New Methodologies for Valuing Derivatives

SPASSIMIR H. PASKOV*

Department of Computer Science
Columbia University
New York, NY 10027
paskov@cs.columbia.edu

****

This is a slightly modified version of a Columbia University Technical Report which appeared in October 1994. It will appear in "Mathematics of Derivative Securities" edited by S. Pliska and M. Dempster, Isaac Newton Institute, Cambridge, Cambridge University Press, U.K. in May 1996.

****

This research was supported in part by the the National Science Foundation and the Air Force Office of Scientific Research. I am grateful to Goldman Sachs for providing the finance problem.

*Currently at Union Bank of Switzerland, 299 Park Avenue, New York, NY 10171, email: nypav@ubss.com

Abstract

High-dimensional integrals are usually solved with Monte Carlo algorithms although theory suggests that low discrepancy algorithms are sometimes superior. We report on numerical testing which compares low discrepancy and Monte Carlo algorithms on the evaluation of financial derivatives. The testing is performed on a Collateralized Mortgage Obligation (CMO) which is formulated as the computation of ten integrals of dimension up to 360.

We tested two low discrepancy algorithms (Sobol and Halton) and two randomized algorithms (classical Monte Carlo and Monte Carlo combined with antithetic variables). We conclude that for this CMO the Sobol algorithm is always superior to the other algorithms. We believe that it will be advantageous to use the Sobol algorithm for many other types of financial derivatives.

Our conclusion regarding the superiority of the Sobol algorithm also holds when a rather small number of sample points are used, an important case in practice.

We have built a software system called FINDER for computing high dimensional integrals. Routines for computing Sobol points have been published. However, we incorporated major improvements in FINDER and we stress that the results reported here were obtained using this software.

The software system FINDER runs on a network of heterogeneous workstations under PVM 3.2 (Parallel Virtual Machine). Since workstations are ubiquitous, this is a cost-effective way to do very large computations fast. The measured speedup is at least $.9N$ for N workstations, $N \leq 25$. The software can also be used to compute high dimensional integrals on a single workstation.

Contents

1 Introduction

High-dimensional integrals are usually solved with Monte Carlo algorithms. Vast sums are spent annually on these algorithms.

Theory [21], [43] suggests that low discrepancy algorithms are sometimes superior to Monte Carlo algorithms. However, a number of researchers, [14], [19], [20], report that their numerical tests show that this theoretical advantage decreases with increasing dimension. Furthermore, they report that the theoretical advantage of low discrepancy algorithms disappears for rather modest values of the dimension, say, $d \leq 30$.

We decided to compare the efficacy of low discrepancy and Monte Carlo algorithms on the valuation of financial derivatives. We use a Collateralized Mortgage Obligation (CMO), provided to us by Goldman Sachs, with ten bond classes (tranches) which is formulated as the computation of ten integrals of dimension up to 360. The reasons for choosing this CMO is that it has fairly high dimension and that each integrand evaluation is very expensive making it crucial to sample the integrand as few times as possible. We believe that our conclusions regarding this CMO will hold for many other financial derivatives.

The low discrepancy sample points chosen for our tests are Sobol and Halton points. We compared the algorithms based on these points with the classical Monte Carlo algorithm and also with the classical Monte Carlo algorithm combined with antithetic variables. We refer to the latter as the antithetic variables algorithm. See Section 7 for a discussion of why this algorithm was tested.

An explanation of our terminology is in order here. *Low discrepancy points* are sometimes referred to as *quasi-random* points. Although in widespread use, we believe the latter term to be misleading since there is nothing random about these deterministic points. We prefer to use the terminology *low discrepancy* or *deterministic*.

We assume the finance problem has been formulated as an integral over the unit cube in d dimensions. We have built a software system called FINDER for computing high dimensional integrals. FINDER runs on a heterogeneous network of workstations under PVM 3.2 (Parallel Virtual Machine). Since workstations are ubiquitous, this is a cost-effective way to do large computations fast. Of course, FINDER can also be used to compute high dimensional integrals on a single workstation. The software system FINDER is available and interested readers should contact the author.

A routine for generating Sobol points is given in [28]. *However, major improvements have been incorporated in FINDER. We emphasize that the results reported in this paper were obtained using FINDER.* One of the improvements was developing the table of primitive polynomials and initial direction numbers for dimensions up to 360.

This paper is based on software construction and testing which began in the Fall of 1992.

Preliminary results were presented to a number of New York City financial houses in the Fall of 1993 and the Spring of 1994. A January, 1994 article in Scientific American discussed the theoretical issues and reported that "Preliminary results obtained by testing certain finance problems suggest the superiority of the deterministic methods in practice." Further results were reported at a number of conferences including [24], [38], [39]. An "extended abstract" of this paper was published in Fall, 1995 [26]. A slightly different version of this paper appeared as a Columbia University Technical Report in October, 1994.

We summarize our main conclusions regarding the evaluation of this CMO. The conclusions may be divided into three groups.

I. Deterministic and Monte Carlo Algorithms

The Sobol algorithm consistently outperforms the Monte Carlo algorithm. The Sobol algorithm consistently outperforms the Halton algorithm. In particular,

- The Sobol algorithm converges significantly faster than the Monte Carlo algorithm;
- The convergence of the Sobol algorithm is smoother than the convergence of the Monte Carlo algorithm. This makes automatic termination easier for the Sobol algorithm;
- Using our standard termination criterion, see Section 6, the Sobol algorithm terminates 2 to 5 times faster than the Monte Carlo algorithm often with smaller error;
- The Monte Carlo algorithm is sensitive to the initial seed.

II. Sobol, Monte Carlo, and Antithetic Variables Algorithms

The Sobol algorithm consistently outperforms the antithetic variables algorithm, which in turn, consistently outperforms the Monte Carlo algorithm. In particular,

- These conclusions also hold when a rather small number of sample points are used, an important case in practice. For example, for 4000 sample points, the Sobol algorithm running on a single Sun-4 workstation achieves accuracies within range from one part in a thousand to one part in a million, depending on the tranche, within a couple of minutes;
- Statistical analysis on the small sample case further strengthens the case for the Sobol algorithm over the antithetic variables algorithm. For example, to achieve similar performances with confidence level 95%, the antithetic variables algorithm needs from 7 to 79 times more sample points than the Sobol algorithm, depending on the tranche. In a similar comparison, the Monte Carlo algorithm needs from 27 to 607 times more sample points than the Sobol algorithm depending on the tranche. These speedups are measured conservatively, see Section 9;

- Statistical analysis for a large number of sample points (1,000,000 points) shows that the superiority of the Sobol algorithm is greater than for a small number of points. For example, to achieve similar performances with confidence level 95%, the antithetic variables algorithm needs from 16 to 230 times more sample points than the Sobol algorithm, depending on the tranche. In a similar comparison, the Monte Carlo algorithm needs from 110 to 1769 times more sample points than the Sobol algorithm depending on the tranche. These speedups are measured conservatively, see Section 9;
- The antithetic variables algorithm is sensitive to the initial seed. However, convergence of the antithetic variables algorithm is less jagged than convergence of the Monte Carlo algorithm.

III. Network of Workstations *All the algorithms benefit by being run on a network of workstations.* In particular,

- For N workstations, the measured speedup is at least $0.9N$, where $N \leq 25$;
- A substantial computation which took seven hours on a Sun-4 workstation took twenty minutes on the network of 25 workstations.

We emphasize that we do not claim that the Sobol algorithm is always superior to the Monte Carlo algorithm. We do not even claim that it is always superior for financial derivatives. After all, the test results reported here are only for one particular CMO. However, we do believe it will be advantageous to use the Sobol algorithm for many other types of financial derivatives.

2 The Monte Carlo Algorithm

In this section, we give a brief discussion of the theory underlying the Monte Carlo algorithm for the problem of multivariate integration. For more details, see for example, [11], [12], [15], and [34].

We now present the classical Monte Carlo algorithm. For brevity, we refer to it as the Monte Carlo algorithm. Let $t_1, \ldots, t_n$ be n randomly selected points which are independent and uniformly distributed over $D = [0,1]^d$, the d dimensional unit cube. Consider a function f from the space $L_2(D)$ of L_2-integrable functions. The problem is to approximately compute

$$I(f) = \int_D f(x)dx$$

using function evaluations at randomly chosen points. The classical Monte Carlo algorithm is given by

$$I(f) \approx U_n(f) = U_n(f; t_1, \ldots, t_n) = \frac{1}{n}\sum_{i=1}^{n} f(t_i). \tag{1}$$

The main idea underlying the Monte Carlo algorithm for multivariate integration is to replace a continuous average by a discrete average over randomly selected points.

The expected value of the estimate $U_n(f; t_1, \ldots, t_n)$ as a function of the random variables $t_1, t_2, \ldots, t_n$ is

$$E(U_n(f; t_1, \ldots, t_n)) = I(f).$$

The expected error of the Monte Carlo algorithm for a function f is defined by

$$E_n(f) = \left(\int_{D^n} (I(f) - U_n(f))^2 \, dt_1 \ldots dt_n \right)^{1/2}.$$

It is well known that

$$E_n(f) = \frac{\sigma(f)}{\sqrt{n}},$$

where the variance $\sigma^2(f)$ of f is defined as

$$\sigma^2(f) = \int_D (f(t) - I(f))^2 \, dt = I(f^2) - I^2(f).$$

Clearly, by reducing the variance of the integrand the expected error would also decrease. In fact, this is the main idea underlying the various variance reduction techniques which are often used in combination with the Monte Carlo algorithm. Examples of variance reduction techniques are importance sampling, control variates, antithetic variables, see for example [15]. Antithetic variables will be discussed in Section 7.

Let $B(L_2(D))$ denote the unit ball of $L_2(D)$. The expected error of the Monte Carlo algorithm with respect to the class $B(L_2(D))$ is measured for the worst function f in $B(L_2(D))$,

$$e_n^{\mathrm{wor-MC}}(B(L_2(D)) = \sup_{f \in B(L_2(D))} E_n(f).$$

It can be easily concluded that

$$e_n^{\mathrm{wor-MC}}(B(L_2(D)) = n^{-1/2}. \tag{2}$$

Since the rate of convergence is independent of d and the Monte Carlo algorithm is applicable to a very broad class of integrands, its advantages for high dimensional integration are clear. Nevertheless, the Monte Carlo algorithm has several serious deficiencies, see for example, [21] and [36]. We mention just three of them here. Even though the rate of convergence is independent of the dimension, it is quite slow. Furthermore, there are fundamental philosophical and practical problems with generating independent random points; instead, pseudorandom numbers are used, see [16] and [41]. Finally, the Monte Carlo algorithm provides only probabilistic error bounds, which is not a desirable guarantee for problems where highly reliable results are needed.

3 Low Discrepancy Deterministic Algorithms

In an attempt to avoid the deficiencies of the Monte Carlo algorithm, many deterministic algorithms have been proposed for computing high dimensional integrals for functions belonging to various subsets of $L_2(D)$, see e.g. [21], [22], [40], [43], [23], and [37]. One class of such deterministic algorithms is based on low discrepancy sequences. First, we define discrepancy, which is a measure of deviation from uniformity of a sequence of points in D. Then, very briefly, we give the theoretical basis of the low discrepancy sequences to be used as sample points for computing multivariate integrals. For more detailed description and treatment of these results, see [21].

For $t = [t_1, \dots, t_d] \in D$, define $[0, t) = [0, t_1) \times \cdots \times [0, t_d)$. Let $\chi_{[0,t)}$ be the characteristic (indicator) function of $[0, t)$. For $z_1, \dots, z_n \in D$, define

$$R_n(t; z_1, \dots, z_n) = \frac{1}{n} \sum_{k=1}^{n} \chi_{[0,t)}(z_k) \; - \; t_1 t_2 \cdots t_d.$$

The L_2 (or L_∞) *discrepancy* of $z_1, \dots, z_n$, is defined as the L_2 (or L_∞) norm of the function $R_n(\cdot\, ; z_1, \dots, z_n)$, i.e.,

$$\begin{aligned} \|R_n(\cdot\, ; z_1, \dots, z_n)\|_2 &= \left(\int_D R_n^2(t; z_1, \dots, z_n)\, dt \right)^{1/2}, \\ \|R_n(\cdot\, ; z_1, \dots, z_n)\|_\infty &= \sup_{t \in D} |R_n(t; z_1, \dots, z_n)|. \end{aligned}$$

Roth, [29] and [30], proves that

$$\inf_{z_1, \dots, z_n} \|R_n(\cdot\, ; z_1, \dots, z_n)\|_2 = \Theta(n^{-1} (\log n)^{(d-1)/2}). \tag{3}$$

Of special interest for numerical integration are infinite low discrepancy sequences $\{z_k\}$ in which the definition of z_k does not depend on the specific value of the number of points n.

