Valuation of Mortgage–Backed Securities in a Distributed Environment

by

Vladimir Surkov

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Valuation of Mortgage–Backed Securities in a Distributed Environment

Vladimir Surkov
Master of Science
Graduate Department of Computer Science
University of Toronto
2004

Valuation of Mortgage–Backed Securities, regarded as integration in high–dimensional space, can be readily performed using the Monte Carlo method. The Quasi–Monte Carlo method, by utilizing low–discrepancy sequences, has been able to achieve better convergence rates at computational finance problems despite analysis suggesting that the improved convergence comes into effect only at sample sizes growing exponentially with dimension. This may be attributed to the fact that the integrands are of low effective dimension and quasi–random sequences' good equidistribution properties in low dimensions allow for the faster convergence rates to be attained at feasible sample sizes. The Brownian bridge discretization is traditionally used to reduce the effective dimension although an alternate choice of discretization can produce superior results. This paper examines the standard Brownian bridge representation and offers a reparametrization to further reduce dimension. The performance is compared both in terms of improvement in convergence and reduced effective dimensionality as computed using ANOVA decomposition. Also, porting of the valuation algorithm to a distributed environment using Microsoft .NET is presented.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Monte Carlo method

The Monte Carlo (MC) method has proved to be a flexible and robust computational tool in many areas of physical sciences, and, more recently, in finance. The MC method becomes especially attractive in valuation of complex, path–dependent derivative securities (securities whose value depends on one or more underlying securities) such as Mortgage–Backed Securities (MBS). The value of a MBS is the sum of discounted cash flows generated by the security over its lifetime, typically 30 years (360 months). However, the cash flows are not deterministic and depend on the interest rates and on the prepayment assumptions of the model, making the valuation of an MBS a complex task that can be conveniently performed using MC simulation.

To value a MBS using MC algorithm we must generate a set of N independetly drawn interest rate paths, aggregate the discounted cash flows of the MBS under each sample path to obtain the value of the MBS, and average the aggregated cash flows over the sample paths. The law of large numbers guarantees consistency of the estimate of the MBS value as the mean of samples. One of the main drawbacks of the straightforward MC method is its slow convergence rate of $O(N^{-1/2})$. Applying variance reduction techniques, such as antithetic sampling, reduces the constant term yet the method remains $O(N^{-1/2})$. The slow convergence of the MC method is due to the clumping of points in the pseudo random sequence used to generate sample paths. Since the points are drawn independently, there is a chance that the points drawn may land close to each other. The advantage of MC is that the order of convergence is independent of the problem dimension and the Central Limit Theorem allows us to perform precise error analysis.

1.2 Quasi–Monte Carlo methods

Quasi–Monte Carlo (QMC) methods have been extensively used to improve upon slow convergence of the standard MC method. QMC methods, by using deterministic sequences that have greater uniformity than standard MC sequences, have error bounds of $O(log(N)^k/N)$ where k depends on the dimension of the problem. Although this suggests superior convergence, the advantage might not materialize at

feasible sample sizes. Although it is very difficult to give a precise bound for large d, it was shown that the QMC methods do not achieve their asymptotic level until N is very large (grows exponentially with d) and impractical for any use. Despite the fact that the error analysis and some numerical examples would present QMC as ineffective for high dimensional problems, QMC methods have shown good accuracy at practical sample sizes. This leads us to believe that the integrands at hand are of lower *effective dimension*.

1.3 Brownian Bridge and Effective Dimension Reduction

There have been several approaches to reducing the effective dimension of a problem discussed in the literature. One of the most widely used approaches in conjunction with quasi–random sequences is the Brownian bridge discretization introduced by Caflisch and Moskowitz (1995). The idea is to contain most of the variance of the random walk with the first few steps of the discretization. Even though the total variance remains the same, this approach benefits from low discrepancy sequences' good equidistribution properties in low dimensional projections by packing most of the variance into the first few dimensions. However, even further dimensional reduction can be achieved by choosing an alternate ordering of the Brownian bridge, thus leading to even faster convergence.

1.4 ANOVA and Dimension Distribution

The analysis of variance (ANOVA) is an important tool in quantifying the importance of variables and their interaction. By working with a measure of variable importance we can efficiently determine the degree to which a particular Brownian bridge representation captures the variance of the function by a given set of variables. The ANOVA on $[0,1]^d$ involves looking at all 2^d-1 possible variable combinations and determining their contribution to the total variance. Although for $d = 360$ this becomes a challenging task, some assumptions can be made to simplify the task. Many problems in financial computation can be shown to be sums of one dimensional integrands that allow us to look only at d one dimensional variable sets. Furthermore, for the particular 360 dimensional problem considered here, 99% of the variance is contained in the first 25 dimensions and 95% of the variance in only 9 dimensions. Thus, by employing reasonable constraints on the problem, we can limit ourselves to looking only at a limited set of variables to speed up the computation.

1.5 Parallel Processing in a Distributed Environment

Performing large–scale computations, such as valuing MBS using or computing ANOVA decomposition for various Brownian bridge representations, can be efficiently done on a networked cluster of workstations. Traditionally most scientific distributed applications were written in C and used message passing libraries such as MPI and PVM to conduct communication between nodes. New technologies such as CORBA and Microsoft .NET are regarded as bulky and unfit for high performance computation. In this paper we'll implement a distributed software system written in C# that uses Microsoft .NET Remoting

architecture to facilitate communication between computers. On the computing end, a MATLAB server will be launched to perform the actual computation.

1.6 Objectives

The purpose of this paper is two fold. First, we try to achieve a better reduction of the effective dimension of the problem by modifying the traditional Brownian bridge discretization approach of generating midpoints of successive intervals; i.e., $y_M, y_{M/2}, y_{M/4}, y_{3M/4}, y_{M/8}, \ldots$ It seems more natural that for problems involving valuation of derivative securities to place the "good points" of a low–discrepancy sequence not in the traditional approach, uniformly filling the space, but rather shift them toward the start of the time series to account for the discount factor and mortgage expiration. This natural assumption is backed by the estimate of mean length of mortgages used in numerical examples being around 100 months. By shifting the discretization points we can generate good path samples when most of the mortgages have not expired yet.

The second goal of this paper is to implement a distributed application that will allow various computational tasks to be performed in a distributed environment. We'll explore the Microsoft .NET architecture and how it compares to the traditional interprocess communication methods. Implementational issues of communicating between MATLAB and C# will also be discussed.

Chapter 2

Mortgage–Backed Securities

2.1 Introduction

A Mortgage–Backed Security (MBS) represents a claim on the cashflows generated by a group of one or more mortgages. Since their inception in the 1970s, Mortgage–Backed Securities have experienced tremendous growth and become very popular as an investment tool among individuals and financial institutions. This is attributed to the benefits offered to investors by MBSs, including attractive yields, minimal credit risk, and a liquid market. Also, one cannot underestimate the size of the Mortgage–Backed Securities market. As of 2002, the total value of outstanding MBS was exceeding $3.8 trillion and constituted 21% of the total outstanding bond debt in the U.S. Fixed Income market Kelman (2002).

The cornerstone of the valuation process of securities dependent on mortgage cashflows is modeling borrowers' refinancing strategy. Early models by Brennan and Schwartz (1985) thought of the borrower as an optimal agent actively seeking to minimize the present value of his mortgage. This approach worked poorly due to its inability to regard borrowers as suboptimal agents and thus match the actual prepayment rates. This led to behavioural mortgage valuation models that produced reduced form prepayment behaviour models.

2.2 Details

A Mortgage–Backed Security is created when a mortgage issuer pools together a set of mortgages. Typically, MBSs are issued by one of three government–sponsored agencies: Gennie Mae (GNMA), Freddie Mac (FHLMC), and Fannie Mae (FNMA), although private mortgage issuers can also issue MBSs. The mortgages in a pool should have common characteristics, such as interest rate, payment terms, etc., and meet specific government guidelines. Once the pool has been set up, the issuer sells units to investors directly or through securites markets. Ownership of a MBS entitles one to the mortgage borrower's interest and principal payments minus the issuer's fees for guaranteeing the timely payments to the investor, even in case of borrower's default. The payments on the MBS are guaranteed by the respective issuing agency. GNMA MBSs are virtually riskless since they are backed by the full faith

and credit of the United States. FHLMC and FNMA MBSs do not have such guarantees, but still bear credit risk comparable to AAA and are traded at par with GNMA MBSs.

