INCLUD(13) since the primitive polynomials now used have degree up to 13.

4. NUMERICAL RESULTS

Here, we provide the results of some calculations obtained by approximating the test integral

$$\int_{[0,1]^d} \prod_{j=1}^{d} \frac{|4x_j - 2| + c_j}{1 + c_j} \, \mathrm{d}\boldsymbol{x} = 1.$$

In the integrand, the c_j are parameters that govern the difficulty of the integration problem. For reasons explained below, we took $c_j = j^{1/3}$ for our numerical experiments. When all the c_j are zero, the integrand becomes the one used in the numerical tests of Bratley and Fox [1988]. However, the numerical results there and the discussion in Section 1.5 of Fox [1986] indicate that the integration problem is already quite difficult when $d = 40$ and will become even more difficult as d increases.

In order to obtain a test integral that is still reasonable in high dimensions, we took $c_j = j^{1/3}$. The test integral was taken from Wang and Fang [2002], where the effective dimension of the integration problem is investigated. In that work, the choices $c_j = j$ and $c_j = j^2$ were used, but the analysis there indicated that the resulting integrals have an effective dimension much less than d. Hence we took $\{c_j\}$ to be a slower growing sequence.

To avoid problems caused by a bad choice of the direction numbers, it is normally recommended that the initial portion of the sequence be dropped. Here we adopt the strategy in Acworth et al. [1998] where the number of points skipped is the largest power of two smaller than n.

For comparison purposes, we present results for Sobol' sequences, Faure sequences (obtained using a modified version of Algorithm 647 [Fox 1986]), and shifted lattice rules. The latter are rules of the form

$$\frac{1}{n} \sum_{i=1}^{n} f\left(\left\{ \frac{i\boldsymbol{z}}{n} + \boldsymbol{\Delta} \right\} \right),$$

where $\boldsymbol{z} \in \mathbb{Z}^d$ has no component divisible by n, $\boldsymbol{\Delta} \in [0, 1)^d$ is the random 'shift', and the braces around a vector indicate that we take the fractional part of

$d = 50$	1009	1997	4001	8009	16001	32003	64007	128021
SOBOL'	0.9743	0.9544	0.9838	0.9982	0.9995	0.9976	0.9969	0.9975
FAURE	1.0562	0.9114	1.0092	1.0324	1.0207	1.0128	1.0033	0.9983
LATTICE	1.0002	0.9871	1.0008	1.0030	1.0006	1.0014	1.0009	0.9999
$d = 100$								
SOBOL'	1.0157	0.9288	0.9649	0.9866	0.9985	0.9817	0.9955	0.9995
FAURE	1.2231	0.9612	1.0816	0.9788	1.0000	1.0079	0.9938	0.9938
LATTICE	1.0026	0.9905	1.0073	1.0017	0.9982	0.9974	0.9988	1.0053
$d = 200$								
SOBOL'	1.1109	0.8877	0.9070	1.0312	1.0216	0.9772	0.9744	0.9925
FAURE	0.6699	0.7872	1.8193	1.1903	1.0970	1.0014	1.0151	1.0602
LATTICE	0.9966	1.0354	0.9773	0.9911	1.0396	1.0082	0.9973	1.0008
$d = 300$								
SOBOL'	0.9770	0.8715	0.9305	0.9546	1.0060	0.9712	0.9770	0.9843
FAURE	0.4644	0.6595	0.8060	1.0534	0.9018	0.8150	0.9765	1.0045
LATTICE	0.8614	0.9220	0.9630	0.9849	0.9804	1.0109	0.9908	1.0012
$d = 400$								
SOBOL'	1.1780	0.8908	0.9710	0.9527	1.0203	0.9754	0.9779	0.9725
FAURE	5.4082	3.5749	2.4613	1.8599	1.1645	1.0616	0.9183	0.8683
LATTICE	1.0737	0.9772	0.8818	1.0037	1.0045	0.9563	0.9690	0.9697
$d = 500$								
SOBOL'	1.0567	0.8548	0.9700	0.9704	0.9768	0.9634	0.9745	0.9670
FAURE	0.3150	0.3009	0.4686	0.8643	0.6999	1.0556	0.8375	0.9319
LATTICE	1.0964	0.7656	1.0947	0.9932	1.0087	1.0316	1.0151	0.9971
$d = 600$								
SOBOL'	0.8735	0.7674	1.0130	0.9874	0.9677	0.9416	0.9580	0.9564
FAURE	10.091	6.0579	4.8223	3.8932	2.4430	1.7774	1.5841	1.1556
LATTICE	0.9619	0.9702	0.8333	1.0277	0.8976	0.9722	1.0140	1.0351
$d = 700$								
SOBOL'	0.8752	0.8241	1.0067	1.0078	0.9987	0.9236	0.9517	0.9475
FAURE	3.2886	2.3165	1.3131	0.7920	0.6197	1.2220	0.9232	0.8749
LATTICE	0.8274	1.0725	0.8967	0.8970	0.9438	0.9282	0.9564	1.0129
$d = 800$								
SOBOL'	0.9139	0.8822	1.0926	1.0560	1.0210	0.9454	0.9330	0.9359
FAURE	0.1397	0.2593	0.9715	0.7057	0.6972	0.8922	0.8361	0.7847
LATTICE	0.9688	0.9789	0.9863	0.9095	0.9839	1.1132	0.9920	1.0058
$d = 900$								
SOBOL'	0.7779	1.0225	1.0593	1.0991	1.0470	0.9115	0.9688	0.9615
FAURE	0.2561	0.2172	0.1553	0.6044	0.4295	1.3487	0.9747	0.9498
LATTICE	0.7497	0.7903	0.9413	0.8846	0.8896	0.9162	0.9830	1.1351
$d = 1000$								
SOBOL'	0.8348	1.0443	1.0710	1.0862	1.0483	0.9188	0.9735	0.9683
FAURE	0.1281	0.6392	1.3443	0.7403	0.7518	0.5455	0.4360	1.8214
LATTICE	1.0471	1.1158	0.7563	1.0610	0.9724	1.0651	0.9274	0.9453
$d = 1111$								
SOBOL'	0.7524	0.9139	0.9592	0.9703	1.0116	0.9124	1.0411	0.9904
FAURE	3.3312	2.9087	1.5509	0.9330	1.2253	0.7802	1.2843	1.0532
LATTICE	0.8144	0.9452	0.9321	0.8588	0.9602	0.9418	1.0510	0.9120

