Parameterization Based on Randomized Quasi-Monte Carlo Methods

Giray Ökten and Matthew Willyard
Department of Mathematics
Florida State University
Tallahassee, FL 32306-4510
okten@math.fsu.edu
mwillyar@math.fsu.edu

Abstract

We present a theoretical framework where any randomized quasi-Monte Carlo method can be viewed and analyzed as a parameterization method for parallel quasi-Monte Carlo. We present deterministic and stochastic error bounds when different processors of the computing environment run at different speeds. We implement two parameterization methods, both based on randomized quasi-Monte Carlo, and apply them to pricing digital options and collateralized mortgage obligations. Numerical results are used to compare the parameterization methods by their parallel performance as well as their Monte Carlo efficiency.

1 Introduction

In recent years, several researchers have suggested methods for parallelizing quasi-Monte Carlo (QMC) sequences. Leap-frogging and blocking were considered in [2], [4], [13], [31], [32], and methods based on parameterization were introduced in [6], [10], and [26]. In heterogeneous computing environments where different processors can run at different speeds (or fail), the disadvantages of leap-frogging are well documented both theoretically and numerically ([26], [33]). In general, blocking requires careful partitioning of the sequence among the processors to prevent possible overlapping. In the absence of overlap, this method can give competitive results. In the numerical results obtained in Ökten and Srinivasan [26] and Srinivasan [33], blocking of the scrambled Halton sequence (see [7]) gave better or comparable results to the parameterization approach introduced in Ökten and Srinivasan [26]. However, blocking of another QMC sequence (see [1]) gave worse results than the parameterization approach in the same paper.

The parameterization approach of De Doncker et. al. [6] use randomly shifted QMC sequences, and the approach of Ökten and Srinivasan [26] selects at random a scrambled Halton sequence from a finite set of Halton sequences. These randomization methods are examples of randomized quasi-Monte Carlo (RQMC) methods. In the next section, we will give a brief overview of RQMC methods. In Section 3, we will provide a unified framework where any RQMC method can be viewed as a parameterized QMC approach, and discuss error analysis. The theoretical advantages of RQMC based parameterization over blocking will be discussed in the same section. Finally, in Section 4, we will use two RQMC based parameterization methods in pricing digital options and mortgage backed securities.

2 Randomized quasi-Monte Carlo methods

The QMC method estimates the integral $I = \int_{[0,1)^s} f(x)dx$, using sums of the form $\frac{1}{N}\sum_{n=1}^{N} f(q^{(n)})$ where $q^{(n)}$ is the nth term of an s-dimensional low-discrepancy[1] sequence. The randomized quasi-Monte Carlo (RQMC) method uses a family of s-dimensional low-discrepancy sequences q_u, indexed by the random parameter u, and approximates I by

$$Q(q_u) = \frac{1}{N}\sum_{n=1}^{N} f(q_u^{(n)}). \tag{1}$$

There are three important results common to most RQMC methods:

1. $E[Q(q_u)] = I$; i.e., the estimates (1) indexed by the random parameter u are unbiased[2];
2. $Var(Q(q_u)) = O\left(N^{-2}(\log N)^{2s}\right)$ (or, better, for certain integrands and RQMC methods);

[1] We use the terms *QMC sequence* and *low-discrepancy sequence* interchangebly in this paper.

[2] The RQMC method *finite random sampling from low-discrepancy sequences*, which is the method used in the parameterization approach of [26], gives only asymptotically unbiased estimates. See [24] for details.

3. $|Q(q_u) - I| \leq V(f)D_N^*(q_u)$; i.e., each estimate satisfies the Koksma-Hlawka inequality where $V(f)$ is the variation of f in the sense of Hardy and Krause, and $D_N^*(q_u)$ is the star-discrepancy of the first N terms of the sequence q_u (see [19] for details).

In practice, the random parameter u usually has the uniform distribution, and $E[Q(q_u)] = I$ is estimated by the sample mean

$$\frac{Q(q_{u_1}) + ... + Q(q_{u_M})}{M}$$

where $u_1, ..., u_M$ are independent samples from u.