Thus, the owner of the MBS is subjected to two types of risk: the interest and prepayment risks. Fluctuating interest rates will vary the interest paid by the borrower and fluctuating prepayment rates will vary the principal on which the interest is paid. All of these factors make valuation of MBSs a complex process which involves modeling the term structure (interest rates) and the prepayment behavior of the individuals, which in turn depends on term structure, and strategical and random factors.

2.3 Term Structure and Prepayment Models

As previously mentioned, MBS value is particularly sensitive to the interest rate movements. This makes the term structure model, on which basis the rates are generated, very important in MBS valuation. The simplest (although not very realistic) way to model the interest rates is using the Rendelman and Barter reference model where the interest rate r follows the geometric Brownian motion model:

$$dr = \mu(t, r)rdt + \sigma(t, r)rdW \tag{2.1}$$

where μ is the drift term, σ is the volatility and W is a Weiner process.

This model assumes that the short–term interest rate behaves like a stock price. The major deficiency of such an assumption is that, unlike stocks, interest rates exhibit a *mean reversion* property, drawing toward a long–run average. Although the Rendelman and Barter model does not exhibit the mean reversion property, we need to set drift term $\mu = 0$ to eliminate any price drift. A more sophisticated model, like the Vasicek model, would include the mean reversion property and the short rate would be pulled toward the mean level more strongly the further the rate drifts away from the mean.

The second crucial factor in valuing a MBS is the prepayment model. Its objective is to describe the behavior of individuals when it comes to paying off their mortgages, and it is not an easy task. One of the major difficulties in producing such a model is that it must accurately describe rational behavior in response to economic factors such as refinancing due to falling interest rates as well as behaviour attributed to exogenous reasons, such as holding on to a premium mortgage (which can be explained by borrower's poor credit history).

There are many prepayment models in the literature, but most agree that the prepayment model must include the following four factors:

Refinancing incentives Borrower's incentive to refinance and is measured as homeowner's weighted average coupon rate (WAC) divided by the pool refinancing rate.

Age of mortgage (seasoning) The tendency of mortgage payments to increase over time. Homeowners aren't very likely to move again within a short span of purchasing a house.

Seasonality Yearly trends in housing turnover. Homeowner's are more likely to move during the summer than winter.

Premium burnout A spike in refinancing due to incentives is followed by a burnout. The remaining mortgage holders are typically less sensitive to further refinancing incentives.

2.4 Model Details

The model and the notation follow Paskov (1997), Caflisch, Morokoff, and Owen (1997). Our model assumes the cash flow consists of interest and principal repayment, and thus depends on the future interest and prepayment rates. Let $M = 360$ be the length of the underlying mortgages in our MBS. Let $\{\xi_k\}_{k=1}^{M}$ be normally distributed random variables with mean 0 and variance σ^2

The interest rate at month k, i_k, is given by:

$$i_k = K_0 e^{\xi_k} i_{k-1} = K_0^k i_0 e^{(\xi_1 + \cdots + \xi_k)} \tag{2.2}$$

where i_0 is the initial interest rate and the constant $K_0 = e^{-\sigma^2/2}$ is chosen to normalize the log-normal distribution; i.e., so that $E(i_k) = i_0$. Although this model is not exponential as presented in 2.1, it still exhibits the mean reversion property.

The prepayment model, a function of the interest rates, is:

$$\begin{aligned} w_k &= K_1 + K_2 \arctan(K_3 i_k + K_4) \\ &= K_1 + K_2 \arctan(K_3 K_0^k i_0 e^{(\xi_1 + \ldots + \xi_k)} + K_4) \end{aligned} \tag{2.3}$$

where K_1, K_2, K_3, K_4 are constants. w_k represents the fraction of outstanding mortgages at month k that are prepaying in that month. Using the above model for the prepayment rate, we can generate various sample problems by modifying the constants K_i, thus modifying the prepayment behavior in response to varying interest rates. Essentially, K_1 and K_2 are the mean and volatility of the prepayment rate respectively, while K_3 and K_4 jointly control the linearity of arctan function on relevant domain. Small magnitudes of K_3 and K_4 make arctan linear and non linear for large values, with the sign of K_3 controlling the sign of the slope in the linear case.

The total cash flow from r_k remaining mortgages for month k consists of two parts: $r_k w_k$ of the mortgages are prepaying the entire mortgage, thus paying $Ca_{360-k+1}$ ($a_{360-k+1}$ is the remaining mortgage annuity), while the remaining $r_k(1 - w_k)$ are paying only the monthly payment C. Thus the cash flow for month k, m_k, can be expressed as:

$$r_k = \prod_{j=1}^{k-1} (1 - w_j) \tag{2.4}$$

$$\begin{aligned} m_k &= r_k(C(1 - w_k) + C w_k a_{360-k+1}) \\ &= C(1 - w_1) \cdots (1 - w_{k-1})((1 - w_k) + w_k a_{360-k+1}) \end{aligned} \tag{2.5}$$

where the remaining annuity after month k is given by $a_{360-k+1} = \sum_{j=0}^{M-k} (1 + i_0)^{-j}$

The present value of a Mortgage–Backed Security is the discounted value of all underlying cash flows in the security

$$PV = E(\sum_{k=1}^{M} u_k m_k) \tag{2.6}$$

where E is the expectation over the random variables involved and the discounting factor is $u_k = \prod_{j=0}^{k-1} \frac{1}{1+i_j}$

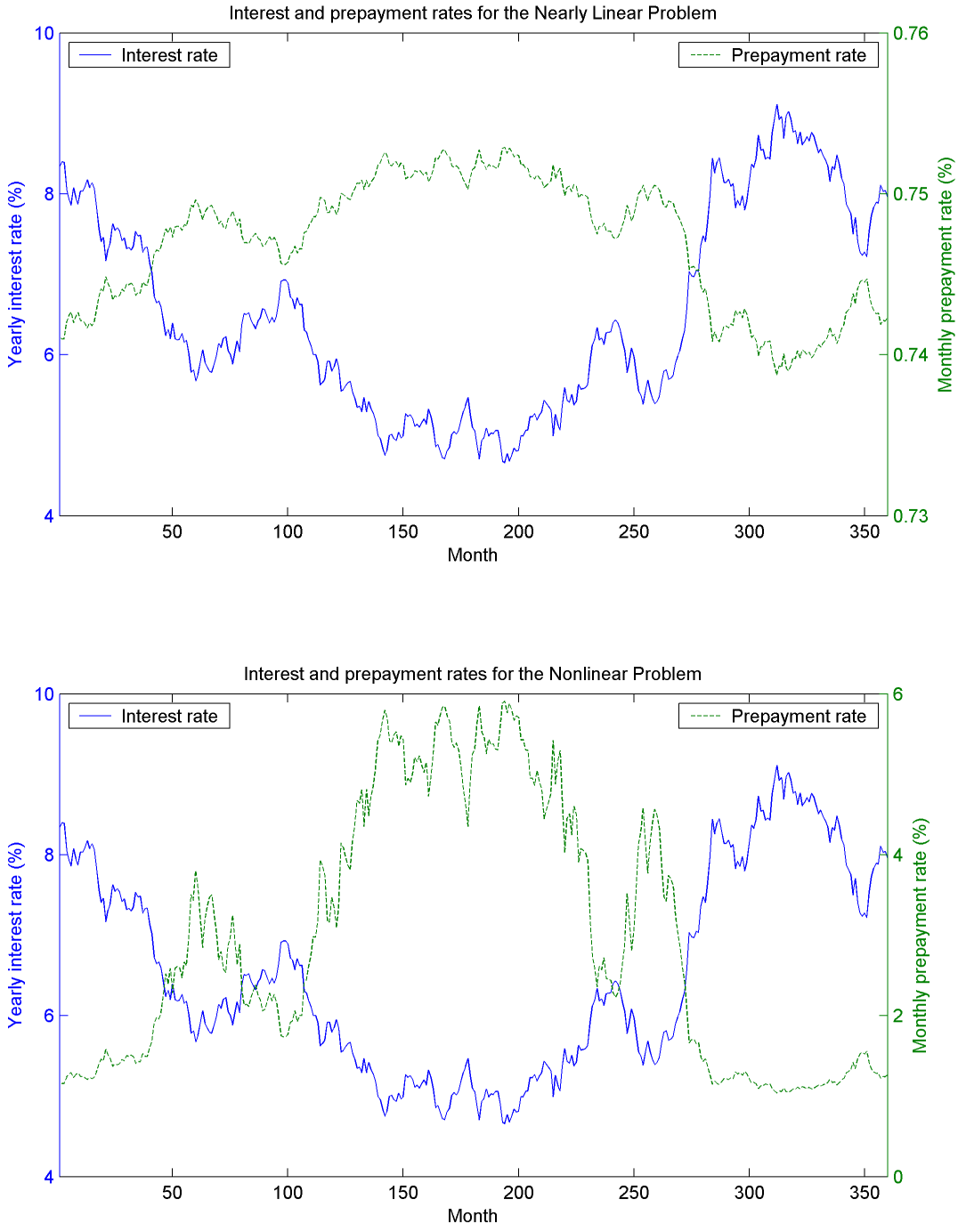

Figure 2.1: Sample Interest and Prepayment rates

Numerical examples in this paper use two problems defined by parameters provided by Caflisch, Morokoff, and Owen (1997). The first problem, referred to as Nearly Linear is defined by:

$$(i_0, K_1, K_2, K_3, K_4, \sigma^2) = (0.007, 0.01, -0.005, 10, 0.5, 0.0004) \tag{2.7}$$

The second problem, referred to as Nonlinear, is defined by:

$$(i_0, K_1, K_2, K_3, K_4, \sigma^2) = (0.007, 0.04, 0.0222, -1500, 7.0, 0.0004) \tag{2.8}$$

We are interested in considering these two problems since they represent two different classes of problems. The Nearly Linear example appears to be a nearly linear function of ξ_k and most of the function variation (99.96%) comes from one dimensional structure. The Nonlinear problem is not as one dimensional and the prepayment function is far more sensitive to extremely low interest rates, as can be seen in Figure 2.1.

Chapter 3

Monte Carlo methods

Pricing a derivative security such as a MBS, which involves monthly cashflows over 30 years (360 months), can be viewed as an integration in 360 dimensional space. For problems of such high dimensionality d, MC and QMC methods can be very attractive, especially in the case of MBS, where the prepayment model can not be easily expressed as a 360 dimensional integrand.

One of the main advantages in using quasi–random sequences over pseudo–random sequences in path construction is their improved uniformity. Quasi–random sequences are designed to fill out the space much more uniformly and are thus called *low– discrepancy* sequences. However, the construction of such sequences in high dimensions is extremely difficult and the improved convergence properties of the quasi-random sequences do not come into effect until the number of iterations is very large and impractical for all intents and purposes.

The following will give a brief introduction to the Monte Carlo methods used in this paper. For an extensive overview of MC methods see Owen (1998). For a detailed survey of low– discrepancy sequences, see Caflisch, Morokoff, and Owen (1997), Paskov (1997).

Consider $f(x) \in L^2(D)$ where $D = [0,1]^d$. We approximate the integral $I = \int_D f(x)dx$ by:

$$\widehat{I}_N = \frac{1}{N}\sum_{i=1}^{N} f(X_i) \tag{3.1}$$

The error of such approximation is defined as

$$e_N(f) = \widehat{I}_N - I = \frac{1}{N}\sum_{i=1}^{N} f(X_i) - \int_D f(x)dx \tag{3.2}$$

3.1 Monte Carlo

In the simple Monte Carlo sampling method, X_i are drawn from the uniform distribution over the domain, $U(D)$. The law of large numbers states that:

$$P(\lim_{N\to\infty} \widehat{I}_N = I) = 1 \tag{3.3}$$

which guarantees that the Monte Carlo method always converges to the correct result as N increases.

If f has finite variance $\sigma(f)$, where:

$$\sigma^2(f) = \int_D (f(x) - I)^2 dx \tag{3.4}$$

then it is well known that $e_N(f)$ has mean 0 and variance $\sigma^2(f)/N$. Thus the error of Monte Carlo integration is $O(N^{-1/2})$, independent of dimension d. Variance reductions techniques, such as antithetic variables or control variates, can improve the constant of the error reduction rate but not the order itself.

3.2 Quasi–Monte Carlo

The Quasi–Monte Carlo method tries to improve the slow convergence of the Monte Carlo method by choosing X_i to uniformly fill the space D without the clustering seen in MC sampling.

To measure the uniformity of a sequence, we use a numerical measure called *discrepancy*. The most commonly used measure is the star discrepancy:

$$D_N^* = D_N^*(X_1, \ldots, X_N) = \sup_{a \in D} \left| \frac{1}{N} \sum_{i=1}^{N} 1_{0 \le X_i < a} - |[0, a)| \right| \tag{3.5}$$

The star discrepancy is found by maximizing over all anchored boxes $[0, a)$ the difference between the volume of the box and the proportion of points of the sequence that it contains. Other definitions of discrepancy are based on arbitrary boxes in $[0, 1]^d$, Morokoff and Caflisch (1994). A sequence $\{X_i\}_{i=1}^{\infty}$ is said to be *low discrepancy* or *quasi–random* if $D_N^*(X_1, \ldots, X_N) = O(log(N)^k/N)$ where k is a constant that may depend on d.

The Koksma–Hlawka inequality relates the sequence discrepancy and integration error via:

$$|\widehat{I}_N - I| \le D_N^*(X_1, \ldots, X_N) V_{HK}(f) \tag{3.6}$$

where $V_{HK}(f)$ is the total variation of f in the sense of Hardy and Krause.

This says the integration error is of size $O(log(N)^k/N)$ for large enough N, which is much better than the standard MC. The error estimate assumes that the variation of f remains bounded and constant as N changes. Initially, however, the error behaves like $O(N^{-1/2})$ for high dimensional integrands and from (3.6) we cannot deduce the transition value of N. This is due to the fact that D_N^* and $V_{HK}(f)$ are hard to estimate and the error bound is not very tight. In fact it is only tight for special functions that are specifically designed to exploit the discrepancies of a particular quasi–random sequence. Moreover, numerical experiments have shown that the transition point, where the error rate changes from $N^{-1/2}$ to $O(log(N)^k/N)$, grows exponentially with d as reported by Caflisch, Morokoff, and Owen (1997).

One of the most readily seen difficulties with quasi–random sequences arises in high dimensions, where on practical sample sizes, low–discrepancy sequences are no more uniform than random sequences. In fact, Akesson and Lehoczy (2000) note that some quasi–random sequences can perform worse than MC in high dimensional financial applications. Although it is very hard to assess the uniformity of low–discrepancy sequences in high dimensions, a necessary but not sufficient condition for such is the uniformity of low dimensional projections of the sequence. The Sobol and Niederreiter sequences used in this paper (generated using NAG C Library Mark 7) have good two dimensional projections in lower dimensions; however, in higher dimensions they can exhibit clustering and undesired regularity.