Examples of such infinite sequences are the Halton [9] and Sobol [33] sequences. For the reader's convenience, we give a short description of both these sequences later in this section. The following bounds hold for the Halton and the Sobol sequences

$$\|R_n(\cdot; z_1, \dots, z_n)\|_2 \le \|R_n(\cdot; z_1, \dots, z_n)\|_\infty = O\left(\frac{(\log n)^d}{n}\right). \tag{4}$$

This is the best upper bound known and it is widely believed that it is sharp for these sequences, see [21].

We stress that the constants in the bounds (3) and (4) depend on the dimension d and good estimates of these constants are not known. Bounds with known constants and n independent of d are studied in [42] and [44].

Sequences satisfying the upper bound in (4) are known as *low discrepancy sequences* or *quasi-random sequences*. We will refer to them as *low discrepancy sequences* or *deterministic sequences*.

Clearly, the idea behind the low discrepancy sequences is for any rectangular $[0, t)$ the fraction of the points within $[0, t)$ to be as "close" as possible to its volume. That way, the low discrepancy sequences cover the unit cube as "uniformly" as possible by reducing gaps and clustering of points. This idea is illustrated in Figure 1 and Figure 2.

We now state the theoretical bases for the use of low discrepancy sequences as sample points for multivariate integration. Let $V(f) < \infty$ be the variation of f on D in the sense of Hardy and Krause, see [21], and let $\{z_k\}$ be a low discrepancy sequence. The *Koksma-Hlawka inequality* guarantees that

$$\left| I(f) - \frac{1}{n}\sum_{k=1}^{n} f(z_k) \right| \le V(f)\ \|R_n(\cdot\,; z_1, \dots, z_n)\|_\infty. \tag{5}$$

Upper bounds in terms of L_2 discrepancy have also been proven, see [21]. Therefore, (4) and (5) provide a worst case assurance for the use of low discrepancy sequences as sample points for numerical integration of functions with bounded variation in the sense of Hardy and Krause. Furthermore, we stress that the deterministic algorithms, based on low discrepancy sequences, have better rates of convergence than the Monte Carlo algorithm.

In addition, low discrepancy sequences also have good properties in the average case setting. Indeed, let $F = C_d$ be the class of real continuous functions defined on D and equipped with the classical Wiener sheet measure w. That is, w is Gaussian with mean zero and covariance kernel R_w defined as

$$R_w(t, x) \stackrel{\text{def}}{=} \int_{C_d} f(t) f(x) w(df) = \min(t, x) \stackrel{\text{def}}{=} \prod_{j=1}^{d} \min(t_j, x_j),$$

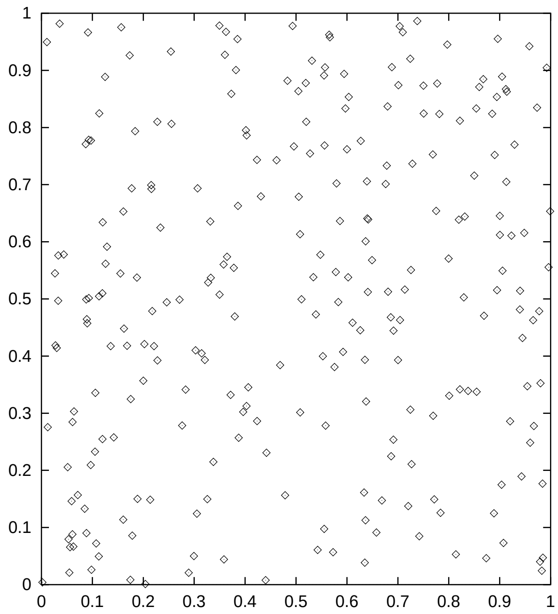

Figure 1: 256 Random points in the unit square

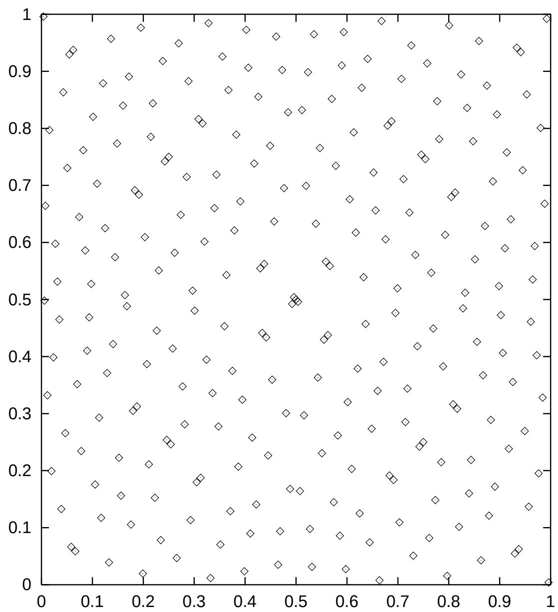

Figure 2: 256 Sobol points in the unit square

where $t = (t_1, \ldots, t_d)$, $x = (x_1, \ldots, x_d)$ for $t, x \in D$.

Define

$$x_k = 1 - z_k, \qquad k = 1, 2, \ldots,$$

where $\{z_k\}$ is a low discrepancy sequence. We approximate the integral of f from C_d by the arithmetic mean of its values at x_k,

$$I(f) = \int_D f(t)dt \approx U_n(f) = \frac{1}{n} \sum_{k=1}^{n} f(x_k), \qquad \forall f \in C_d.$$

Woźniakowski [43] relates the average case error of $U_n(f)$ with the L_2 discrepancy,

$$\int_{C_d} (I(f) - U_n(f))^2 w(df) = \int_D R_n^2(t; z_1, \ldots, z_n) dt. \tag{6}$$

Thus, (3), (4), and (6) provide an average case assurance for the use of the sequence $\{x_k\}$ as a set of sample points for numerical integration of the functions in C_d equipped with the classical Wiener sheet measure. Using the identity $\chi_{[0,t)}(z_k) = \chi_{(1-t,1]}(x_k)$ and Proposition 2.4 in [21], we conclude that if $\{z_k\}$ is a low discrepancy sequence then $\{x_k\}$ is also a low discrepancy sequence.

3.1 Halton Points

We give a short description of the Halton low discrepancy sequence.

Let $p_1, p_2, \ldots, p_d$ be the first d prime numbers. Any integer $k \geq 0$ can be uniquely represented as $k = \sum_{i=0}^{\lceil \log k \rceil} a_i p_j^i$ with integers $a_i \in [0, p_j - 1]$. The radical inverse function ϕ_{p_j} is defined as

$$\phi_{p_j}(k) = \sum_{i=0}^{\lceil \log k \rceil} a_i p_j^{-i-1}. \tag{7}$$

The Halton d-dimensional points are defined as

$$z_k = (\phi_{p_1}(k), \phi_{p_2}(k), \ldots, \phi_{p_d}(k)), \qquad k \geq 0.$$

3.2 Sobol Points

We give a short description of the Sobol low discrepancy sequence.

Assume first that $d = 1$. A one-dimensional Sobol sequence is generated as follows. For $i = 1, 2, \ldots, w$, let $v_i = m_i / 2^i$ be a sequence of binary fractions with w bits after the binary point, where $0 < m_i < 2^i$ are odd integers. The numbers v_i are called *direction numbers*. Assume for a moment that they are already generated; we will discuss their generation later.

In Sobol's original algorithm, a one-dimensional Sobol sequence is generated by

$$x_k = b_1 v_1 \oplus b_2 v_2 \oplus \cdots \oplus b_w v_w, \qquad k \geq 0$$

where $k = \sum_{i=0}^{\lceil \log k \rceil} b_i 2^i$ is the binary representation of k and $\oplus$ denotes a bit-by-bit exclusive-or operation. For example, $k = 3$ is 11 in base 2. If $v_1 = 0.1$ and $v_2 = 0.11$ then

$$b_1 v_1 \oplus b_2 v_2 = 0.1 \oplus 0.11 = 0.01.$$

Antonov and Saleev [2] suggest a shuffling of the original Sobol sequence which preserves good convergence properties and which can be generated much faster. This version of the Sobol sequence is used here.

The sequence of direction numbers v_i is generated by a primitive polynomial, see [16] and [33], with coefficients in the field Z_2 with elements $\{0, 1\}$. Consider, for example, the primitive polynomial

$$P(x) = x^n + a_1 x^{n-1} + \ldots + a_{n-1} x + 1$$

of degree n in $Z_2[x]$. Then the direction numbers are obtained from the following recurrence formula

$$v_i = a_1 v_{i-1} \oplus a_2 v_{i-2} \oplus \cdots \oplus a_{n-1} v_{i-n+1} \oplus v_{i-n} \oplus (v_{i-n}/2^n), \qquad i > n,$$

where the last term v_{i-n} is shifted right n places. The initial numbers $v_1 = m_1/2, \ldots, v_n = m_n/2^n$ are such that m_i is odd and $0 < m_i < 2^i$ for $i = 1, 2, \ldots, n$.

Consider now an arbitrary $d \geq 1$. Let $P_1, P_2, \ldots, P_d$ be d primitive polynomials in $Z_2[x]$. Denote by $\{x_k^i\}_{k=1}^{\infty}$ the sequence of one-dimensional Sobol points generated by the polynomial P_i. Then the sequence of d-dimensional Sobol points is defined as

$$x_k = (x_k^1, x_k^2, \ldots, x_k^d).$$

4 Software System for Computing High Dimensional Integrals

4.1 Algorithms

The theory presented in the previous sections suggests that the low discrepancy deterministic algorithms provide an interesting alternative to the Monte Carlo algorithm for computing high dimensional integrals. We have developed and tested a distributed software system for computing multivariate integrals on a network of workstations. The deterministic algorithms and the Monte Carlo algorithm are implemented. The software utilizes the following sequences of sample points:

- Halton points;
- Sobol points;
- Uniformly distributed random points.

The user can choose the sequence of sample points from a menu. The software is written in a modular way so other kinds of deterministic and random number generators can be easily added. One or several multivariate functions defined over the unit cube of up to 360 variables can be integrated simultaneously. The number of variables could be extended as well.

A routine for generating Sobol points is given in [28]. However, we have made major improvements and we stress that the results reported in this paper were obtained using FINDER and not the routine in [28]. One of the improvements was developing the table of primitive polynomials and initial direction numbers for dimensions up to 360.

The software uses various kinds of random number generators. More specifically, the random number generators ran1 and ran2 from the first edition of Numerical Recipes [27], and RAN1 and RAN2 from the second edition of Numerical Recipes [28] are used because of their wide availability and popularity. All of the above random number generators are based on linear congruential generators with some additional features. For more details on these random number generators we refer to [27] and [28].

4.2 Systems

Since workstation clusters and networks provide cost-effective means to perform large-scale computation, we have built and debugged a software system under PVM 3.2 (Parallel Virtual Machine) for computing multivariate integrals. This system runs on a heterogeneous network of machines. PVM is a software package that allows a network of heterogeneous Unix computers to be used as a single large parallel computer. Thus large computational problems can be solved by using the aggregate power of many computers.

A master/slave model is used as the programming paradigm for computing multivariate integrals. In this model, the master program spawns and directs some number of slave processes. Each of the slave processes generates a sequence of sample points specified by the master and evaluates the integrand at those points. A partial sum of integrand values is returned to the master by each of the slave processes. The master combines partial sums as specified by an algorithm, computes an approximate value of the integral, and checks a termination criterion. If the termination criterion is not satisfied, the master spawns a new round of computations. This process continues until the termination criterion is satisfied or some prespecified upper bound of the number of sample points is reached. At the end, the master returns the final result of computation and timing information. In addition, the master process keeps information about each spawned process. If some host dies, the master reallocates its job to some other host.

In a multiuser network environment the load balancing method can be one of the most important factors for improving the performance. A dynamic load balancing method is used. Namely, the Pool of Tasks paradigm is especially suited for a master/slave program. In the Pool of Tasks method the master program creates and holds the "pool" and sends out tasks to slave programs as they become idle. The fact that no communication is required between slave programs and the only communication is to the master makes our integration problem suitable for the Pool of Tasks paradigm.