A detailed survey of RQMC methods is given by Ökten and Eastman [25]. Here we will give a brief summary of them:

1. Scrambled (t, m, s)-nets and (t, s)-sequences.

   (t, m, s)-nets and (t, s)-sequences are special constructions of QMC point sets and sequences; their theory is discussed in [19]. In [21], Owen introduced a way of randomizing these nets and sequences by scrambling. Additional results were obtained in [8], [9], [14], [22], and [23]. Some of these results provide improvements for $Var(Q(q_u))$. For example, for "smooth" functions Owen [23] shows that the variance of the quadrature rule based on scrambled nets can be as small as $O\left(N^{-3}(\log N)^{s-1}\right)$.

2. Alternative scrambling methods for (t, m, s)-nets and (t, s)-sequences.

   Owen's scrambling method is computationally very demanding. In [16], Matoušek introduced alternative scrambling methods, which limit the "randomness" in the original scrambling method for efficiency. These alternative scrambling methods still satisfy some of the theoretical properties of the original scrambling method. Two of the methods Matoušek discussed are called *random digit scrambling*, and *random linear scrambling*. These are the methods we will consider in Section 4.

3. Random shifting.

   This method can be applied to any QMC sequence. An independent randomization of the sequence $q^{(n)}$ is obtained by first generating a random vector u from the uniform distribution on $(0, 1)^s$. Then we add u to $q^{(n)}$ (for all n) componentwise, and take the fractional part of the sum. Properties and applications of random shifting are considered in [5], [11], [17], [18], and [36].

4. Random-start Halton sequences.

   The van der Corput and Halton sequences are popular examples of low-discrepancy sequences. One way to describe the van der Corput sequence is by using the von Neuman-Kakutani transformation, T_b, which is a well-known example of an ergodic mapping on $(0, 1)$. The van der Corput sequence in base b is simply the orbit of 0 under T_b. The Halton sequence in bases $b_1, ..., b_s$ is defined as the orbit of 0 under the s-dimensional von Neuman-Kakutani transformation $T_{\mathbf{b}}(\mathbf{x}) = (T_{b_1}(x_1), ..., T_{b_s}(x_s))$. An independent realization of a random-start Halton sequence involves generating a random vector u from the uniform distribution on $(0, 1)^s$, and then constructing the orbit of $T_{\mathbf{b}}(u)$. For a given u, the orbit of $T_{\mathbf{b}}(u)$ is known to be a low-discrepancy sequence. Properties of these sequences and the random-start approach are studied in [12], [34], and [37].

5. Finite random sampling from low-discrepancy sequences.

   This method constructs a finite space of QMC sequences. An independent randomization involves picking at random (from the discrete uniform distribution) a sequence from this space, and obtaining an estimate for I. This method does not produce unbiased estimates, but only asymptotically unbiased estimates. The parameterization approach of [26] is based on this method. See [24] for details on this method.

3 Parameterization and error analysis

Consider a computing environment with m processors, and let us assume that the problem at hand is to estimate

$$I = \int_{[0,1)^s} f(x)dx.$$

By an RQMC based parameterization method we mean the following: using one of the RQMC methods described in the previous section, each processor independently generates an RQMC sequence at random. Since this generation involves pseudorandom numbers, we assume there is a parallel pseudorandom number generator available for the computing environment.

Each processor computes a specific sequence of QMC vectors, and evaluates the function values at these vectors. Assume that a total of N vectors have been generated at a given time, and $\lambda_i(N)$ is the number of vectors generated by the ith processor, and thus, $N = \sum_{i=1}^{m} \lambda_i(N)$. If an estimate for the problem is requested at this time (by the master processor), then each processor computes its own estimate

$$I_i = \frac{1}{\lambda_i(N)} \sum_{n=1}^{\lambda_i(N)} f(q_i^{(n)}).$$

Having collected this information from all processors, the master processor can compute the final estimate:

$$I = \sum_{i=1}^{m} \frac{\lambda_i(N)}{N} I_i. \tag{2}$$

The error of the estimate (2) satisfies the following bound

$$\begin{aligned} & \left| I - \int_{[0,1)^s} f(x)dx \right| && (3) \\ \leq & \sum_{i=1}^{m} \left(\frac{\lambda_i(N)}{N} \left| I_i - \int_{[0,1)^s} f(x)dx \right| \right) && (4) \\ \leq & \sum_{i=1}^{m} \frac{\lambda_i(N)}{N} D^*_{\lambda_i(N)}(q_i^{(n)}) V(f), \end{aligned}$$

where the last step uses the Koksma-Hlawka inequality and assumes that f is a function of bounded variation in the sense of Hardy and Krause. This result shows that the estimation error is bounded by a weighted average of terms that have the QMC convergence rate of $O(\lambda_i(N)^{-1}(\log \lambda_i(N))^s)$. If we assume that each processor has a fixed speed relative to the other processors, i.e.,

$$\lim_{N\to\infty} \frac{\lambda_i(N)}{N} = \lambda_i$$

for some constant λ_i, then we can take the limit of (3) as $N \to \infty$ to conclude that $\left| I - \int_{[0,1)^s} f(x)dx \right| \to 0$. Of course, this result only applies to sequences, not finite point sets. For the special case when all processors run at equal speeds, and thus $\lambda_i(N) = N/m$, the error bound (3) simplifies to

$$\left| I - \int_{[0,1)^s} f(x)dx \right| \leq \frac{1}{m} \sum_{i=1}^{m} D^*_{N/m}(q_i^{(n)}) V(f). \tag{5}$$

If one of the unbiased RQMC methods ((1)-(4) in Section 2) is used, then the following probabilistic error analysis can be used:

$$\begin{aligned} Var\left[I - \int_{[0,1)^s} f(x)dx \right] &= E \left| I - \int_{[0,1)^s} f(x)dx \right|^2 \\ &= \sum_{i=1}^{m} \left(\frac{\lambda_i(N)}{N} \right)^2 E \left| I_i - \int_{[0,1)^s} f(x)dx \right|^2 \\ &= \sum_{i=1}^{m} \left(\frac{\lambda_i(N)}{N} \right)^2 Var(I_i) \end{aligned}$$

and since $Var(I_i)$ is $O\left(\lambda_i(N)^{-2}(\log \lambda_i(N))^{2s}\right)$, or possibly better, as explained in the previous section, we obtain

$$\begin{aligned} & Var\left[I - \int_{[0,1)^s} f(x)dx \right] \\ = & \sum_{i=1}^{m} \left(\frac{\lambda_i(N)}{N} \right)^2 O\left(\frac{(\log \lambda_i(N))^{2s}}{\lambda_i(N)^2} \right). \end{aligned}$$

For equal speeds, this result simplifies to

$$Var\left[I - \int_{[0,1)^s} f(x)dx \right] = \frac{1}{m} O\left(\frac{(\log N/m)^{2s}}{(N/m)^2} \right).$$

An advantage of RQMC based parameterization over blocking or leap-frogging is the availability of statistical error estimation. This, in turn, yields a stopping criterion: one can compute the sample variance of the estimates coming from different processors, and stop when it reaches a predetermined level. There is no such stopping criterion when a single QMC sequence is partitioned through blocking or leap-frogging.

4 Numerical results

4.1 Digital options

We assume the lognormal model for the stock price process, and consider a digital option written on the stock. A discussion of digital options can be found in [27]. There is an analytical formula for the price of a digital option. We use this formula and the exact price it gives to compare different simulation techniques by the exact error they produce.