each component of the vector. More information about such rules may be found in Sloan et al. [2002]. The generating vectors $\boldsymbol{z}$ used were obtained using the algorithm given in this reference with all the weights taken to be one.

We present approximations to the integral in the table above for $d = 50$, then from $d = 100$ to $d = 1000$ in steps of 100, followed by results for $d = 1111$. The number of points used in the calculations were $n = 1009, 1997, 4001, 8009,$

16001, 32003, 64007, and 128021. These values of n were taken to be prime to be consistent with the theory for shifted lattice rules in Sloan et al. [2002].

The numerical results for the Faure sequences tend to be the least accurate. For the Sobol' sequences and shifted lattice rules, there does not appear to be any significant difference in their results.

REFERENCES

ACWORTH, P. A., BROADIE, M., AND GLASSERMAN, P. 1998. A comparison of some Monte Carlo and quasi-Monte Carlo techniques for option pricing. In *Monte Carlo and Quasi-Monte Carlo Methods 1996*, H. Niederreiter, P. Hellekalek, G. Larcher, and P. Zinterhof, Eds. Lecture Notes in Statistics, vol. 127. Springer, New York, 1–18.

ANTONOV, I. A. AND SALEEV, V. M. 1979. An economic method of computing LP_τ-sequences. *Zh. vȳchisl. Mat. mat. Fiz. 19*, 243–245. English translation: *U.S.S.R. Comput. Maths. Math. Phys. 19*, 252–256.

BRATLEY, P. AND FOX, B. L. 1988. Algorithm 659: Implementing Sobol's quasirandom sequence generator. *ACM Trans. Math. Softw. 14*, 88–100.

CHENG, J. AND DRUZDZEL, M. J. 2000. Computational investigation of low-discrepancy sequences in simulation algorithms for Bayesian networks. In *Proceedings of the 16th Annual Conference on Uncertainty in Artificial Intelligence*, C. Boutilier and M. Goldszmidt, Eds. Morgan-Kaufmann, San Francisco, Calif., 72–81.

FOX, B. L. 1986. Algorithm 647: Implementation and relative efficiency of quasirandom sequence generators. *ACM Trans. Math. Softw. 12*, 362–376.

MOROKOFF, W. J. AND CAFLISCH, R. E. 1994. Quasi-random sequences and their discrepancies. *SIAM J. Sci. Comput. 15*, 1251–1279.

NIEDERREITER, H. 1992. *Random Number Generation and Quasi-Monte Carlo methods*. SIAM, Philadelphia, Pa.

PASKOV, S. H. AND TRAUB, J. F. 1995. Faster valuation of financial derivatives. *J. Portf. Manage. 22*, 113–120.

PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. 1992. *Numerical Recipes in Fortran 77: The Art of Scientific Computing*, 2nd ed. Cambridge University Press, Cambridge, U.K.

SLOAN, I. H., KUO, F. Y., AND JOE, S. 2002. Constructing randomly shifted lattice rules in weighted Sobolev spaces. *SIAM J. Numer. Anal. 40*, 1650–1665.

SOBOL', I. M. 1967. On the distribution of points in a cube and the approximate evaluation of integrals. *Zh. vȳchisl. Mat. mat. Fiz. 7*, 784–802. English translation: *U.S.S.R. Comput. Maths. Math. Phys. 7*, 86–112.

SOBOL', I. M. 1976. Uniformly distributed sequences with an additional uniform property. *Zh. vȳchisl. Mat. mat. Fiz. 16*, 1332–1337. English translation: *U.S.S.R. Comput. Maths. Math. Phys. 16*, 236–242.

SOBOL', I. M. AND LEVITAN, Y. L. 1976. The production of points uniformly distributed in a multi-dimensional cube. Tech. Rep. 40, Institute of Applied Mathematics, USSR Academy of Sciences. (In Russian).

WANG, X. AND FANG, K.-T. 2002. The effective dimensions and quasi-Monte Carlo integration. *J. Complexity*, submitted for publication.

Received February 2002; revised November 2002; accepted December 2002