5 Finance (CMO) Problem

We now consider a finance problem which is a typical Collateralized Mortgage Obligation (CMO). A CMO consists of several bond classes, commonly referred to as tranches, which derive their cash flows from an underlying pool of mortgages. The cash flows received from the pool of mortgages are divided and distributed to each of the tranches according to a set of prespecified rules. The cash flows consist of interest and repayment of the principal. The technique of distributing the cash flows transfers the prepayment risk among different tranches. This results in financial instruments with varying characteristics which might be more suitable to the needs and expectations of investors. For more details on CMOs, we refer to [5]. We stress that the amount of obtained cash flows will depend upon the future level of interest rates. Our problem is to estimate the expected value of the sum of present values of future cash flows for each of the tranches.

We now give some of the details related to the studied CMO and the way it is reduced to the problem of multivariate integration. The CMO[1] consists of ten tranches. Denote them by A, B, C, D, E, G, H, J, R, Z. Throughout this section, we describe results on tranche A in more details. Results for other tranches are similar unless stated explicitly. The underlying pool of mortgages has a thirty-year maturity and cash flows are obtained monthly. This

[1]It is labeled as "Fannie Mae REMIC Trust 1989-23".

implies 360 cash flows. The monthly cash flows are divided and distributed according to some prespecified rules. The actual rules for distribution are rather complicated and are given in details in the prospectus describing the financial product.

For $1 \le k \le 360$, denote by

$$\begin{aligned} C &\quad - \quad \text{the monthly payment on the underlying pool of mortgages;} \\ i_k &\quad - \quad \text{the appropriate interest rate in month } k; \\ w_k &\quad - \quad \text{the percentage prepaying in month } k; \\ a_{360-k+1} &\quad - \quad \text{the remaining annuity after month } k. \end{aligned}$$

Recall that the remaining annuity a_k is given by

$$a_k = 1 + v_0 + \cdots + v_0^{k-1}, \qquad k = 1, 2, \ldots, 360,$$

with $v_0 = 1/(1 + i_0)$ and i_0 the current monthly interest rate. In the notation above, C and $a_{360-k+1}$ are constants; i_k and w_k are stochastic variables to be determined below.

We now describe the interest rate model. Assume that the interest rate i_k is of the form

$$i_k = K_0 e^{\xi_k}\, i_{k-1} = K_0^k\, i_0\, e^{\xi_1 + \cdots + \xi_k},$$

where $\{\xi_k\}_{k=1}^{360}$ are independent normally distributed random variables with mean 0 and variance σ^2, and K_0 is a given constant. In our case $\sigma^2 = 0.0004$ is chosen.

Suppose that the prepayment model w_k, as a function of i_k, is computed as

$$\begin{aligned} w_k &= w_k(\xi_1, \ldots, \xi_k) = K_1 + K_2 \arctan(K_3 i_k + K_4) \\ &= K_1 + K_2 \arctan(K_3 K_0^k\, i_0\, e^{\xi_1 + \cdots + \xi_k} + K_4), \end{aligned}$$

where K_1, K_2, K_3, K_4 are given constants.

The cash flow in month k, $k = 1, 2, \ldots, 360$, is

$$\begin{aligned} M_k &= M_k(\xi_1, \ldots, \xi_k) \\ &= C(1 - w_1(\xi_1)) \cdots (1 - w_{k-1}(\xi_1, \ldots, \xi_{k-1}))(1 - w_k(\xi_1, \ldots, \xi_k) + w_k(\xi_1, \ldots, \xi_k) a_{360-k+1}). \end{aligned}$$

This cash flow is distributed to the tranches according to the rules of the CMO under consideration. Let $G_{k;T}(\xi_1, \ldots, \xi_k)$ be the portion of the cash flow M_k for month k directed to the tranche T. The form of the function $G_{k;T}$ is very complex. Here, it suffices to say that it is a continuous function which is a composition of min functions and smooth functions. By min function we mean a function which is the minimum of two functions.

To find the present value of the tranche T for month k, $G_{k;T}(\xi_1, \ldots, \xi_k)$ has to be multiplied by the discount factor

$$u_k(\xi_1, \ldots, \xi_{k-1}) = v_0 v_1(\xi_1) \cdots v_{k-1}(\xi_1, \ldots, \xi_{k-1}),$$

with

$$v_j(\xi_1, \ldots, \xi_j) = \frac{1}{1 + i_j(\xi_1, \ldots, \xi_j)} = \frac{1}{1 + K_0^j i_0 e^{\xi_1 + \cdots + \xi_j}}, \qquad j = 1, 2, \ldots, 359.$$

Summing up the present values for every month k, $k = 1, 2, \ldots, 360$, for tranche T will give us the present value PV_T,

$$PV_T(\xi_1, \ldots, \xi_{360}) = \sum_{k=1}^{360} G_{k;T}(\xi_1, \ldots, \xi_k) u_k(\xi_1, \ldots, \xi_{k-1}).$$

We want to compute the expected value $E(PV_T) = E(PV_T(\xi_1, \ldots, \xi_{360}))$. By change of variables, it is easy to see that

$$E(PV_T) = \int_{[0,1]^{360}} PV_T(y_1(x_1), \ldots, y_{360}(x_{360}))\, dx_1 \cdots dx_{360},$$

where $y_i = y_i(x_i)$ is implicitly given by

$$x_i = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{y_i} e^{-t^2/(2\sigma)} dt. \tag{8}$$

Therefore, our problem is reduced to a problem of computing ten multivariate integrals over the 360-dimensional unit cube. We stress that after generating a point $(x_1, \ldots, x_{360})$ in the 360-dimensional unit cube, the point $(y_1, \ldots, y_{360})$ has to be computed by finding the value of the inverse normal cumulative distribution function at each $x_k, k = 1, 2, \ldots, 360$. We use software available through NETLIB to compute the points $(y_1, \ldots, y_{360})$. Then the function values $PV_T(y_1, \ldots, y_{360})$ are computed for all tranches T.

We now discuss some of the smoothness properties of the integrand PV_T, where T is one of the tranches. Since the change of variables in (8) is based on smooth functions we may restrict our discussion to PV_T as a function of $y_1, \ldots, y_{360}$. Recall that

$$PV_T(y_1, \ldots, y_{360}) = \sum_{k=1}^{360} G_{k;T}(y_1, \ldots, y_k) u_k(y_1, \ldots, y_{k-1}).$$

Clearly, u_k is a smooth function. As mentioned before, $G_{k;T}$ is a composition of min functions and smooth functions. That is why we believe that the function PV_T has finite variation in the sense of Hardy and Krause and, that is why we decided to use low discrepancy sequences for these type of integrands. Of course, it would be desirable to have a good bound on the variation of PV_T. Unfortunately, due to the complex form of PV_T, this is a very difficult task.

As reported in the Introduction, it is widely believed that the theoretical advantage of low discrepancy sequences degrades with increase of dimension and low discrepancy sequences are effective, in general, only for $d \leq 30$. Although the CMO problem is apparently of dimension 360, some of the tranches are of lower dimension. To understand the performance of the different algorithms for the CMO problem it is important to know the dimensions of the various tranches.

The integrand PV_T of tranche T depends on the cash flows $G_{k;T}$. The cash flow $G_{k;T}$ has an important property. If the tranche T is retired at month k_T, i.e., the whole principal is paid off, then $G_{k;T}$ is zero for month $k > k_T$. More precisely,

$$G_{k;T}(\xi_1, \dots, \xi_k) = 0 \qquad \forall k > k_T = k_T(\xi_1, \dots, \xi_{k_T})$$

with $\xi_1, \dots, \xi_k$ corresponding to a particular interest rate scenario.

We checked computationally that k_T is typically smaller than 360. We generated $n = 3,000,000$ random sample points and determined the maximum k_T for each tranche T. These numbers are reported in Table 1. As we see, the dimension of only one tranche is 360. However, for the other tranches, the dimension is still high.

The concept of the dimension of a tranche can be further relaxed. Intuitively, if the dependence of PV_T is negligible on the variables $x_{k+1}, x_{k+2}, \dots, x_{360}$ then only the first k variables $x_1, x_2, \dots, x_k$ contribute substantially to the integral of PV_T. This concept of *effective dimension* can be defined rigorously as follows.

Let f be a function with domain $[0,1]^d$ and let $\varepsilon > 0$. The *effective dimension* $K(\varepsilon)$ of f is the smallest integer $k \in [1, d]$ such that

$$\left| \int_{[0,1]^k} f(x_1, \dots, x_k, 0, \dots, 0)\, dx_1 \cdots dx_k - \int_{[0,1]^d} f(x) dx \right| \leq \varepsilon \left| \int_{[0,1]^d} f(x) dx \right|.$$

We apply this definition of effective dimension to the ten tranches of the CMO problem. Clearly, the effective dimension is a function of ε. What value of ε should we choose? In

Tranche	Maximum of k_T
A	189
B	250
C	278
D	298
E	309
G	77
H	91
J	118
R	360
Z	167

Table 1: The maximum of the numbers k_T obtained by generating 3,000,000 random sample points for each tranche of the CMO problem

practice, the CMO problem does not have to be solved with high accuracy, see also the discussion in Section 8, and so we choose $\varepsilon = 0.001$. We checked computationally that the values of the integrals of all tranches are about $a10^6$ with $a \in [2, 42]$. Hence by taking $\varepsilon = 0.001$ we introduce an absolute error which is on the order of several thousand dollars.

We estimate $K_T(\varepsilon)$ for $\varepsilon = 0.001$, where $K_T(\varepsilon)$ denotes the effective dimension of tranche T. To compute $K_T(\varepsilon)$ we need a fairly accurate approximation of the integral $I(PV_T) = \int_{[0,1]^{360}} PV_T(x)dx$. We approximate $I(PV_T)$ by using 20,000,000 sample points as explained in Section 8. Using these results we approximately compute the effective dimensions for each of the tranches. The results are summarized in Table 2. As we see, the effective dimension, although smaller, is still high for most of the tranches.

6 Comparison of the Deterministic and Monte Carlo algorithms

We now present the results of extensive testing of the deterministic and Monte Carlo algorithms for the CMO problem. For the reader's convenience, the results are summarized in a number of graphs.

Figure 3 shows the results for tranche A of Sobol, Halton, and Monte Carlo runs with two randomly chosen initial seeds. The pseudorandom generator RAN2 from [28] is used to

Tranche	$K_T(\varepsilon)$
A	131
B	175
C	212
D	239
E	261
G	42
H	63
J	84
R	338
Z	114

Table 2: The approximate values of $K_T(\varepsilon)$ for $\varepsilon = 0.001$

generate random sample points for the Monte Carlo runs. Figure 3 exhibits behavior that is common in all our comparisons of the Sobol algorithm with the Monte Carlo algorithm. For the Halton algorithm see the discussion at the end of this section.

- The Monte Carlo algorithm is sensitive to the initial seed;
- The deterministic algorithms, especially the Sobol algorithm, converge significantly faster than the Monte Carlo algorithm;
- The convergence of the deterministic algorithms, especially of the Sobol algorithm, is smoother than the convergence of the Monte Carlo algorithm. This makes automatic termination easier for the Sobol algorithm; see the discussion below;
- The Sobol algorithm outperforms the Halton algorithm.