In Fig. 1, we plot the absolute value of error when the price of the option is estimated by simulation. The methods we compare are the Monte Carlo method that uses the SPRNG library ([15]), and two RQMC methods; random digit scrambling, and random linear scrambling of the Faure sequence. All of these methods are implemented within the parameterization framework discussed earlier. The computing environment is a high performance cluster (FSU HPC) consisting of four head nodes and 128 compute nodes (512 cores). For a given sample size, two estimates are obtained using two cores. We then compute their average, which we call the grand estimate, and Fig. 1 plots the error of the grand estimate. Both RQMC implementations outperform SPRNG. Linear scrambling is slightly better than linear digital scrambling. The approximate convergence rate is $O(N^{-0.42})$ for SPRNG (N is the sample size), which is consistent with the Monte Carlo convergence rate $O(N^{-0.5})$. Since the dimension (number of time steps) is 10 in this problem, the theoretical QMC convergence rate

is $O(N^{-1}(\log N)^{10})$. However, the approximate convergence rates observed are closer to $O(N^{-1})$: we observe $O(N^{-0.9})$ & $O(N^{-0.87})$ for linear scrambling & linear digital scrambling methods.

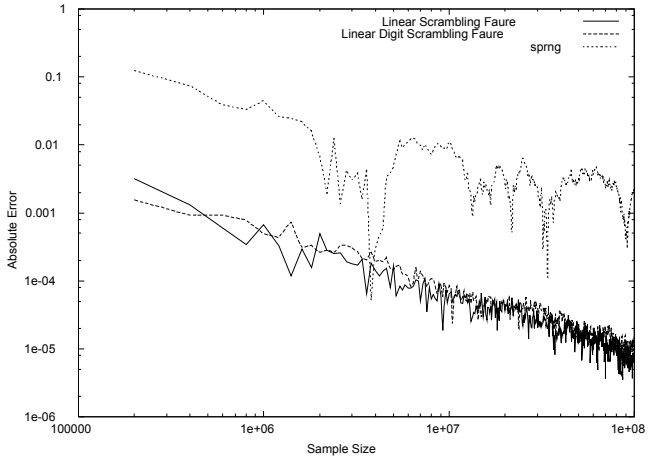

Figure 1. Digital options. Error of RQMC and Monte Carlo.

In Fig. 2, we investigate the speedup and efficiency of the linear scrambling (LS) and linear digital scrambling (LDS) methods against the number of processors. Speedup is the time to run the sequential algorithm divided by the execution time of the parallel algorithm, and efficiency is the speedup divided by the number of processors. The number of processors used are: 1 node & 1 core (1 processor), 1 node & 2 cores (2 processors), 1 node & 4 cores (4 processors), and 2 nodes & 4 cores (8 processors).

It is interesting to note the superlinear speedup when there is a single node. In linear scrambling and linear digital scrambling of the Faure sequence, there are several matrix calculations, and the size of the matrices grow with the total number of Faure vectors that need to be generated. Therefore, generating N vectors is slightly more than twice as expensive as generating $N/2$ vectors. For example, it took 2,689 seconds to generate the first fifty million Faure vectors on a single core, but only 1,116 seconds, less than half as long, to generate the first twenty-five million. Thus on a single node where the communication time is extremely small, we can expect super linear growth.

4.2 Mortgage backed securities

Pricing a collateralized mortgage obligation problem using QMC methods was first discussed in Paskov [28], [29]. This is an example of a *nominally* high-dimensional problem where QMC provides very fast convergence. A discussion on the effective dimension of the problem is given