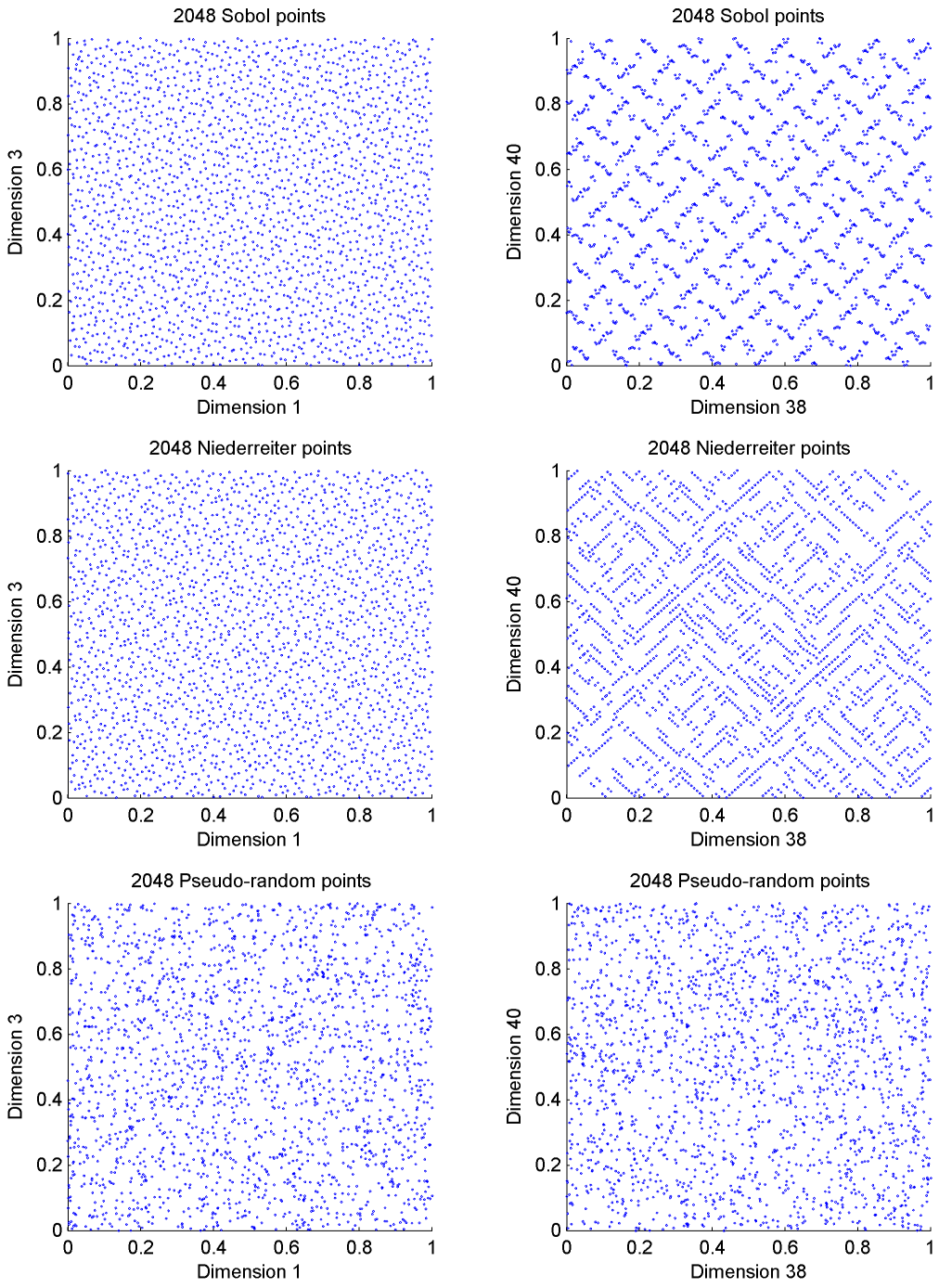

Figure 3.1: Two–dimensional projections of Sobol, Niederreiter and Pseudo–random sequences

Consider Figure 3.1, which plots two dimensional projections of Sobol, Niederreiter, and pseudo–random sequences. The pseudo–random sequences exhibit clustering regardless of dimensions chosen. The low–discrepancy sequences are much more uniform and structured, compared to pseudo–random sequences, in projections of coordinates 1 and 3. That regularity is retained in dimensions 38 and 40, but projection's uniformity is very poor. One must note that the uniformity can be improved by adding more points. When the number of points is taken to be 8192, the additional points fill in the gaps to make the projection practically uniform. However, it takes too long to achieve such uniformity across all projections when the number of dimensions is large. Thus we need to use a technique, such as Brownian bridge discretization, to contain most of the function variance within the first several dimensions of the sequence and make QMC much more effective by achieving greater uniformity at smaller sample sizes.

Chapter 4

Brownian Bridge

Simulating prices of a financial security is done by simulating characteristics and values of underlying securities and pricing the derivative security according to these values. For Mortgage–Backed Securities, the underlying security is the path of interest rates since they determine the prepayment strategy of the mortgage borrowers and thus the value of the MBSs. Since the interest rates are modeled as a continuous Wiener process (Brownian motion) $W(t)$, we would like to discretize the path over the interval $[0, T]$, where T is typically 30 years.

The number of points used in the discretization should be big enough to allow for the path to accurately describe the generating process, yet at the same time not too big to make calculations too time consuming. Also, a high degree of discretization, which results in a high dimensionality of the integrand, inhibits performance of quasi–random methods. For a problem such as valuation of MBS, the number of points d used in the discretization is typically 360 (one dimension per month).

There are numerous ways of generating such discretizations of the Brownian motion. Path generation methods are surveyed in Akesson and Lehoczy (2000). The standard discretization algorithm generates successive points of the path, while the Brownian bridge discretization, first introduced by Caflisch and Moskowitz (1995), generates midpoints of successive intervals, uniformly filling the path values. The great advantage of Brownian bridge discretization is that the alternate representation of the path concentrates most of the variance of the path within the first few components of the normals, thus effectively reducing dimension of the problem.

A simple random walk involving a series of discrete steps of fixed length (such as Brownian motion) is a particular case of a Markov process, which states that the distribution of future path values is conditional on the present and independent of the past. The problems discussed here can be expressed as stochastic differential equations dependent on the underlying Brownian motion $W(t)$ and once we've modeled the Brownian motion we can apply our differential models. From the properties of the Wiener process we can obtain the following identities:

For a given observation $W(r)$, the distribution of $W(s)$ for $r < s$ is a Normal distribution with mean $W(r)$ and variance $s - r$:

$$W(s) \sim N(W(r), s - r) \tag{4.1}$$

For two given observations $W(r)$ and $W(t)$, the distribution of $W(s)$ for $r < s < t$ is also a Normal

distribution:

$$W(s) \sim N((1-\beta)W(r) + \beta W(t), \beta(1-\beta)(t-r)) \qquad \beta = \frac{s-r}{t-r} \tag{4.2}$$

Let $X_i = \{X_i^0, X_i^1, \ldots, X_i^d\}$ be the path we wish to simulate (we'll drop the subscript in this section since we are dealing with one particular path only). We'll assume that $X^0 = 0$ and $\Delta t = X^i - X^{i-1}$ for $i = 1, 2, \ldots, 360$ is constant.

4.1 Standard Discretization

The standard discretization (SD) method generates the path value X^i as a step from the previous value X^{i-1} using (4.1):

$$X^i = X^{i-1} + \sqrt{\Delta t} \cdot z \tag{4.3}$$

where $z \sim N(0,1)$. Although this approach is very simple and takes the least amount of computational resources, it does not perform very well for QMC methods. As mentioned in the previous chapter, to obtain good equidistribution properties over all d dimensions takes an exponential amount of time with respect to d. Thus it becomes impractical to generate full paths using quasi–random sequences while generating parts of the path using quasi–random sequences, and the rest using pseudo–random suffers from the same problem that renders the MC method ineffective, its slow convergence rate.

4.2 Brownian Bridge Discretization

The standard Brownian bridge discretization (SBB) is based on the following two identities of a Markov process:

For any $j > i$, we can simulate a future value X^j given X^i, using:

$$X^j = X^i + \sqrt{(j-i)\Delta t} \cdot z \tag{4.4}$$

And for any $i < j < k$, we can simulate an intermediate value X^j given X^i and X^k using:

$$X^j = (1-\beta)X^i + \beta X^k + \sqrt{\beta(1-\beta)(k-i)\Delta t} \cdot z \qquad \beta = \frac{j-i}{k-i} \tag{4.5}$$

where $z \sim N(0,1)$ in (4.4), (4.5). These two formulas are essentially discretizations of (4.1) and (4.2).

The Brownian Bridge Discretization algorithm first generates X^d, given $X^0 = 0$. Then it generates $X^{d/2}$, given X^0 and X^d, and continues to fill the midpoints of subintervals in the following order: $X^0, X^d, X^{d/2}, X^{d/4}, X^{3d/4}, X^{d/8}, X^{3d/8}, X^{5d/8}, X^{7d/8}, \ldots$. Although this algorithm is defined for d being a power of 2, it is easily extendable to arbitrary d by choosing the nearest available integer index when a fraction index is encountered.

It is important to note that any order can be used to fill the points. The chosen ordering however, should maximize the variance contained in the first few path components, thus achieving maximal dimensional reduction. The breadth–first ordering previously described is the most common encountered in literature and has been used effectively to reduce dimension.

There has been several alternative orderings proposed, although none have been extensively studied. For instance Morokoff (1998) proposes a depth–first ordering:

$$X^0, X^d, X^{d/2}, X^{d/4}, X^{d/8}, \ldots, X^1, X^{3d/4}, X^{3d/8}, \ldots, X^3, \ldots, X^{d-1}$$

and conjectures that employing additional knowledge of the integrand in determining the path structure can lead to optimal dimension reduction.