Figure 4 shows the same Sobol and Halton runs but a different random number generator, namely ran1 from [27], is used to generate four Monte Carlo runs using four randomly chosen initial seeds. The plot again exhibits sensitivity of the Monte Carlo algorithm to the initial seed. The plot also indicates that there are some problems with this random number generator since the Monte Carlo results seem to lie on horizontal lines when the number of sample points exceeds 300,000 and it seems unlikely that they will converge to the same value. This claim is also supported by the fact that the same effect has been observed for sixteen additional Monte Carlo runs. We assume that is why ran1 has been replaced by a

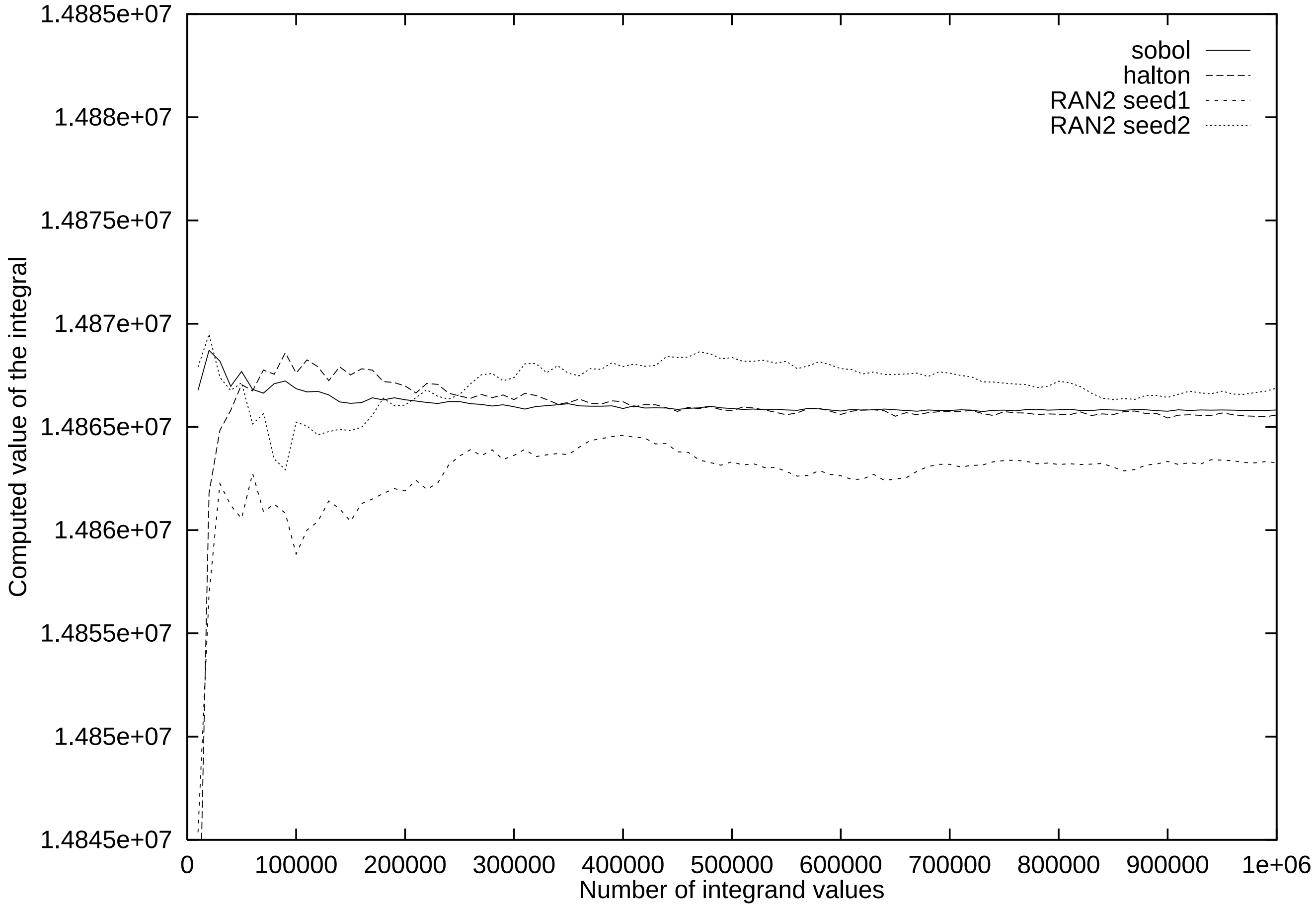

Figure 3: Sobol and Halton runs for tranche A and two Monte Carlo runs using RAN2

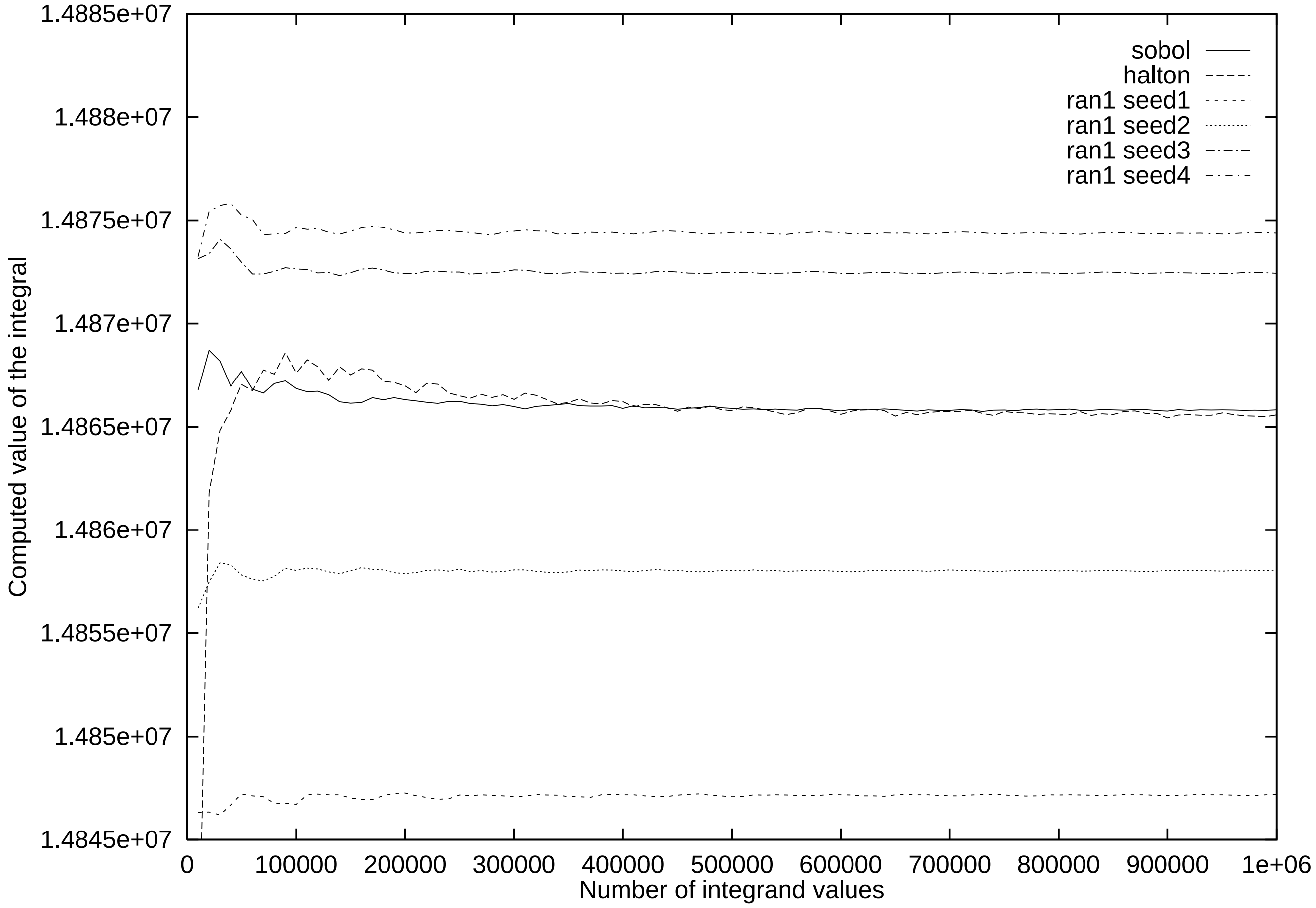

Figure 4: Sobol and Halton runs for tranche A and four Monte Carlo runs using ran1

different random number generator in [28]. Figure 4 is included to show that the results can be affected by the poor performance of the random number generator.

Figure 5 plots the same Sobol and Halton runs versus the arithmetic mean of twenty Monte Carlo runs using RAN2 from [28]. We stress that the number of sample points on the x-axis is correct only for the deterministic algorithms. The actual number of sample points for the averaged Monte Carlo graph is twenty times the number of sample points on the x-axis. The results of the deterministic algorithms and the averaged Monte Carlo result are approximately the same. We thus conclude that to achieve similar performances, we have to take about 20 times more random than deterministic sample points.

In Figure 6, an automatic termination criterion is applied to Sobol, Halton, and three

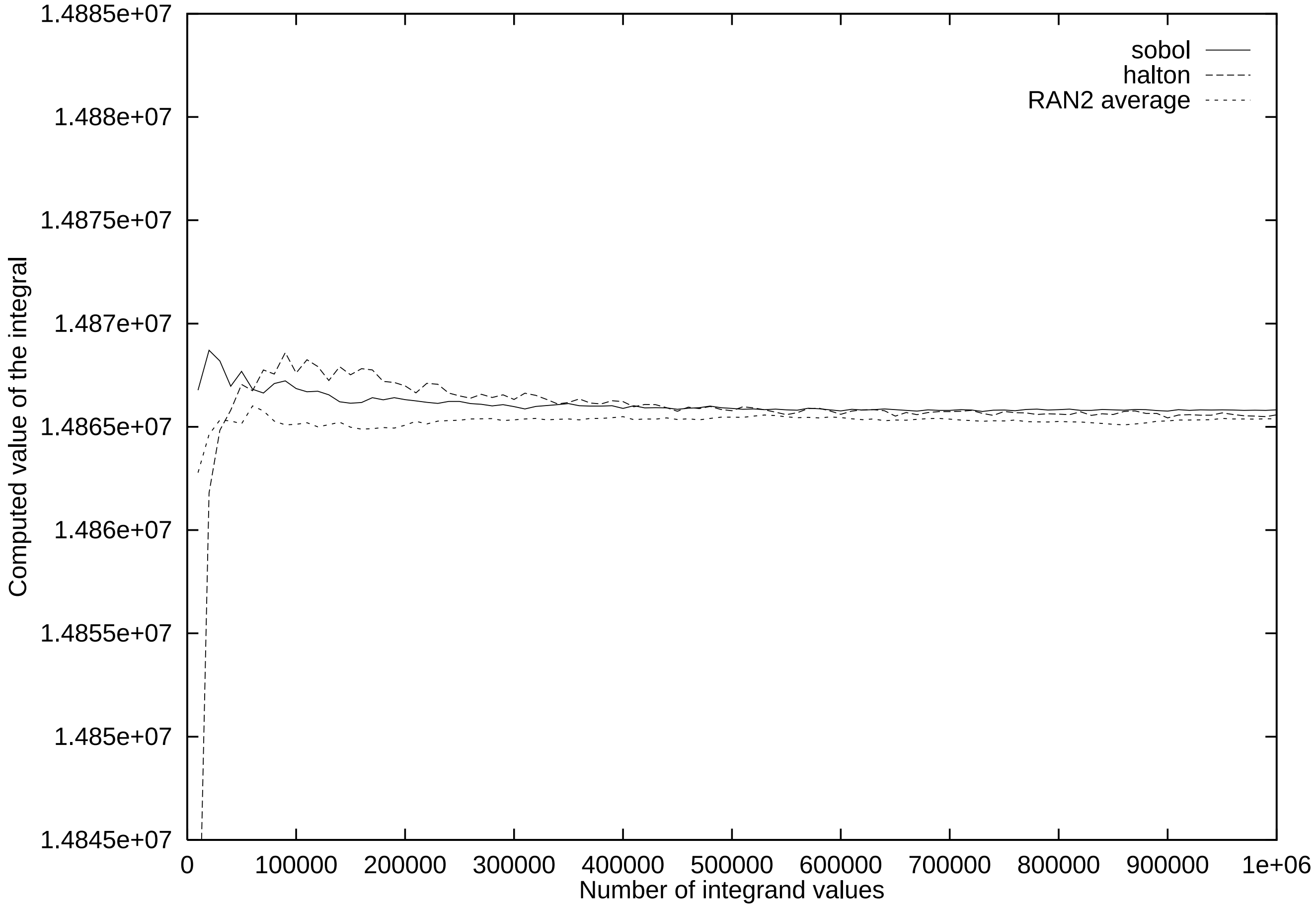

Figure 5: Sobol and Halton runs for tranche A and an average of twenty Monte Carlo runs using RAN2

Monte Carlo runs using RAN2 from [28] as the pseudorandom generator. We choose a standard automatic termination criterion. Namely, when two consecutive differences between consecutive approximations using $10,000\,i$, $i = 1, 2, \ldots, 100$, sample points become less than some threshold value for all of the tranches of the CMO, the computational process is terminated. With the threshold value set at 250, the Sobol run terminates at 160,000 sample points, the Halton run terminates at 700,000 sample points, and the three Monte Carlo runs terminate at 410,000, 430,000, and 780,000 sample points, respectively. Hence, the Sobol run terminates 2 to 5 times faster than the Monte Carlo runs.

We also stress that even though the Sobol algorithm terminates faster, the result is more accurate than two of the results achieved by the Monte Carlo algorithm. In Section 8, we show that the value of the integral for tranche A by using 20,000,000 sample points is about $m = 14,865,801$. Using this value of m, we compute the absolute error for each of the results at the point of termination:

- The error of 160,000 Sobol sample points is 379;
- The error of 700,000 Halton sample points is 69;
- The error of 410,000 random sample points with initial seed 1 is 1,298;
- The error of 430,000 random sample points with initial seed 1147902781 is 2,177;
- The error of 780,000 random sample points with initial seed 1508029952 is 180.