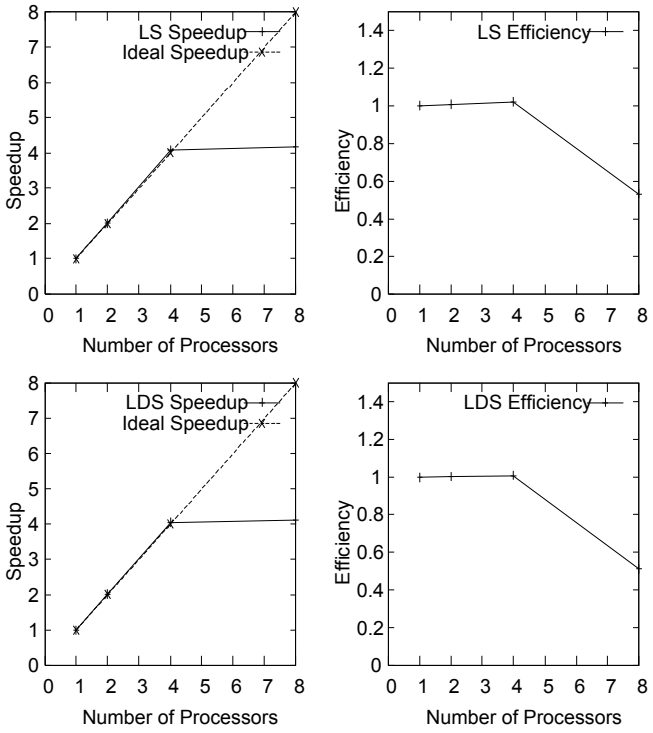

Figure 2. Digital options. Speedup and efficiency of RQMC.

in [3]. Several other researchers tested various simulation techniques by applying them to this problem; see, for example, [20] and [35].

An interesting feature about this problem is that it exposed a defect of a *then* popular pseudorandom number generator, ran(1) of Numerical Recipes in C, First Edition [30]. The generator gives results that do not converge to the correct answer. This was first observed by Paskov [28], and later studied in detail by Tajima et. al. [35]. In Fig. 3 we recreate these results by plotting the price of the security against the sample size. Mersenne twister[3] is a good pseudorandom number generator, and comparing it with ran1() (using three different seeds) we see that ran1() converges to different values.

As observed by Tajima et. al. [35], the defect of ran1() is that it has a cycle that is a multiple of the dimension of the problem, which is 360 in Fig. 3. If the dimension is not a multiple of 360, the defect disappears. For example, in Fig. 4, we consider the same problem using 359 monthly payments. Clearly, ran1() now gives results consistent with Mersenne twister.

[3] The twister is written by Makoto Matsumoto and Takuji Nishimura, and can be found at http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html

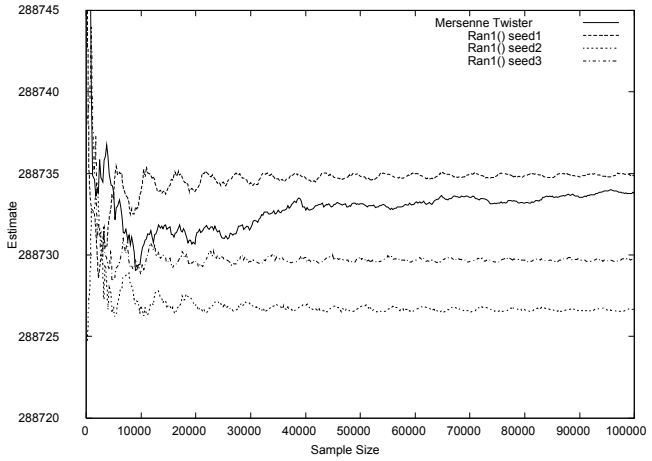

Figure 3. Ran1() with different seeds and Mersenne Twister. 360 months.

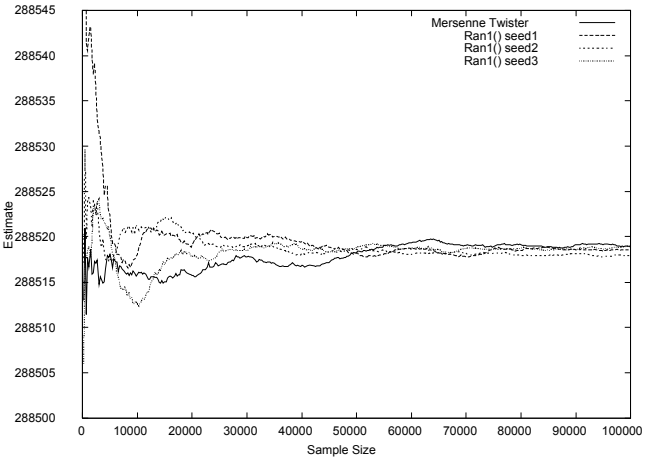

Figure 4. Ran1() with different seeds and Mersenne twister. 359 months.