To illustrate the importance of appropriate ordering, take a European option as an example. One can conclude that although nominally it is a d dimensional integrand, where d is the number of time steps until maturity, the effective dimension is 1 since the value of the option depends only on the terminal value of the underlying stock X^d. Thus the best ordering of points would have X^d in the first place (the order of the rest of the points is insignificant). For Mortgage–Backed Securities, however, we expect the early steps of the path to be more important for two reasons: the declining present value of a dollar and the declining number of remaining mortgages in the pool as functions of time. Hence, we would like to modify the SBB discretization to take into account these factors. A well–distributed sequence over the entire interval will perform worse than the same sequence that is better distributed over the interval of importance.

4.3 Weighted Brownian Bridge Discretization

The weighted Brownian bridge (WBB) discretization utilizes the same identities of the Markov process as SBB, except that the order of the points is chosen to maximize the discretization's efficiency. By adjusting the weight parameter one can vary the order of the chosen path components. Define $U(t)$ to be the 'utility' of generating a path value at t on interval $[t_a, t_b]$ where path values at t_a and t_b are known. The utility function is concave with $U(t_a) = U(t_b) = 0$, thus there exists a unique maximum. Also, we consider the case where the future point is unknown and we have an infinite time horizon. Here, $U(t)$ is defined on $[t_a, \infty)$ with $U(t_a) = 0$ and $U(t) \to 0$ as $t \to \infty$. The path component that will be generated by the Brownian bridge method is the node that maximizes the utility function on the given interval: $t = max_{t \in [t_a, t_b]} U(t)$.

The standard Brownian bridge discretization can be modeled using

$$U_{SBB}(t) = (t - t_a)(t_b - t) \tag{4.6}$$

which places equal importance on points in the beginning and end of the interval. Finding the maximum of the utility function t_* yields the midpoint of the interval:

$$t_* = \frac{t_a + t_b}{2} \tag{4.7}$$

To account for the discounting factor, we modify the 'utility' function by placing more weight toward the beginning of the interval. We achieve this by multiplying $U(t)$ by the continuous compound factor $e^{-\lambda t}$ where λ is a free parameter specific to each integrand. Thus, we get the following expression for $U(t)$:

$$U_{WBB_\lambda}(t) = \begin{cases} (t - t_a)(t_b - t)e^{-\lambda t}, & t_b \text{ is known;} \\ (t - t_a)e^{-\lambda t}, & \text{otherwise.} \end{cases} \tag{4.8}$$

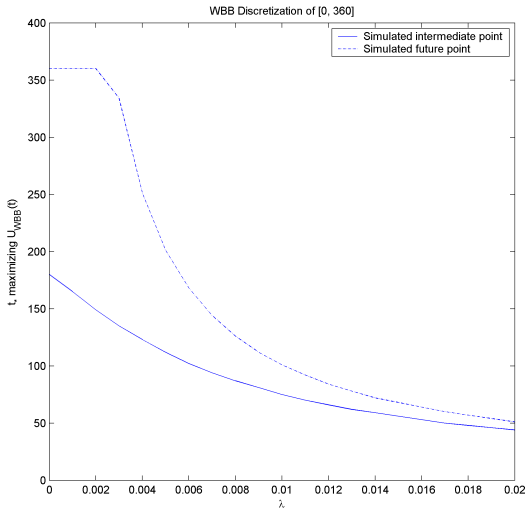

Figure 4.1: WBB_λ discretization as a function of λ

To find the point that maximizes the utility we solve $U'(t) = 0$, which gives us a value of t_* dependent on the free parameter λ :

$$t_* = \begin{cases} \frac{t_a+t_b}{2} + \frac{1}{\lambda}(1 - \sqrt{\lambda^2(t_b - t_a)^2 + 1}), & t_b \text{ is known;} \\ t_a + \frac{1}{\lambda}, & \text{otherwise.} \end{cases} \tag{4.9}$$

In effect, WBB_λ discretization represents a generalization of path generation methods presented above. Note that the expression inside the brackets is always negative, thus $t_* < \frac{t_a+t_b}{2}$. Via straightforward algebraic manipulation it can be shown that as $\lambda \to 0$, $t_* \approx \frac{t_a+t_b}{2} - \frac{1}{8}\lambda(t_b - t_a)^2$, and the discretization produced by WBB_λ approaches the one generated by SBB. This can intuitively be seen from the fact that as $\lambda \to 0$, $e^{-\lambda t} \to 1$, and $U_{WBB_\lambda}(t) \to U_{SBB}(t)$. Conversely, as $\lambda \to \infty$, $t_* \approx t_a + \frac{1}{\lambda}$, and WBB_λ produces a standard discretization given by (4.3). Intermediate values of λ allow to achieve a discretization that benefits from the properties of Brownian bridge discretization, i.e. it still concentrates most of its variance within the first few variables, and at the same time by fine–tuning the parameter λ we can tailor the discretization to work best on the particular integrand. The effect of choice of λ is summarized by Figure 4.1, which shows the choice of points when discretizing $[0, 360]$. A future point is the point that is chosen when only the past value of the path is known and $U_{WBB_\lambda}(t)$ is modified to disregard the terminal point, i.e. $U_{WBB_\lambda}(t) = (t - t_a)e^{-\lambda t}$, while an intermediate point is the point that is chosen when both past and future values of the path are known and the full form of utility function given in (4.8) is used.

Case 1: $\lambda = 0$ When $\lambda = 0$, neither earlier nor later points have more weight. The first point is chosen given only x^0, thus it is a future point, namely x^{360} (the terminal point of the interval). The next point is generated given x^0 and x^{360}, thus it is an intermediate point and it is x^{180} (the midpoint of the interval). All subsequent points will be intermediate midpoints and this algorithm exactly replicates the behavior of standard Brownian bridge.

Case 2: $\lambda \to \infty$ When λ becomes very large, all of the weight is concentrated in the earlier points. At the limit, the future point chosen given x^0 is x^1. Since there are no intermediate points on $[0, 1]$, the next point chosen will be x^2 and so on. This essentially replicates the standard discretization method.

Case 3: Non-boundary λ The most effective choices of λ for our numerical examples lie in $[0.005, 0.015]$. For example, take $\lambda = 0.01$; its future simulated point on [0,360] is 101 and intermediate point is 75. Similarly to the above, the first point is chosen given only x^0, yet it is not a terminal point of the interval like in SBB but x^{101}. The second point is an intermediate point of [0, 101] and it is $x^{0+(101-0)*\frac{75}{360}} = x^{21}$. The third point, however, will be a future point again since it is generated on [101,360] given only x^{101} and it is $x^{101+(360-101)*\frac{101}{360}} = x^{174}$.

As previously mentioned, λ is integrand specific and we would like to choose a λ that will maximize the performance of the WBB_λ discretization for that particular integrand. Although this choice can be done by trial and error, i.e. comparing performances of various discretization, ANOVA decomposition allows a more precise and quantitative approach.

Chapter 5

Estimating Effective Dimension

In this section we will quantify the notion of *effective dimension* and estimate the dimension reduction effect of various Brownian bridge discretization methods. As previously noted, many problems with nominal high dimensions can be effectively computed using a Quasi–Monte Carlo method. Although the error bounds analysis suggests that the sample sizes required to attain the improved convergence rates of QMC are not feasible, many numerical examples use QMC to obtain good accuracy at relevant sample sizes. This leads us to believe that the integrand may be of lower *effective dimension*. For certain problems in computational finance, Caflisch, Morokoff, and Owen (1997) in fact suggest that the integrand may be a sum of lower–dimensional integrands. It is not surprising, then, that the good low dimensional equidistribution properties of low–discrepancy sequences give QMC methods good performance on these problems at relevant sample sizes.

One of the methods to reduce the effective dimension is to use the Brownian bridge discretization technique. Although the standard discretization offers significant improvement to the QMC method, further gains can be achieved by altering the representation as described in the previous chapter. Thus in order to choose the best WBB_λ representation as a function of λ, it becomes essential to be able to measure and compare the various discretizations. Most of the literature dealing with the topic of dimensionality reduction in multi dimensional integration problems associates the dimension reduction with the increased convergence rate of the Brownian bridge. We could adapt the same approach and choose λ to maximize the convergence rate of the associated WBB_λ representation; however, this approach is very cost ineffective and imprecise. Since the discretization would depend on the choice of the particular integrand, finding a suitable λ for each integrand by comparing the associated convergence rates would take too long to justify the computational costs. Also, estimating convergence rates is a very imprecise matter that limits the confidence of our choice of λ.