Automatic termination criteria are often used in computational practice. It is of interest to study the relation between the threshold value and the actual error of approximation. See Paskov [25] for the approximation of linear operators in the average case setting assuming that arbitrary linear continuous functionals can be computed. It is proved that standard termination criteria work well. The corresponding problem for approximation of linear operators in the average case setting with information consisting only of function values is still open.

We stress that the conclusions observed in this section for the Monte Carlo and Sobol algorithms hold also for the rest of the tranches. Although, the Halton algorithm performs better than the Monte Carlo algorithm for tranche A and a few other tranches it does not perform well for most of them. Therefore our emphasis for the remainder of this section will be on Sobol rather than Halton points.

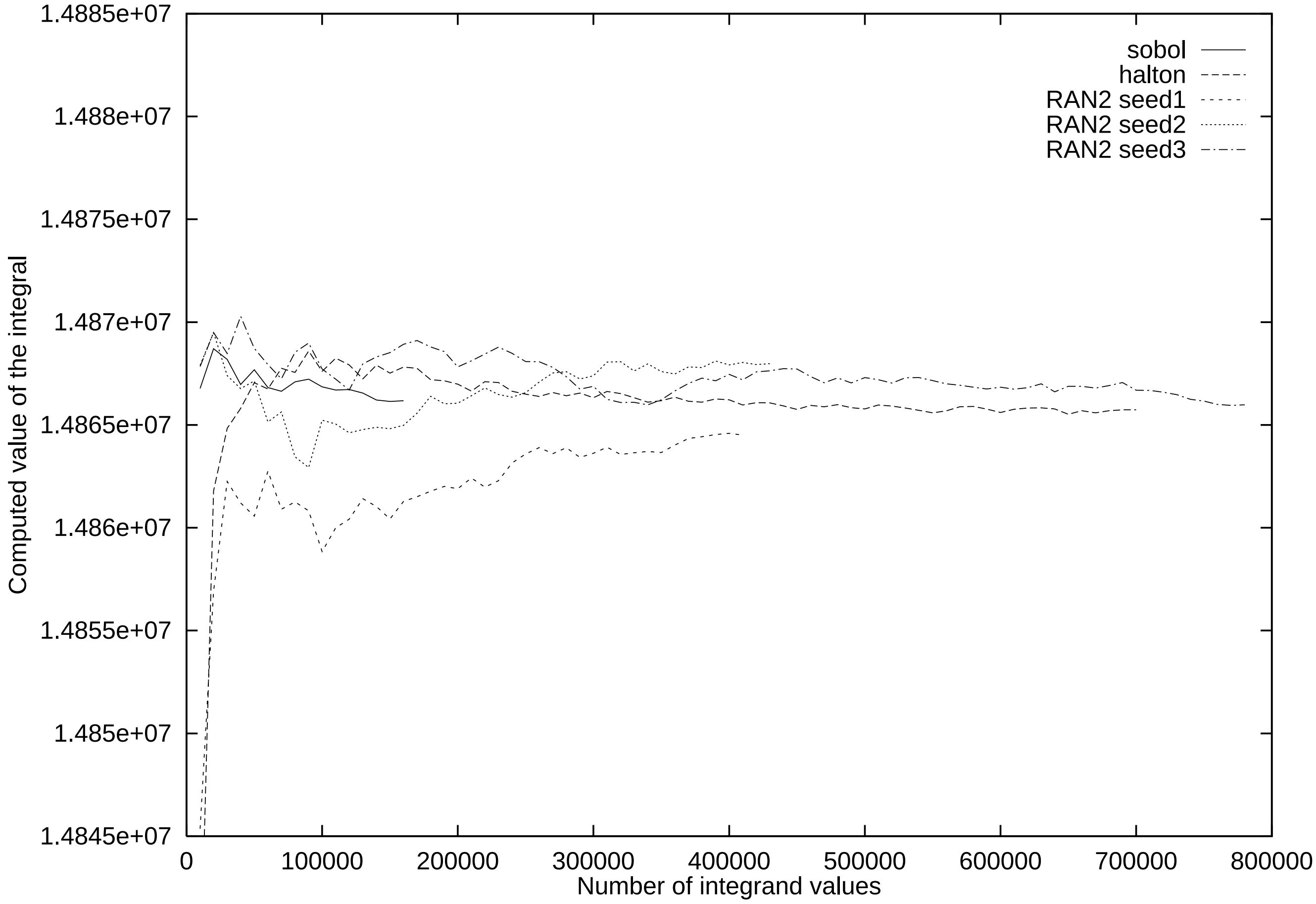

Figure 6: Automatic termination criterion applied to Sobol, Halton, and three Monte Carlo runs using RAN2 for tranche A

7 Antithetic Variables

As already mentioned in Section 2, the expected error of the classical Monte Carlo algorithm depends on the variance of the integrand. Therefore by reducing the variance of the integrand the expected error would also decrease. Various variance reduction techniques such as importance sampling, control variates, antithetic variables and others, see for example [15], are often used with the classical Monte Carlo algorithm.

For the low discrepancy algorithms, the error bound depends on the variation of the integrand, see Section 3. Therefore the error bound will be decreased by reducing of the variation. Reducing the variation of the integrand has not been studied as extensively as reducing the variance. See, however, [4], [17], [20], and [36]. This is a major area for further research.

An important advantage of the classical Monte Carlo algorithm and of the deterministic algorithms studied here, is that they can be utilized very generally. This is important in a number of situations:

- If a financial house has a book with a wide variety of derivatives, it is advantageous to use algorithms which do not need to be tuned to a particular derivative;

- If a new derivative has to be priced, then there is no immediate opportunity to tailor a variance reduction technique to a particular integrand.

Although variance reduction techniques can be very powerful, they can require considerable analysis before being applied. We will therefore limit ourselves here to just one variance reduction technique; antithetic variables. The advantage of antithetic variables is that it can be easily utilized. Tests reveal that it is superior to the classical Monte Carlo algorithm for our CMO problem. We emphasize that antithetic variables is not a palliative; it can be inferior to the classical Monte Carlo algorithm.

For brevity, we refer to the antithetic variables variance reduction technique combined with the Monte Carlo algorithm as the antithetic variables algorithm. This algorithm is based on the identity

$$\int_D f(x)dx = \int_D g(x)dx, \qquad \text{where } g(x) = \frac{1}{2}\left(f(x) + f(1-x)\right) \quad \text{for } f \in L_2(D).$$

Clearly, if the variance $\sigma^2(g)$ of g is much smaller than the variance $\sigma^2(f)$ of f then the Monte Carlo algorithm for g will have much smaller expected error.

We must, of course, remember that the cost of one evaluation of g is equal to the cost of two function evaluations of f. Hence, the antithetic variables algorithm is preferable to the

Tranche	r
A	2.82
B	2.78
C	3.09
D	3.61
E	4.70
G	2.80
H	2.81
J	3.14
R	1.55
Z	2.50

Table 3: The ratios r for the ten tranches of the CMO

Monte Carlo algorithm only if the reduction of the variance is by at least a factor of two. In general, let

$$r = \frac{\sigma(f)}{\sqrt{2}\,\sigma(g)}.$$

Then the expected error of the antithetic variables algorithm is r-times smaller than the expected error of the Monte Carlo algorithm. Furthermore, the cost of both algorithms is about the same. Since r can be smaller than one for some functions, the antithetic variables variance reduction technique, although simple to use, does not work in general and should be used with care, see also [15].

We tested the antithetic variables algorithm for the CMO problem with the ten tranches. Let f_T be the integrand for tranche T. We approximately computed $\sigma(f_T)$, $\sigma(g_T)$, and $r_T = \sigma(f_T)/(\sqrt{2}\,\sigma(g_T))$ for all tranches T. The results for the ratios r_T are given in Table 3. Hence, the antithetic variables algorithm has at least 1.55 and at most 4.70 smaller expected error than the Monte Carlo algorithm for the ten tranches of the CMO. Therefore the antithetic variables algorithm works better than the Monte Carlo algorithm for the CMO problem.

We also tested the antithetic variables technique combined with Sobol points for the CMO problem. Since it did not perform as well as the Sobol algorithm we omit the results.

We now present graphs of the results obtained for the deterministic and antithetic variables algorithms for the CMO problem.

Figure 7 is analogous to Figure 3 in Section 6. It shows the results of Sobol, Halton, and

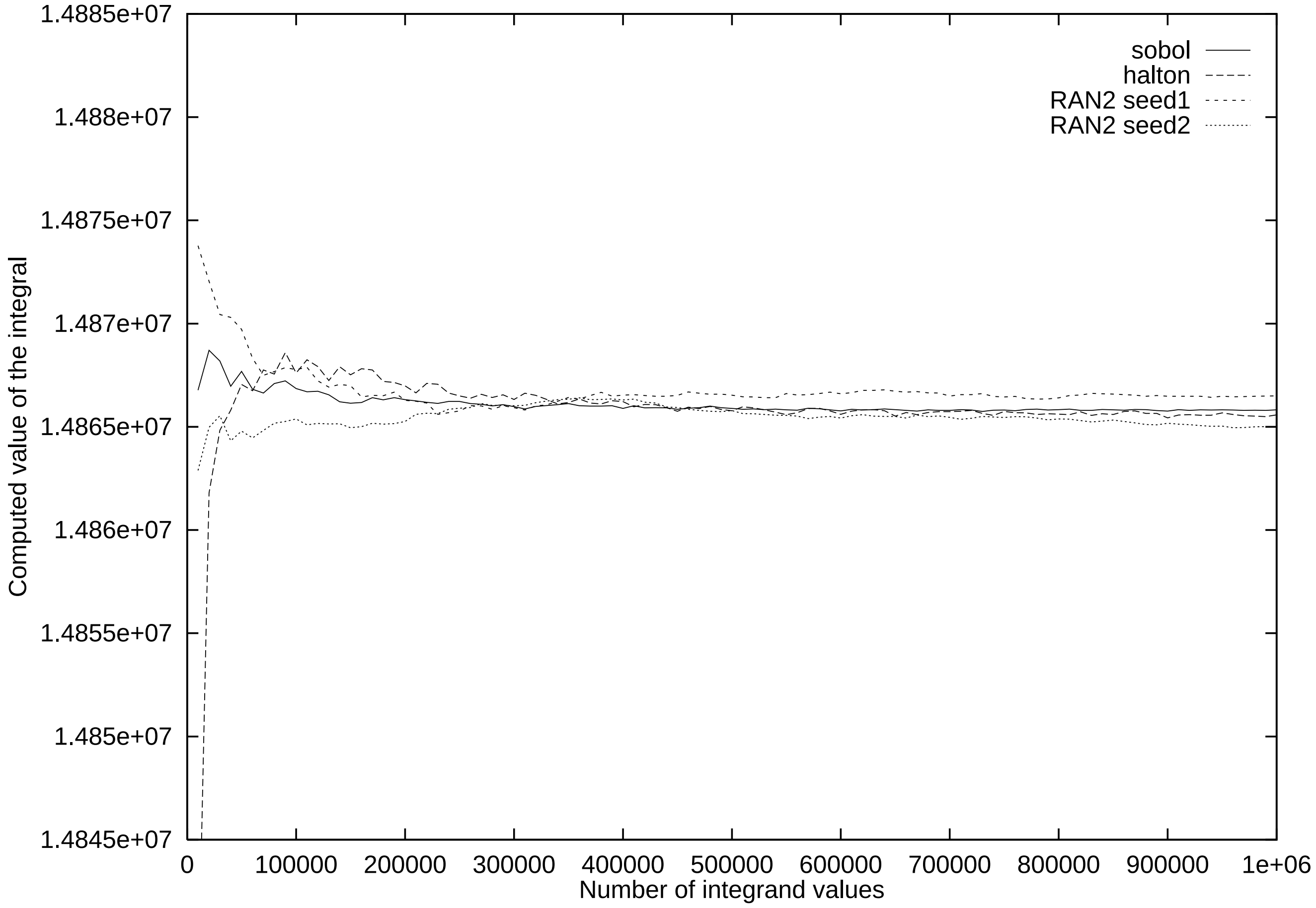

Figure 7: Sobol and Halton runs for tranche A and two antithetic variables runs using RAN2

antithetic variables runs with two randomly chosen initial seeds. Again, Figure 7 exhibits the sensitivity of the antithetic variables algorithm to the initial seed. However, as it should be expected the antithetic variables algorithm works better than the Monte Carlo algorithm. That is why, the spread and the jaggedness of antithetic variables runs in Figure 7 are smaller than the corresponding ones for Monte Carlo runs in Figure 3.