In Fig. 5, we investigate the speedup and efficiency of the linear scrambling (LS) and linear digital scrambling (LDS) methods against the number of processors. As before, the number of processors are: 1 node & 1 core (1 processor), 1 node & 2 cores (2 processors), 1 node & 4 cores (4 processors), and 2 nodes & 4 cores (8 processors). The results are slightly better than those obtained in the digital options example. In particular, we still observe superlinear speedup up to 4 processors.

We now consider the following question: For a fixed sample size N, what would be the optimal number of processors to obtain an estimate for the price of the security? If we use m processors, then each processor will generate an independent randomization of a QMC sequence, and then use the first N/m vectors of the sequence to compute an estimate. This will give m estimates, and we will compute their average, i.e. the grand estimate, I_m, as the final answer. For example, if $m = 1$, this implies using a single processor and computing one estimate from the first N vectors of one QMC sequence. In this analysis we keep the total sample size, and thus the total number of function evaluations constant. In this setting, we define the MC efficiency of an algorithm as the standard deviation of the grand estimate I_m, multiplied by the time to compute I_m.

In Table 1 we compute the MC efficiency, eff, of the linear scrambling (LS) and linear digital scrambling (LDS) methods. We use $N = 10,000$ and $100,000$ samples and let $m = 1$ (1 node & 1 core), 2 (1 node & 2 cores), and 4 (1 node & 4 cores). We estimate the standard deviation of I_m by computing the sample standard deviation σ of forty estimates. The computing time used in MC efficiency is the time to compute a single grand estimate I_m.

The MC efficiency gets better as m increases, however, the largest improvement in efficiency happens when m is increased from 1 to 2. Linear digit scrambling is consistently better, both in sample standard deviation and efficiency. Perhaps this is not surprising since the collateralized mortgage obligation problem is mainly linear (see [3]).

LS $(10K)$	$m=1$	$m=2$	$m=4$
σ	2.36	1.94	1.80
eff	17.1	7.24	3.44
LDS $(10K)$	$m=1$	$m=2$	$m=4$
σ	1.38	1.02	1.52
eff	9.77	3.62	2.79
LS $(100K)$	$m=1$	$m=2$	$m=4$
σ	0.24	0.24	0.35
eff	17.6	9.04	6.67
LDS $(100K)$	$m=1$	$m=2$	$m=4$
σ	0.20	0.20	0.32
eff	14	7.28	5.62

Table 1: Monte Carlo efficiency

5 Conclusions and future work

We presented a general framework for parameterization methods based on RQMC. Our error bounds apply to any such method, provided that the RQMC method is unbiased. The two parameterization methods we considered have excellent speedup and efficiency especially when the number of processors are small. They also provided superior convergence rates in the digital option example where the true solution is known. Further numerical work comparing all parameterization methods and blocking will be considered