A more heuristic approach is to estimate the effective dimension of the integrand as given by the WBB_λ discretization. Essentially, this approach involves decomposing the integrand into 2^d functions, one for each subset of $\{1, 2, \ldots, d\}$, dependent only on the variables in the subset. Then, by looking at the contribution of each subset to the total variance, we can quantify the importance of each subset and hence quantify the effective dimension of the integrand. This approach works well on functions that can be decomposed analytically and it has been studied in Owen (1998) and Liu and Owen (2003).

For the integrand used in the numerical examples in this paper we would like to generate a discrete approximation to the decomposition and use it to estimate the variances attributed to each subset. It can also be shown that the estimate becomes more accurate as the number of sampled points is increased.

5.1 ANOVA Decomposition

The notation is taken from Owen (1998), Liu and Owen (2003), and Wanga and Fang (2003). Consider $f(x) \in L^2[0,1]^d$ where $x = (x^1, \ldots, x^d)$. The expectation of f is $I = I(f) = \int f(x)dx$ and the variance is $\sigma^2 = \sigma^2(f) = \int (f(x) - I)^2 dx$. To rule out trivialities we assume $\sigma^2 > 0$.

Let $u \subseteq D = \{1, \ldots, d\}$ be a subset of D, $-u$ is the complement of u, $D - u$, and $|u|$ is the cardinality of u. Denote by x^u the $|u|$-tuple consisting of components x^j with $j \in u$. For $x, y \in [0,1]^d$, the expression $f(x^u, y^{-u})$ means $f(p)$ where $p \in [0,1]^d$ s.t. $p^i = x^i$ for $i \in u$ and $p^i = y^i$ for $i \in -u$. When $f(x^u, x^{-u}) = f(x^u, y^{-u})$ for all $x, y \in [0,1]^d$, we say that f depends on x only through x^u.

Using analysis of variance (ANOVA) decomposition we would like to write:

$$f(x) = \sum_{u \subseteq \{1,\ldots,d\}} f_u(x) \tag{5.1}$$

where $f_u(x)$ is a function dependent only on x^u. $f_u(x)$ is defined recursively by subtracting from f what can be attributed to strict subsets of u and averaging over $-u$

$$f_u(x) = \int \left(f(x) - \sum_{v \subsetneq u} f_v(x) \right) dx^{-u} = \int f(x) dx^{-u} - \sum_{v \subsetneq u} f_v(x) \tag{5.2}$$

where $f_\emptyset = I$ is the constant function using usual conventions.

ANOVA analysis states that the total variance of f can be decomposed into parts attributed to each subset u:

$$\sigma^2 = \sum_u \sigma_u^2 \tag{5.3}$$

where the total variance of f is $\sigma^2 = \sigma^2(f) = \int (f(x) - I)^2 dx$, $\sigma_u^2 = \sigma^2(f_u) = \int f_u(x)^2 dx$ is the variance of f corresponding to the subset u and $\sigma_\emptyset^2 = \sigma^2(f_\emptyset) = 0$.

5.2 Effective Dimension

In many numerical examples, it has been noted that the asymptotic convergence rates of the quasi–monte carlo method take effect much sooner than the theoretical bounds. This leads us to believe that the dimension of the problem at hand is not the nominal dimension d but much smaller.

The effective dimension of f, in the *superposition* sense, is the smallest integer d_S such that

$$\sum_{|u| < d_S} \sigma^2(f_u) \geq p \cdot \sigma^2(f) \tag{5.4}$$

The effective dimension of f, in the *truncation* sense, is the smallest integer d_T such that

$$\sum_{u \subseteq \{1,2,\ldots,d_T\}} \sigma^2(f_u) \geq p \cdot \sigma^2(f) \tag{5.5}$$

where p is the critical level and 0.99 is a common choice for p.

As an example, consider functions that are nearly sums of one dimensional integrands:

$$f(x) = f_\emptyset + f_1(x^1) + \ldots + f_d(x^d) \tag{5.6}$$

Using the definitions (5.4) and (5.5), f has dimension 1 in superposition sense and d in truncation sense. It has been noted in Caflisch, Morokoff, and Owen (1997) and other papers that many integrands in computational finance are essentially sums of one dimensional integrands. ANOVA decomposition of multiplicative, isotropic and other special functions have also been extensively studied, see Owen (2003). They are used to test performance of quasi–random methods, but their importance in computational finance is rather limited.

The difficulty with such a definition is that the choice for p can greatly vary the resulting effective dimension. From the results obtained through our numerical experiments, available in Tables 5.1 and 5.2, changing p from 0.99 to 0.95 changes effective dimension from 25 to 9 for the linear problem and from 25 to 17 for the nonlinear problem using SBB. This also highlights the problem that the value of the effective dimension by itself does not provide insight into the overall variance reduction and the discretization's effectiveness. For instance compare two integrands, f_a and f_b, where using $p = 0.99$, the functions have effective dimensions 20 and 30 respectively, and for $p = 0.95$ the respective effective dimensions are 15 and 10. Fewer dimensions of f_b are needed to account for 95% of its variance, while f_a has lower dimensionality in traditional sense ($p = 0.99$). Based on this information it is very challenging to assert for which integrand QMC with a Brownian bridge discretization will produce superior results or what dimensionality of a quasi–random sequence should be used. Thus we would like to construct a parameterless definition of effective dimension based on variance attributed to successive dimensions or the dimension distribution density function.

5.3 Dimension Distribution

Here, we define an underlying notion to the effective dimension of a function, the dimension distribution, which describes the variation effect due to each dimension. Formally, the dimension distribution of f, in the *superposition* sense, is the discrete probability distribution of $|U|$ with $Pr(U = u) = \sigma_u^2/\sigma^2$. The distribution has density function

$$\nu_S(s) = \sum_{|u|=s} \sigma_u^2/\sigma^2, \quad s = 1, \ldots, d \tag{5.7}$$

The dimension distribution of f, in the *truncation* sense, is the discrete probability distribution of $\max\{j|j \in U\}$ with $Pr(U = u) = \sigma_u^2/\sigma^2$. The distribution has density function

$$\nu_T(s) = \sum_{\max\{j|j\in u\}=s} \sigma_u^2/\sigma^2, \quad s = 1, \ldots, d \tag{5.8}$$

Thus the dimension distribution is a function $F_\nu(s) = \sum_{t=1}^{s} \nu(t)$. This distribution function satisfies the three characteristic properties of a distribution function: $F_\nu(0) = 0$, $F_\nu(d) = 1$, and $F_\nu(s)$ is nondecreasing function of s. Essentially the dimension distribution can be thought of as the cumulative variance of the function explained by the first s dimensions.

5.4 Effective Dimension Order

As previously mentioned, describing the effective dimension using a single p percentile does not provide the necessary information to be able to judge the effectiveness of a QMC method used to evaluate the integral. A measure that describes the overall behaviour of $F_\nu(s)$ and from which we can deduce the various percentiles is needed. Although one could simply store $F_\nu(s)$ for all s, once again we run into the difficulty of making distribution comparisons. A natural approach in this situation is to assess the order of the dimension distribution by defining the effective dimension order (EDO). A function has a EDO r if

$$1 - F_\nu(s) \sim \alpha s^{-r} \qquad s \in [1, d] \tag{5.9}$$

which means that the function $1-F_\nu(s)$ behaves approximately like αs^{-r} for a certain choice of constants α and r. Using such a definition allows one to describe the behavior of the cumulative variability of a function over all dimensions. Moreover, from the function's EDO one can obtain the effective dimension in the sense of (5.4) or (5.5) for any value of percentile p.

Although this definition is very similar to a definition of order of numerical methods, there are some subtle yet important differences. First, since the number of dimensions is finite, we do not let $s \to \infty$. Moreover, the convergence properties of a good discretization scheme will be largely determined by the variance reduction in the initial dimensions. Thus, when estimating EDO, it is beneficial to limit s to lie in the first dimensions of importance. Our experiments have shown that for a 360–dimensional MBS, when evaluated using a standard Brownian bridge discretization technique, estimating EDO with $s \in [0, 24]$ was enough to capture accurately the structure of the function and the performance of SBB and WBB$_\lambda$. Second, since the value of s is finite, one cannot disregard the value of the constant term α. Unlike in convergence analysis of numerical methods, where a higher–order method will eventually produce better results as the number of iterations increases, no matter what the constants are, the constant term may be the reason a function with lower EDO has smaller residual variability for dimensions in the relevant range.