Figure 8 plots the same Sobol and Halton runs versus the arithmetic mean of twenty antithetic variables runs using RAN2 from [28]. We stress that the number of sample points on the x-axis is correct only for the deterministic algorithms. The actual number of sample points for the averaged antithetic variables graph is twenty times the number of sample points on the x-axis. Clearly, the Sobol and the averaged antithetic variables graphs are very close after 450,000 points.

8 Small Number of Sample Points

In this section we compare the performance of the deterministic algorithms with the Monte Carlo and antithetic variables algorithms for a small number of sample points. Results for a small number of points are sometimes of special importance for people who evaluate CMOs and other derivative products. They need algorithms which can evaluate a derivative in a matter of minutes. Rather low accuracy, on the order of 10^{-2} to 10^{-4}, is often sufficient. The integrands are complicated and computationally expensive. Furthermore, many may have to be evaluated on a daily basis with limited computational resources, such as workstations. We must, therefore, limit ourselves to only a small number of sample points.

In this section, by a small number of points we mean 4000 sample points. As we shall see, this still leads to reasonable results and takes less than a couple of minutes of workstation CPU time. We believe that this sample size may yield comparable results for other mortgage-backed securities and interest rate derivatives.

We drop the Halton algorithm from consideration in this section since it is outperformed by both the Monte Carlo and antithetic variables algorithms for 4000 sample points.

From now on, let f be the integrand for tranche A and let

$$g(x) = \frac{1}{2}\left(f(x) + f(1-x)\right), \qquad \text{for } x \in D.$$

Consider the twenty results

$$U_{MC}^{(1)}(f), U_{MC}^{(2)}(f), \ldots, U_{MC}^{(20)}(f) \tag{9}$$

obtained by the Monte Carlo algorithm and the twenty results

$$U_{AV}^{(1)}(g), U_{AV}^{(2)}(g), \ldots, U_{AV}^{(20)}(g) \tag{10}$$

obtained by the antithetic variables algorithm with different initial seeds, each using 4,000 f function evaluations. For 4000 Sobol points, we obtained

$$U_S(f) = 14,868,261. \tag{11}$$

To compute the relative errors of all presented results, we need to know the true or almost true value $I(f)$ of the integral. In Section 7 we showed that the antithetic variables

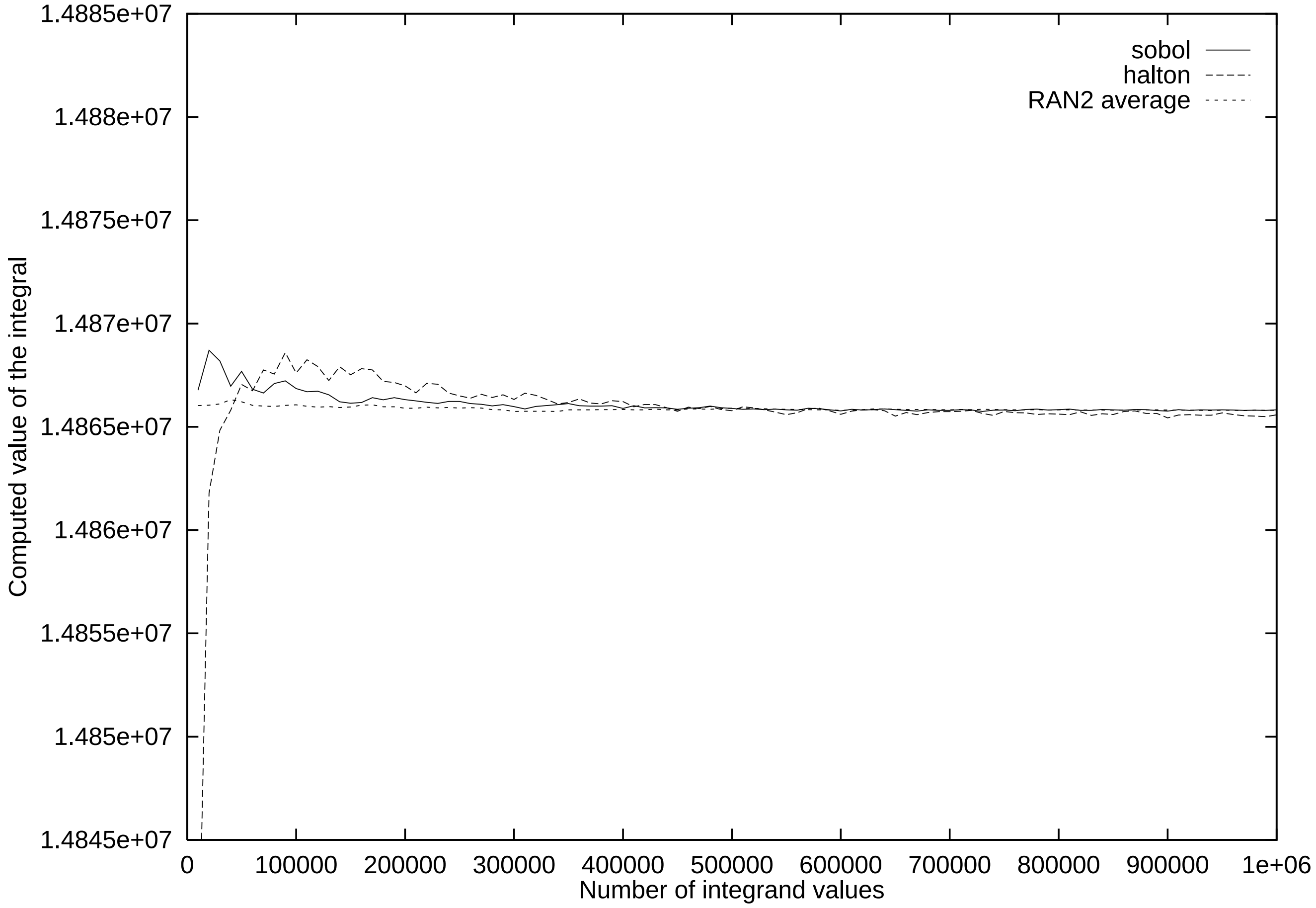

Figure 8: Sobol and Halton runs for tranche A and an average of twenty antithetic variables runs using RAN2

algorithm works better than the Monte Carlo algorithm for the CMO problem. That is why it is reasonable to compute an approximation m to $I(f)$ using 20,000,000 function evaluations with the antithetic variables algorithm. For tranche A, we obtain

$$I(f) \approx m = 14,865,801.$$

We now compute the relative error for each of the results in (9), (10), and (11). In Table 4, we indicate how many times out of 20 the Sobol algorithm has a smaller relative error than the antithetic variables algorithm. As we see, the Sobol algorithm wins for 8 of the tranches, it ties for 1, and it loses for 1. However, for 5 of the tranches the Sobol algorithm wins decisively. In total, the Sobol algorithm wins 139 times and loses 61 times. Therefore, the antithetic variables algorithm sometimes performs better than the Sobol algorithm but in most cases it does not.

In Table 5 we report the smallest and the largest relative errors of the twenty antithetic variables runs in (10). The last column of Table 5 reports the relative errors for the Sobol algorithm.

Note that each of these runs performs very well since the relative error is always at most 0.005. The Sobol algorithm performs even better; it achieves accuracies within range from one part in a thousand to one part in a million, depending on the tranche. We add that it takes approximately 103 seconds to compute the Sobol results and about 113 seconds to compute the antithetic variables results for the ten tranches; each algorithm uses 4000 function evaluations and it is run on a Sun-4/630-M140 workstation.

The results of comparison between the Monte Carlo and the Sobol algorithms are summarized in Table 6 and Table 7. As it can be expected, the performance of the Sobol algorithm versus the Monte Carlo algorithm is even better than the performance of the Sobol algorithm versus the antithetic variables algorithm. In this case, the Sobol algorithm wins decisively for all tranches. In total, the Sobol algorithm wins 177 times and loses 23 times.

9 Statistical Analysis

In this section, we perform statistical analysis of the Monte Carlo and antithetic variables algorithms for a small number of points. We also compare these two randomized algorithms with the Sobol algorithm.

Again, we study tranche A. Consider the one thousand results

$$U_{MC}^{(1)}(f), U_{MC}^{(2)}(f), \ldots, U_{MC}^{(1000)}(f) \tag{12}$$

Tranche	Antithetic variables	Sobol
A	9	11
B	1	19
C	6	14
D	10	10
E	11	9
G	2	18
H	3	17
J	2	18
R	8	12
Z	9	11

Table 4: Number of "wins" of the antithetic variables algorithm and the Sobol algorithm

Tranche	The smallest error	The largest error	Sobol error
A	9.850831e-06	1.078310e-03	1.654482e-04
B	1.684136e-06	8.965597e-04	1.766718e-06
C	2.351182e-06	1.017674e-03	2.154940e-04
D	3.762572e-06	8.231360e-04	2.626482e-04
E	2.111560e-05	5.663271e-04	2.243416e-04
G	1.179616e-06	1.677086e-04	7.305784e-06
H	2.368361e-06	6.108946e-04	3.811441e-05
J	6.517283e-06	7.988188e-04	3.172316e-05
R	3.910856e-05	4.748614e-03	1.179118e-03
Z	1.456121e-05	1.979723e-03	3.784033e-04

Table 5: The smallest and the largest relative error of 20 antithetic variables results and the relative error of the Sobol result each using 4000 sample points

Tranche	Monte Carlo	Sobol
A	3	17
B	0	20
C	3	17
D	3	17
E	2	18
G	0	20
H	0	20
J	0	20
R	8	12
Z	4	16

Table 6: Number of "wins" of the Monte Carlo algorithm and the Sobol algorithm

Tranche	The smallest error	The largest error	Sobol error
A	3.339877e-05	2.633738e-03	1.654482e-04
B	1.037307e-04	3.040330e-03	1.766718e-06
C	3.329284e-05	3.341837e-03	2.154940e-04
D	1.099033e-04	3.536105e-03	2.626482e-04
E	7.673315e-05	3.665118e-03	2.243416e-04
G	3.094756e-05	5.107036e-04	7.305784e-06
H	9.123779e-05	1.454661e-03	3.811441e-05
J	5.152820e-05	1.768061e-03	3.172316e-05
R	3.564157e-05	9.429284e-03	1.179118e-03
Z	5.896356e-05	3.749949e-03	3.784033e-04

Table 7: The smallest and the largest relative error of 20 Monte Carlo results and the relative error of the Sobol result each using 4000 sample points

obtained by the Monte Carlo algorithm and the one thousand results

$$U_{AV}^{(1)}(g), U_{AV}^{(2)}(g), \ldots, U_{AV}^{(1000)}(g) \tag{13}$$

obtained by the antithetic variables algorithm with different initial seeds, each using 4,000 evaluations of f. Let

$$m_{MC} = \frac{1}{1000} \sum_{i=1}^{1000} U_{MC}^{(i)}(f) = 14,864,881,$$

$$m_{AV} = \frac{1}{1000} \sum_{i=1}^{1000} U_{AV}^{(i)}(g) = 14,865,749$$

be the sample means, and let

$$s_{MC} = \frac{1}{999} \sum_{i=1}^{1000} (U^{(i)}(f) - m_{MC})^2 = 216,929,148,$$

$$s_{AV} = \frac{1}{999} \sum_{i=1}^{1000} (U^{(i)}(g) - m_{AV})^2 = 37,331,468$$

be the sample variances of these one thousand Monte Carlo and antithetic variables results, respectively.

According to the Central Limit Theorem, see e.g. [15], the distributions of $U_n(f)$ and $U_n(g)$ are normal for $n \to \infty$ with the same mean $\int_D f(x)dx = \int_D g(x)dx$ and variances $\sigma^2(f)/n$ and $\sigma^2(g)/n$, respectively. To check how far we differ from the normal distribution we plot the density functions of these one thousand results, see Fig. 9 and Fig. 10, by using the statistical package S-Plus. We conclude that these are very good approximations of the normal distribution.

Further statistical analysis require the knowledge of the true value m of the integral. In Section 8, we showed that it is reasonable to assume that $m \approx 14,865,801$. We seek the number of antithetic variables runs, each using 4,000 sample points, which after averaging would give an error δ with probability η. This procedure is widely used in statistical analysis, see [13].