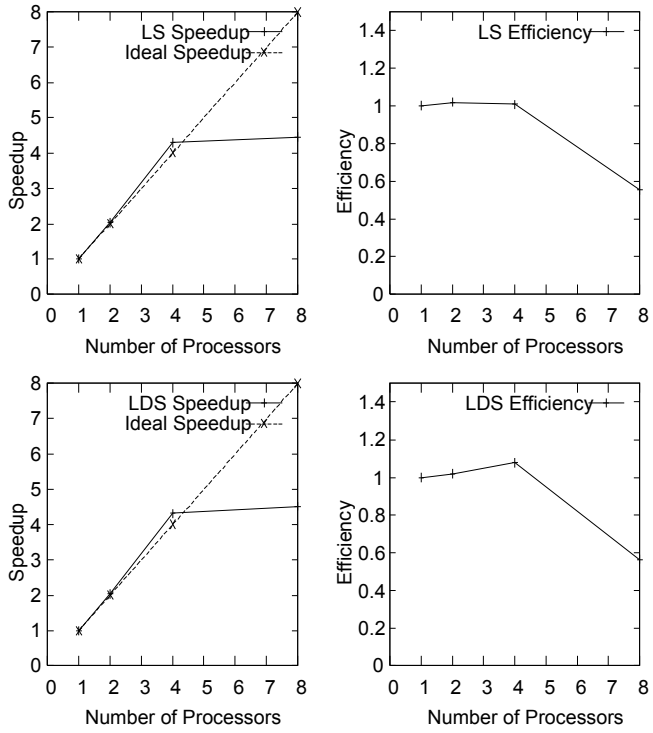

Figure 5. Mortgage problem. Speedup and efficiency of RQMC.

in future research.

References

[1] P. Bratley, B. L. Fox, and H. Niederreiter. Implementation and tests of low-discrepancy sequences. *ACM Transactions on Modeling and Computer Simulation*, 2:195–213, 1992.

[2] B. C. Bromley. Quasirandom number generators for parallel Monte Carlo algorithms. *Journal of Parallel and Distributed Computing*, 38:101–104, 1996.

[3] R. Caflisch, W. Morokoff, and A. Owen. Valuation of Mortgage Backed Securities using Brownian Bridges to Reduce Effective Dimension, *Journal of Computational Finance*, 1:27-46, 1997.

[4] G. Chen, P. Thulasiraman, and R. K. Thulasiram. Distributed Quasi-Monte Carlo Algorithm for Option Pricing on HNOWs Using mpC. In *Proceedings of the 39th Annual Simulation Symposium*, pages 90-97, Huntsville, USA, 2006.

[5] R. Cranley, and T. N. L. Patterson. Randomization of number theoretic methods for multiple integration. *SIAM Journal of Numerical Analysis*, 13:904-914, 1976.

[6] E. De Doncker, R. Zanny, M. Ciobanu, and Y. Guan. Distributed quasi Monte-Carlo methods in a heterogeneous environment. In *Proceedings of the Heterogeneous Computing Workshop 2000*, pages 200–206, 2000.

[7] H. Faure. Good Permutations for Extreme Discrepancy. *Journal of Number Theory*, 42:47–56, 1992.

[8] F. J. Hickernell. The mean square discrepancy of randomized nets. *ACM Transactions on Modeling and Computer Simulation,* 6:274-296, 1996.

[9] F. J. Hickernell, and H. S. Hong. The asymptotic efficiency of randomized nets for quadrature. *Mathematics of Computation* 68:767-791, 1999.

[10] H. Hofbauer, A. Uhl, and P. Zinterhof. Parameterization of Zinterhof Sequences for GRID-based QMC Integration. In J. Volkert, T. Fahringer, D. Kranzlmüller, and W. Schreiner, editors, *Proceedings of the 2nd Austrian Grid Symposium*, books@ocg.at, Innsbruck, Austria, 2007. Austrian Computer Society. To appear.

[11] S. Joe. Randomization of lattice rules for numerical multiple integration. *Journal of Computational and Applied Mathematics,* 31:299-304, 1990.

[12] J. P. Lambert. Quasi-Monte Carlo, low discrepancy sequences, and ergodic transformations. *Journal of Computational and Applied Mathematics,* 12&13:419-423, 1985.

[13] J. X. Li and G. L. Mullen. Parallel computing of a quasi-Monte Carlo algorithm for valuing derivatives. *Parallel Computing*, 26:641–653, 2000.

[14] W. L. Loh. A combinatorial central limit theorem for randomized orthogonal array sampling designs. *Annals of Statistics,* 24:1209-1224, 1996.

[15] M. Mascagni and A. Srinivasan. SPRNG: a scalable library for pseudorandom number generation. *ACM Transactions on Mathematical Software*, 26:436–461, 2000.

[16] J. Matoušek. On the L_2-Discrepancy for Anchored Boxes. *Journal of Complexity,* 14:527-556, 1998.