A power law relationship between two variables translates into a linear relationship between the logs of both variables:

$$1 - F_\nu(s) \sim \alpha s^{-r} \iff \log(1 - F_\nu(s)) \sim \log \alpha - r \log s \tag{5.10}$$

where α and r are constants that can be obtained by fitting a straight line using a least–squares method on a $\log - \log$ plot of $1 - F_\nu(s)$ vs s.

An alternative definition to the EDO was to use the exponential law instead of power law, i.e., $1 - F_\nu(s) \sim \alpha e^{-rs}$. An advantage of such a definition is the inherent dimensionless nature of the exponential function. Since the $-rs$ term cannot have any dimension associated with it, $1/r$ can be

considered as the natural scaling parameter for the expression. Similarly, an exponential relationship between two variables translates into a linear relationship between the first variable and the log of the second:

$$1 - F_\nu(s) \sim \alpha e^{-rs} \iff \log(1 - F_\nu(s)) \sim \log\alpha - rs \tag{5.11}$$

and similarly to (5.10), α and r are constants that can be obtained by fitting a straight line using a least–squares method on a log-linear plot of $1 - F_\nu(s)$ vs s. However, numerical experiments do not support such a model and the linear relationship between $1 - F_\nu(s)$ and s was obtained on a log-log plot.

The above formulas can be related to the traditional definition of effective dimension by substituting $F_\nu(s) = p$ and solving for the effective dimension s in equations (5.10), (5.11) respectively to obtain $s = (\frac{1-p}{\alpha})^{-\frac{1}{r}}$ and $s = -\frac{1}{r}\log\frac{1-p}{\alpha}$

5.5 Decomposition Approximation

Since any definition of effective dimension requires estimates for σ_u^2, $\widehat{\sigma}_u^2$, we need to be able to generate approximation to f_u, $\widehat{f}_u$, within a required error tolerance level. However, since any quadrature method used for the integration uses a finite number of function values, we only estimate $\widehat{f}_u$ at a finite number of points. The algorithm to approximate functional ANOVA decomposition closely follows the definition (5.2) and thus is recursive on the cardinality of u — when approximating $f_u(x^u)$, all $f_v(x^v)$ for $|v| < |u|$ should have been previously approximated.

Consider a simple case where $f(x)$ is one dimensional in the superposition sense. We'll generate the approximation using the same sequence $\{X_i\}_{i=1}^N$ that was used to approximate the integral $\widehat{I}_N$, see equation (3.1). From the definition, we estimate $f_\emptyset$ as the computed value of the integral, $\widehat{f}_\emptyset = \widehat{I}_N$. The next step is to approximate $f_{\{s\}}$; from the definition (5.2) we obtain:

$$\widehat{f}_{\{s\}}(x^s) = \int \left(f(x) - \widehat{f}_\emptyset\right) dx_1 \dots dx^{s-1} dx^{s+1} \dots dx^d = \int f(x) dx^1 \dots dx^{s-1} dx^{s+1} \dots dx^d - \widehat{f}_\emptyset \tag{5.12}$$

Since we need to discretize $\widehat{f}_{\{s\}}(x^s)$ at a finite number of points $x_1^s, \dots, x_k^s$, we choose the points to match the quadrature method use in the evaluation of $\widehat{\sigma}_u^2 = \int \widehat{f}_u(x)^2 dx$. So for a simple midpoint rule and $x^s \in [0,1]$, $x_i^s = \frac{1}{k}(i - \frac{1}{2})$. Estimating $\widehat{f}_{\{s\}}(x_i^s)$ directly using a Quasi–Monte Carlo method is difficult since it requires holding one component of the sequence fixed and it becomes even harder in higher dimensions to achieve desired sequence uniformity. Instead, we can evaluate the integral $\int f(x) dx^1 \dots dx^{s-1} dx^{s+1} \dots dx^d$ for a fixed x^s using a subset of $\{X_i\}$ by taking terms where $X_i^s \subset (x^s - \epsilon, x^s + \epsilon)$. This approach, however, will discard many sequences for which none of the terms are within ϵ of x^s. Our numerical experiments have shown that by taking $\epsilon = 1/2k$ (essentially dividing the interval into k 'bins'), we use all of the sequences (since each term falls within a bin) and achieve better estimates. Since x_i^s is a normally distributed random variable and can take on values in $[-\infty, \infty]$, we would like to transform it into a uniform distribution on $[0,1]$ via a change of variables $y^u = \Phi(x^u)$, where Φ is the normal cumulative distribution function applied componentwise.

There are two difficulties inherent in the approximation of an ANOVA decomposition — the exponential number of sets examined and the accuracy of the approximation. Computing approximations to 2^d functions f_u and estimating their respective variances σ_u becomes impractical for large d as is the

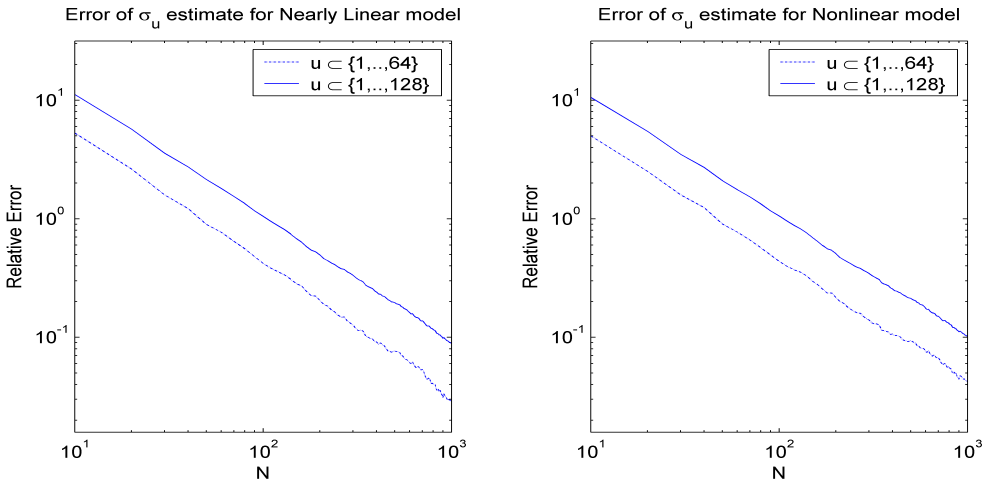

Figure 5.1: Relative error of σ_u^2 estimate

case in computational finance where d is as high as 360. Given that each decomposition function has to be discretized at k points for each dimensions and assuming each discretization point takes N samples to achieve reasonable accuracy, the entire task requires $O(N(2k)^d)$ computational work. Moreover, the memory requirements become so demanding that it becomes impossible to estimate σ_u within a desired tolerance level for $|u| > 2$.

However, when we consider functions that are one dimensional in the superposition sense (functions that are sums of one dimensional functions, for instance), this reduces the number of possible subsets of $\{1, \ldots, d\}$ from 2^d to d and makes the approximation task of order $O(Ndk)$. Our results agree with Caflisch, Morokoff, and Owen (1997), whose nearly linear and nonlinear numerical examples had respectively 99.96% and 94.1% of the structure contained in one dimension. For both of these problems we've verified that $\min_{|u|=1} \sigma_u^2 \gg \max_{|u|=2} \sigma_u^2$ when considering the first 32 components of the Brownian bridge path. Although the choice of the cutoff dimension at 32 seems arbitrary, it will be shown later that it is enough to contain 96% of the total function variance.