That is, let $\xi_1, \xi_2, \ldots, \xi_k$ be the results for k antithetic variables runs each using 4,000 f function evaluations. As already discussed, we may assume that they are approximately normally distributed with mean m and variance $s = s_{AV}$. Hence we want to find k such that

$$P\left\{ \left| \frac{1}{k} \sum_{i=1}^{k} \xi_i - m \right| \le \delta \right\} \ge 1 - \eta.$$

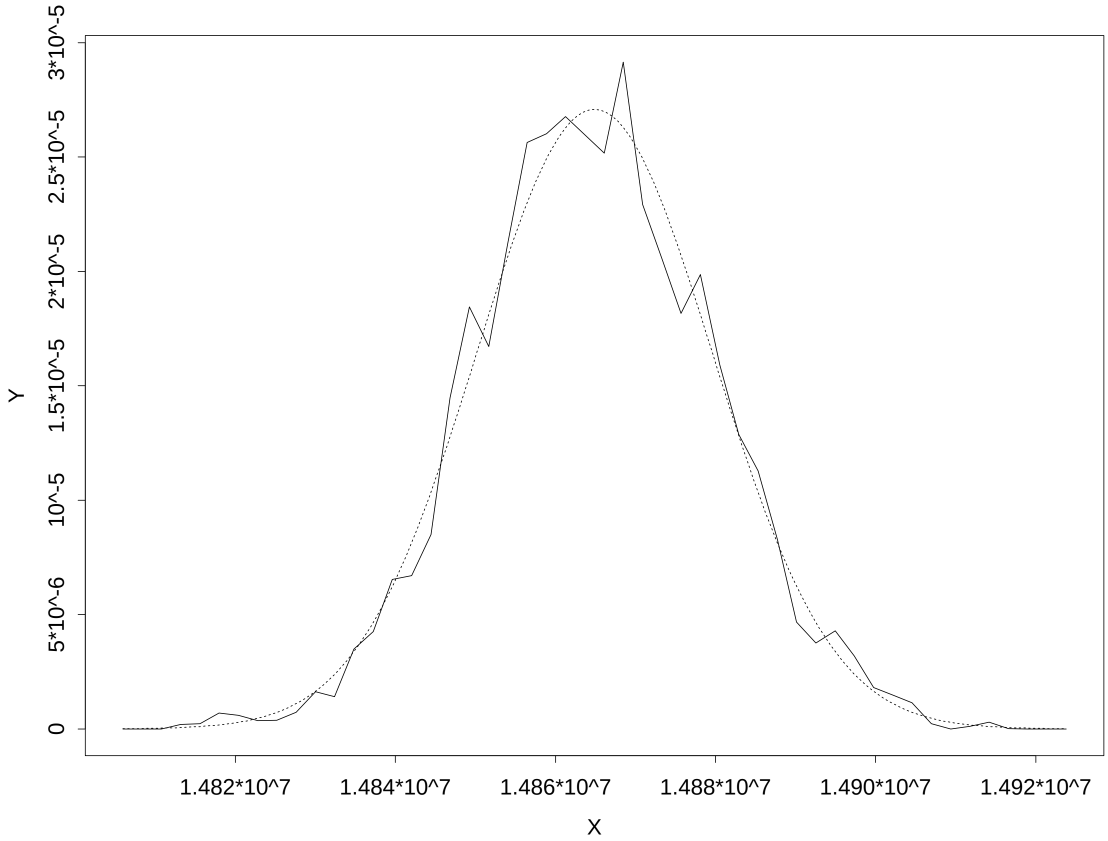

Figure 9: Density function (continuous line) based on one thousand Monte Carlo results each using 4,000 sample points generated by RAN2 and the normal density function (dashed line) with mean m_{MC} and variance s_{MC}

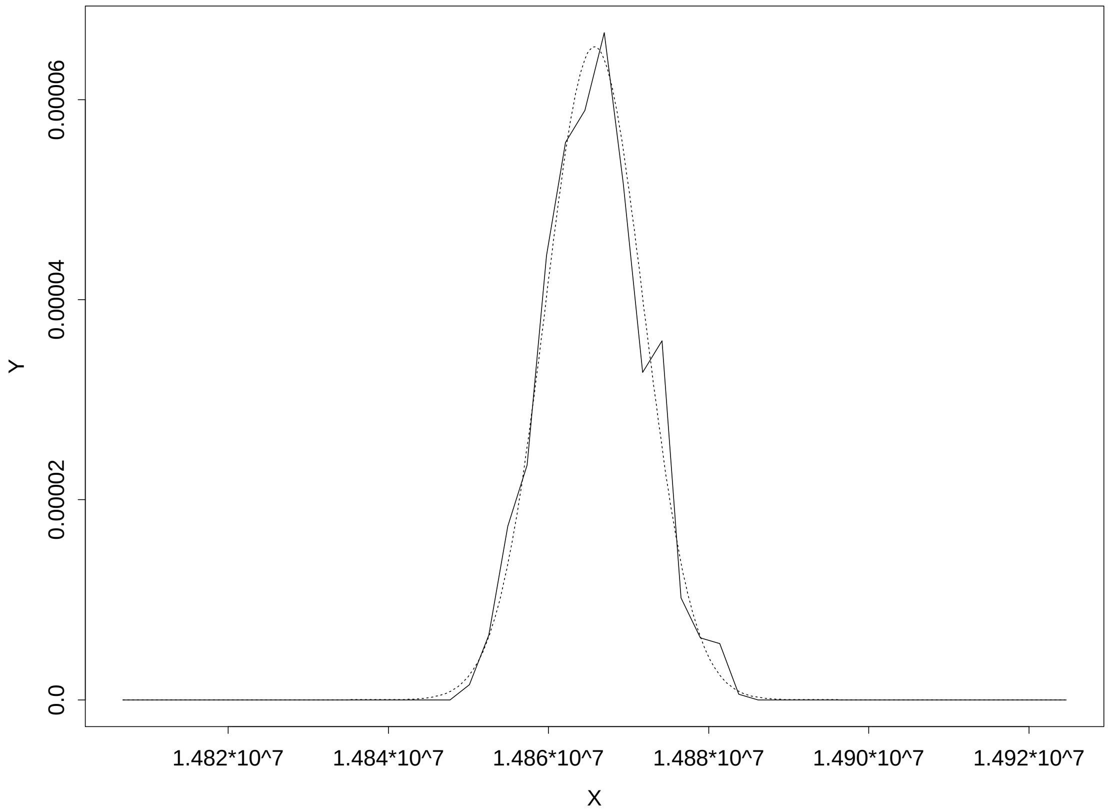

Figure 10: Density function (continuous line) based on one thousand antithetic variables results each using 4,000 sample points generated by RAN2 and the normal density function (dashed line) with mean m_{AV} and variance s_{AV}

Note that the sum $\frac{1}{k}\sum_{i=1}^{k}\xi_i$ is normally distributed with mean m and variance s/k, i.e., standard deviation $\sqrt{s/k}$. To guarantee an error δ with confidence level 68%, i.e., with $\eta = \sqrt{2/\pi}\int_0^1 e^{-t^2/2}dt \approx 0.68269$ which corresponds to one standard deviation from the mean, we need to determine k_1 such that $\sqrt{s/k_1} = \delta$. In this case,

$$k_1 = s/\delta^2 = 37,331,468/\delta^2. \tag{14}$$

To guarantee an error δ with confidence level 95%, i.e., with $\eta = \sqrt{2/\pi}\int_0^2 e^{-t^2/2}dt \approx 0.9545$ which corresponds to two standard deviations from the mean, we need to determine k_2 such that $2\sqrt{s/k_2} = \delta$. Therefore, in this case,

$$k_2 = 4s/\delta^2 = 149,325,872/\delta^2. \tag{15}$$

We now compare the performance of the Sobol algorithm with the antithetic variables algorithm. Since $U_S(f) = 14868261$ is the result for 4,000 Sobol points, the error in this case is

$$|U_S(f) - m| = 2460.$$

Set $\delta = 2460$. Due to (14), to achieve an error at most δ of 4,000 Sobol points with confidence level 68%, we need to obtain approximately 7 antithetic variables runs each using 4,000 sample points and then to average them. Due to (15), to achieve error at most δ with confidence level 95%, we need to obtain approximately 25 antithetic variables runs.

Clearly, the above analysis depends on the assumption about the true answer of the integral. Furthermore, one may argue that by chance the result from Sobol points might happen to be very close to the true answer. That is why we performed similar analysis for the ten Sobol results that use $n = 4,100 - 20i$, $i = 0,1,\ldots,9$, sample points and we set δ as the worst error of the ten Sobol results for tranche A. Then 7 antithetic variables runs obtained before are replaced by 3 runs.

Results for the other tranches are summarized in Table 8. We emphasize that the number of antithetic variables runs are for the worst error of n Sobol points for $n = 4,100 - 20i$, $i = 0,1,\ldots,9$. Hence, for a fixed i, the number of antithetic variables runs could be even higher than the corresponding number in Table 8. From Table 8 we conclude that we need from 7 to 79 antithetic variables runs each using 4,000 sample points to achieve an error with confidence level 95% comparable with the worst error of n Sobol points, $n = 4100 - 20i$, $i = 0,1,\ldots,9$.

For the Monte Carlo algorithm, we proceed analogously. The results are summarized in Table 9. From Table 9 we conclude that we need from 27 to 607 Monte Carlo runs each

Tranche	with confidence 68%	with confidence 95%
A	3	10
B	20	79
C	4	15
D	2	8
E	2	7
G	9	36
H	7	27
J	12	47
R	4	15
Z	2	8

Table 8: Number of antithetic variables runs each using 4,000 random points needed to achieve the worst error of the n Sobol points with $n = 4100 - 20i,\ i = 0, 1, \ldots, 9$

using 4,000 sample points to achieve an error with confidence level 95% comparable with the worst error of n Sobol points, $n = 4100 - 20i,\ i = 0, 1, \ldots, 9$.

Remark 1

Similar statistical analysis may also be performed for a large number of sample points. Due to the large computational cost of these simulations, we compare Sobol results with twenty antithetic variables results (instead of 1000 results as for a small number of sample points), each with different initial seed and using 1,000,000 function evaluations. Results for all tranches are summarized in Table 10. We emphasize that the number of antithetic variables runs are for the worst error of n *Sobol points for* $n = 1,050,000 - 10,000i$, $i = 0, 1, \ldots, 9$.

By comparing Table 8 and Table 10, we conclude that the superiority of the Sobol algorithm over the antithetic variables algorithm for a large number of sample points is greater than for a small number of sample points.

Similar conclusion also holds for the Monte Carlo algorithm. Proceeding analogously, we summarize the results of this comparison in Table 11.

Tranche	with confidence 68%	with confidence 95%
A	14	56
B	153	607
C	44	173
D	35	137
E	53	212
G	64	255
H	38	153
J	67	266
R	13	52
Z	7	27

Table 9: Number of Monte Carlo runs each using 4,000 random points needed to achieve the worst error of the n Sobol points with $n = 4100 - 20i,\ i = 0, 1, \ldots, 9$

Tranche	with confidence 68%	with confidence 95%
A	55	219
B	58	230
C	26	103
D	27	108
E	15	60
G	8	32
H	4	16
J	5	18
R	13	50
Z	5	18

Table 10: Number of antithetic variables runs each using 1,000,000 random points needed to achieve the worst error of the n Sobol points with $n = 1,050,000 - 10,000i,\ i = 0, 1, \ldots, 9$

Tranche	with confidence 68%	with confidence 95%
A	434	1735
B	443	1769
C	246	982
D	350	1399
E	331	1324
G	62	245
H	31	122
J	44	176
R	30	120
Z	28	110

Table 11: Number of Monte Carlo runs each using 1,000,000 random points needed to achieve the worst error of the n Sobol points with $n = 1,050,000 - 10,000i,\ \ i = 0, 1, \ldots, 9$

10 Timing Results and Parallel Speedup for the CMO Problem

Since workstation clusters and networks provide a cost-effective means to perform large-scale computation, we have built and debugged a distributed software system under PVM 3.2 for computing multivariate integrals on a network of workstations. PVM is a software package that allows a network of heterogeneous Unix computers to be used as a single large parallel computer. In this section we report the timing results of the different generators on a single workstation. Then the speedup of the distributed on a network of workstations software system is also measured for the CMO problem.

The CPU time in seconds for simultaneous evaluation of the ten tranches is given in Table 12 for Sobol, Halton, RAN2 from [28], RAN1 from [28], ran1 from [27], and ran2 from [27] generators. The real time in seconds, as should be expected, is slightly higher than the CPU time. It is given in the second column since it is later compared with the real time for the network of workstations to derive the parallel speedup. These results have been obtained on a single Sun-4/630-M140 at the Department of Computer Science, Columbia University. All programs have been compiled with the gcc v2.4 compiler with the highest level O2 of optimization. Clearly, the generator of Sobol points is the fastest one. However, we do not regard this as the most significant reason for the use of Sobol points.