[17] H. Morohosi, and M. Fushimi. Experimental studies on the error estimation of the numerical integration

by generalized Niederreiter sequences. Operations Research and Its Applications, *Proceedings of the 4th International Symposium, ISORA'02*, World Publishing, pages 46-53, 2002.

[18] H. Morohosi, K. Kobayashi, and M. Fushimi. Estimating the error of quasi-Monte Carlo integrations - Experimental studies in financial problems. Research Institute for Mathematical Sciences, Kyoto University, Kokyuroku, pages 1-11, 2000.

[19] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. SIAM, Philadelphia, 1992.

[20] S. Ninomiya and S. Tezuka. Toward real-time pricing of complex financial derivatives. *Applied Mathematical Finance*, 3:1-20, 1996.

[21] A. B. Owen. Randomly Permuted (t,m,s)-Nets and (t,s)-Sequences. In H. Niederreiter and P.J.-S. Shiue, editors, *Monte Carlo and Quasi-Monte Carlo in Scientific Computing*, Lecture Notes in Statistics, Vol. 106, pages 299-317, Springer, New York, 1995.

[22] A. B. Owen. Monte Carlo variance of scrambled net quadrature. *SIAM Journal on Numerical Analysis*, 34:1884-1910, 1997.

[23] A. B. Owen. Scrambled net variance for integrals of smooth functions. *Annals of Statistics*, 25:1541-1562, 1997.

[24] G. Ökten. Random sampling from low-discrepancy sequences: applications to option pricing. *Mathematical and Computer Modelling*, 35:1221-1234, 2002.

[25] G. Ökten and W. Eastman. Randomized quasi-Monte Carlo methods in pricing securities. *Journal of Economic Dynamics & Control,* 28:2399–2426, 2004.

[26] G. Ökten and A. Srinivasan. Parallel Quasi-Monte Carlo Applications on a Heterogeneous Cluster. In K. T. Fang, F. J. Hickernell and H. Niederreiter, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2000*, pages 406-421, Springer-Verlag, Berlin, 2002.

[27] A. Papageorgiou. The Brownian Bridge Does Not Offer a Consistent Advantage in Quasi-Monte Carlo Integration, *Journal of Complexity*, 18:171-186, 2002.

[28] S. H. Paskov. Computing High Dimensional Integrals with Applications to Finance. Technical Report. Columbia University. 1994.

[29] S. H. Paskov and J. F. Traub. Faster evaluation of financial derivatives. *Journal of Portfolio Management,* 22:113-120, 1995.

[30] W. H. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C*, First Edition, Cambridge University Press, 1998.

[31] W. Schmid and A. Uhl. Parallel Quasi-Monte Carlo integration using (t,s)-sequences. *Lecture Notes in Computer Science*, 1557:96–106, Springer, 1999.

[32] W. Schmid and A. Uhl. Techniques of parallel Quasi-Monte Carlo integration with digital sequences and associated problems. *Mathematics and Computers in Simulation*, 55:249–257, 2001.

[33] A. Srinivasan. Parallel and distributed computing issues in pricing financial derivatives through Quasi Monte Carlo. In *Proceedings of the Sixteenth International Parallel and Distributed Processing Symposium (IPDPS)*, Fort Lauderdale, FL, USA, IEEE Computer Society, 2002.

[34] J. Struckmeier. Fast generation of low-discrepancy sequences. *Journal of Computational and Applied Mathematics,* 61:29-41, 1995.

[35] A. Tajima, S. Ninomiya, and S. Tezuka. On the anomaly of ran1() in Monte Carlo pricing of financial derivatives. In *Proceedings of the 28th conference on Winter simulation*, pages 360-366, Coranado, California, USA, IEEE Computer Society, 1996.

[36] B. Tuffin. On the use of low discrepancy sequences in Monte Carlo methods. *Monte Carlo Methods and Applications* 2:295–320, 1996.

[37] X. Wang and F. J. Hickernell. Randomized Halton Sequences. *Mathematical and Computer Modelling* 32:887–899, 2001.