Generator	CPU time	Real time
Sobol	25634	25654
Halton	29814	29881
RAN2	31501	31607
RAN1	28092	28126
ran1	30911	30946
ran2	28320	28332

Table 12: Timing results in seconds for evaluation of the CMO using 1,000,000 sample points on a single workstation

Since the speed of the Sun-4/630-M140 is 4.2 MFLOPS, we note from the above table that it takes approximately 107,500 floating point operations to generate one 360-dimensional Sobol point and to evaluate integrands for this problem. Since the cost of generating one 360-dimensional Sobol point is significantly smaller than the evaluation of the integrands, it takes about 100,000 floating point operations to evaluate only integrands. Clearly, this is a rather rough estimate. We stress that some interest rate and prepayment models used in practice are significantly more complicated and more time-consuming to evaluate than the ones considered in this paper.

We now discuss the parallel speedup for the finance problem achieved with the distributed on a network of workstations software system. The software system for computing multivariate integrals was tested on up to 25 SUN workstations and the timing results were measured. The random number generator RAN2 and the Sobol generator were used. The results are given in Table 13. We now compare the real time entry of RAN2 in Table 12 with entries in Table 13. Let N be the number of workstations. In our case N is 5,10,15,20,25. The speedup for the first three entries is about $0.99N$. Then it degrades slightly to $0.92N$ and $0.9N$ for the fourth and the fifth entries, respectively. We stress that this degradation of the performance is partially due to the fact that the last several machines added for the test, were slightly slower than the Sun-4/630-M140, which was used to measure the time on a single workstation.

Similar speedups are measured for the Sobol generator. The speedups for the first three entries are $0.96N, 0.98N, 0.98N$, respectively. The speedup for the last two entries is $0.91N$.

Therefore, networks of workstations are a cost-effective way to perform these computations. The measured speedup for the parallel/distributed software system for both the Monte Carlo and deterministic algorithms is at least $0.9N$ for $N \leq 25$.

Number of machines	Real time for RAN2	Real time for Sobol
5	6382	5360
10	3170	2625
15	2113	1750
20	1705	1408
25	1398	1125

Table 13: Timing results in seconds for evaluation of the CMO using 1,000,000 sample points generated by RAN2

11 Discussion and Future Directions

In this section we present some thoughts about why the Sobol points are so successful. We end with some directions for future research.

We give two reasons which we believe explain, at least in part, the success of Sobol points in solving the CMO problem.

First, let us assume that there are no prepayments. This implies that the cash flow from the underlying pool of mortgages for every month is constant. Due to the time value of the money this means that the present value of the cash flow is a decreasing function of the number of the month. In other words, the present value of the cash flow for earlier months is greater than the present value of the cash flow for later months. The presence of prepayments and rules of distribution to different tranches will distort this monotonicity property. However, this property will be reflected to some extent in the cash flows of the ten tranches. It is a well-known property of the Sobol points[2], see [33], that the low dimensional components are more uniformly distributed than the high dimensional components. As pointed out by Sobol in [35], the Sobol points will be more efficient for numerical integration of functions for which the influence of the x_j-th variable decreases as j increases. Since this property also holds for many other financial derivatives we expect that the Sobol points will provide a powerful alternative to the Monte Carlo and antithetic variables algorithms.

Second, appropriate choices for the initial direction numbers increase the uniformity of the Sobol points in the 360-dimensional unit cube. We believe that our choices of the initial direction numbers contributes to the successful performance of the Sobol points.

The Halton algorithm did not perform as well as the Sobol algorithm. We believe that

[2] Other low discrepancy sequences also satisfy this property, see [21]

this behavior is due to the fact that the Halton points are less uniformly distributed than the Sobol points; especially in high dimensions and sample sizes of 1,000,000 or less, see also [18].

We end this section by suggesting some directions for future work:

- Compare the performance of low discrepancy and Monte Carlo algorithms on other financial derivatives;
- Test the performance of other known low discrepancy sequences on various derivatives;
- As mentioned in Section 8, results for a small number of samples are often of special interest in finance. It would be attractive to design new deterministic sequences which are very uniformly distributed for a small number of points;
- Characterize analytic properties of classes of financial derivatives and design new algorithms tuned to these classes;
- Study error reduction techniques for deterministic algorithms;
- There are numerous open theoretical problems concerning high dimensional integration and low discrepancy sequences; see [42] for some of them. We believe that their solution will aid in the design of better algorithms for finance problems.

Acknowledgments

An earlier report on this work was presented by Prof. J. Traub at the Bank of England Conference at the Isaac Newton Institute, Cambridge, U.K. in May 1995.

I express my gratitude to I. Vanderhoof for his invaluable help, numerous discussions and suggestions, and introducing me to Goldman Sachs.

I also express my gratitude to J. Traub and H. Woźniakowski for numerous discussions, suggestions, and guidance on research and on the preparation of this paper.

I would like to thank A. Belur for his help, advice, and cooperation in providing the finance problem.

I am also grateful to T. Boult and S. Baker for their numerous discussions and suggestions during the preparation of this paper.

I appreciate discussions with N. Marinovich, J. Tilley, J. Langsam, P. Karasinski, D. Schutzer, B. Bojanov, S. Tezuka, A. Werschulz, I. M. Sobol, B. Shukhman, V. Temlyakov, S. Heinrich, E. Novak, and T. Warnock.

I would like sincerely to thank the Mortgage Research Group at Goldman Sachs and people associated with it and particularly P. Niculescu, R. Wertz, and J. Davis who gave me a chance to greatly improve my knowledge in mortgage-backed securities and interest rate derivatives.

I also appreciate the help of P. Cheah in building the distributed platform used in the software system for computing integrals.

References

[1] Adler, S. J., The Geometry of Random Fields, *Wiley Series in Prob. and Math. Stat.*, 1981.

[2] Antonov, I.A., and Saleev, V.M., An Economic Method of Computing LP_τ-sequences, *USSR Computational Mathematics and Mathematical Physics*, 19, 252-256, 1979.

[3] Bratley, P. and Fox, B.L., Algorithm 659, Implementing Sobol's Quasirandom Sequence Generator, *ACM Trans. Math. Software*, 14, 88-100, 1988.

[4] Chelson, P., Quasi-random Techniques for Monte Carlo Methods, *PhD Dissertation, The Claremont Graduate School*, 1976.

[5] Fabozzi, F. J., Handbook of Mortgage Backed Securities, Probus Publishing Co., 1992.

[6] Fabozzi, F. J., Fixed Income Mathematics Probus Publishing Co., 1988.

[7] Fox, B.L., Algorithm 647, Implementation and Relative Efficiency of Quasirandom Sequence Generators, *ACM Trans. Math. Software*, 12, 362-376, 1986.

[8] Geweke, J., Monte Carlo Simulations and Numerical Integration, *to appear in Handbook of Computational Economics edited by H. Amman, D. Kendrick, and J. Rust*, Elsevier, Amsterdam, 1996.

[9] Halton, J.H., On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals, *Numer. Math.*, 2, 84-90, 1960.

[10] Halton, J.H. and Smith, G.B., Algorithm 247, Radical-inverse quasi-random point sequence, *Commun. ACM*, 7, 701-702, 1964.

[11] Halton, J.H., A retrospective and prospective survey of the Monte Carlo method *SIAM Review*, 12, 1970, 1-63.

[12] Hammersley, J. and Handscomb, D., Monte Carlo methods, *Methuen, London*, 1964.

[13] Hoel, P. G., Elementary Statistics, *John Wiley and Sons*, 1976.

[14] Janse van Rensburg, E. J. and Torrie, G. M., Estimation of multidimensional integrals: Is Monte Carlo the best method? *J. Phys. A.: Math. Gen.*, 26, 1993, 943-953.

[15] Kalos, M. H. and Whitlock, P. A., Monte Carlo Methods, Volume I, *John Wiley and Sons*, 1986.

[16] Knuth, D.E., Seminumerical Algorithms, vol. 2 of The Art of Computer Programming, *Addison Wesley*, 1981.

[17] Maize, E., Contributions to the theory of error reduction in quasi-Monte Carlo methods, *PhD Dissertation, The Claremont Graduate School*, 1981.

[18] Morokoff, W. J. and Caflisch, R. E., Quasi-Random Sequences and their Discrepancies, *SIAM J. Scientific Computing*, 15, 1251-1279, 1994.

[19] Morokoff, W. J. and Caflisch, R. E., Quasi-Monte Carlo Integration, *J. Comp. Physics*, 122, 218-230, 1995.

[20] Moskowitz, B. and Caflisch, R. E., Smoothness and Dimension Reduction in Quasi-Monte Carlo Methods, *to appear in Math. Comp. Modeling.*

[21] Niederreiter, H., Random Number Generation and Quasi-Monte Carlo Methods, *CBMS-NSF,63, SIAM*, Philadelphia, 1992.

[22] Novak, E., Deterministic and Stochastic Error Bounds in Numerical Analysis, Lecture Notes in Math., *Springer Verlag, Berlin*, 1988.

[23] Paskov, S. H., Average Case Complexity of Multivariate Integration for Smooth Functions, *J. Complexity*, 9, 291-312, 1993.

[24] Paskov, S. H., Computing High Dimensional Integrals with Applications to Finance, *Joint Summer Research Conference on Continuous Algorithms and Complexity,* Mount Holyoke College, June, 1994.

[25] Paskov, S. H., Termination Criteria for Linear Problems, *J. Complexity*, 11, 105-137, 1995.

[26] Paskov, S. H. and Traub, J. F., Faster Valuation of Financial Derivatives, *The Journal of Portfolio Management*, Vol. 22, No. 1, 113-120, Fall 1995.

[27] Press, W., Teukolsky S., Vetterling, W., and B. Flannery, Numerical Recipes in C, First Edition, *Cambridge University Press*, 1988.

[28] Press, W., Teukolsky S., Vetterling, W., and B. Flannery, Numerical Recipes in C, Second Edition, *Cambridge University Press*, 1992.

[29] Roth, K. F., On irregularities of distribution, *Mathematika*, 1, 73-79, 1954.

[30] Roth, K. F., On irregularities of distribution, IV, *Acta Arith.*, 37, 67-75, 1980.

[31] Rust, J., Using randomization to break the curse of dimensionality, *Social Systems Research Institute Working Paper Series*, No. 9429, 1994.

[32] Sarkar, P.K. and Prasad, M. A., A comparative study of pseudo and quasi random sequences for the solution of integral equations, *J. Computational Physics*, 68, 66-88, 1978.

[33] Sobol, I.M., On the distribution of points in a cube and the approximate evaluation of integrals, *USSR Computational Mathematics and Mathematical Physics*, 7, 86-112, 1967.

[34] Sobol, I.M., Numerical Monte Carlo methods (in Russian), *Izdat "Nauka", Moscow*, 1973.

[35] Sobol, I.M., Quadrature formulas for functions of several variables satisfying general Lipschiz condition, *USSR Computational Mathematics and Mathematical Physics*, 29, 935-941, 1989.

[36] Spanier, J. and Maize, E., Quasi-Random Methods for Estimating Integrals using Relatively Small Samples, *SIAM Review*, 36, 19-44, 1994.

[37] Tezuka, Shu, A Generalization of Faure Sequences and its Efficient Implementation, *Technical Report, IBM Research, Tokyo*, 1994.

[38] Traub, J. F., Average Case Computational Complexity of High-Dimensional Integration with Applications to Finance, *NSF Symposium on Simulation and Estimation,* Department of Economics, University of California, Berkeley, August, 1994.

[39] Traub, J. F., Solving Hard Problems with Applications to Finance, *Thirteenth World Computer Congress, IFIP* 94, Hamburg, August, 1994.

[40] Traub, J. F., Wasilkowski, G. W., and Woźniakowski, H., Information-based Complexity, *Academic Press*, New York, 1988.

[41] Traub, J. F. and Woźniakowski, H., The Monte Carlo Algorithm with a Pseudorandom Generator, *Mathematics of Computation*, 58, 323-339, 1992.

[42] Wasilkowski, G. W. and Woźniakowski, H., Explicit Cost Bounds of Algorithms for Multivariate Tensor Product Problems, *J. Complexity*, 11, 1-56, 1995.

[43] Woźniakowski, H., Average Case Complexity of Multivariate Integration, *Bull. AMS (New Series)*, 24, 185-194, 1991.

[44] Woźniakowski, H., Tractability and Strong Tractability of Linear Multivariate Problems, *J. Complexity*, 10, 96-128, 1994.