5.6 Error Analysis

In order to be confident in the results of the approximation and inferences made from them, we need to look at the behavior of $F_\nu(s)$ as we increase the number of samples of decomposition functions f_u. Let the estimate of $F_\nu(s)$ be $F_{\widehat{\nu}}(s)$ and define the relative error of such an estimate as:

$$e_\nu(s) = \left| \frac{F_\nu(s) - F_{\widehat{\nu}}(s)}{F_\nu(s)} \right| = \left| 1 - \frac{F_{\widehat{\nu}}(s)}{F_\nu(s)} \right| \tag{5.13}$$

Caflisch, Morokoff, and Owen (1997) compute the $F_\nu(1)$ values using results from numerical integration using Latin hypercube. We'll use these estimates to perform error analysis on the approximation algorithm presented in the previous section. Figure 5.1 contains the error estimate $e_\nu(1)$ as a function of the number of samples per function value for both linear and nonlinear problems. The top line

	SBB	$WBB_{\lambda=0.005}$	$WBB_{\lambda=0.01}$	$WBB_{\lambda=0.015}$	$WBB_{\lambda=0.02}$
$ED_{p=0.99}$	25	21	23	25	25
$ED_{p=0.95}$	9	5	4	4	7
EDO	$0.71s^{-1.31}$	$0.36s^{-1.21}$	$0.17s^{-0.92}$	$0.17s^{-0.92}$	$0.26s^{-1.01}$

Table 5.1: Effective dimensions for SBB and WBB_λ of the nearly linear problem using $ED_{p=0.99}$, $ED_{p=0.95}$, and EDO

	SBB	$WBB_{\lambda=0.005}$	$WBB_{\lambda=0.01}$	$WBB_{\lambda=0.015}$	$WBB_{\lambda=0.02}$
$ED_{p=0.99}$	25	25	20	21	25
$ED_{p=0.95}$	17	7	4	4	5
EDO	$0.73s^{-0.99}$	$0.40s^{-0.98}$	$0.21s^{-1.03}$	$0.16s^{-0.93}$	$0.17s^{-0.82}$

Table 5.2: Effective dimensions for SBB and WBB_λ of the nonlinear problem using $ED_{p=0.99}$, $ED_{p=0.95}$, and EDO

on both plots represents the case when we only look at $u \subset \{1, \ldots, 64\}$ while the bottom line is for $u \subset \{1, \ldots, 128\}$. Obviously, looking at a greater number of sets produces more precise estimates, yet it comes at increased computational cost that is exponential with the cardinality of u. For all cases, we confirmed that, using the approximation algorithm, $F_{\widehat{\nu}}(1) \to F_\nu(1)$ at a rate of $O(N^{-1})$, and the algorithm can be used to estimate σ_u^2 within any error tolerance level.

5.7 ANOVA Results

We now discuss the results of estimating dimensionality of the two integrands defined in the chapter on Mortgage–Backed Securities. The results are presented in Tables 5.1 and 5.2, one for each example. ED refers to the effective dimension given by equations (5.4) and (5.5) using $p = 0.99$ unless noted otherwise. EDO refers to the effective dimension order given by equation (5.9). For a full plot of $1 - F_\nu(s)$ vs. s and λ refer to Figure 5.2.

There are several conclusions that can be drawn from the results:

The WBB_λ discretization has lower dimensionality than the SBB discretization as measured by ED

Using WBB_λ discretization for $\lambda \in \{0.005, 0.01, 0.015, 0.02\}$, we've been able to achieve further dimensionality reduction compared to SBB. As it will be shown in the following chapter, this translates into increased convergence rates for the WBB_λ method. WBB_λ was especially effective on the nonlinear problem where we were able to reduce the effective dimension of the nonlinear integrand from 25 in the SBB case to 20 for $WBB_{\lambda=0.01}$. For $p = 0.95$ the effect is more dramatic: the reduction is from 17 to 4. This is also reflected in the EDO structure where both the constant term and rate are lower for WBB than for SBB. For the linear problem, however, SBB has a superior EDO rate, which is partially offset by WBB_λ having a smaller constant term. From Figure 5.2, we can see that, although the slope of the unexplained variability of SBB is steeper, WBB_λ explains more variability on the first 24 dimensions due to its smaller constant, thus making the discretization more effective, as will be seen in the next chapter.

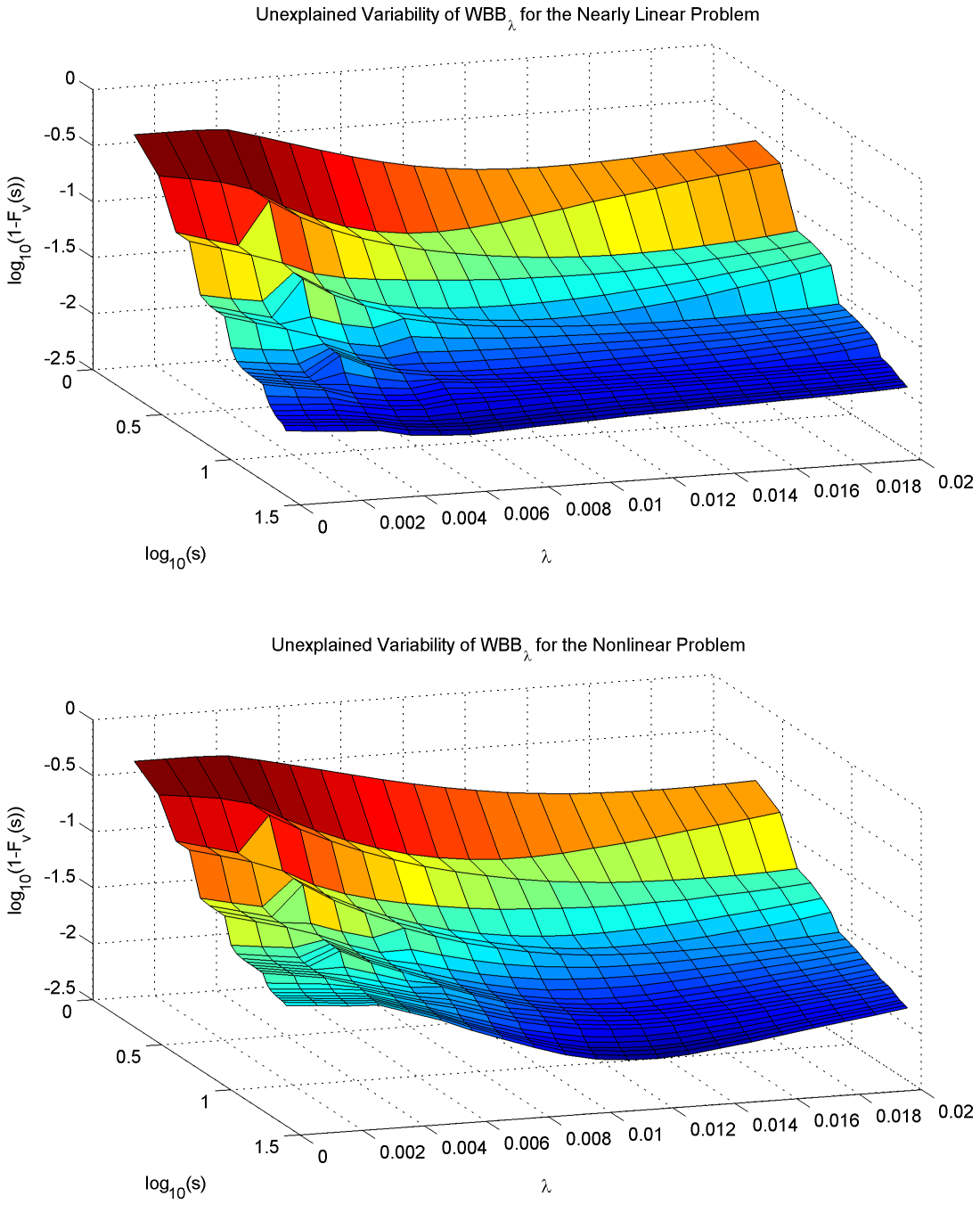

Figure 5.2: Unexplained variability of WBB_λ

The ED value varies greatly with the choice of percentile p The effective dimension, as given by the standard definition, varies significantly for small changes in p. Although the standard choice of $p = 0.99$ can always be used, this highlights the definition's inability to capture the overall structure of variance reduction.

ED is not consistent — relative dimensionality changes with p Due to the difference in slopes of $\log(1-F_\nu(s))$ among various Brownian bridge discretizations, a change in p can make a different discretization superior in the ED sense. Since the overall structure of $1-F_\nu(s)$ is not known, using ED to compare performance of discretizations is unreliable.

EDO matches the behaviour of ED Using EDO to express the relative variance attributed to each dimension allows one easily to estimate the ED value for any percentile p. The quality of such estimate depends on the closeness of the fit. In our numerical examples, since the $\log-\log$ plot of $\log(1-F_\nu(s))$ vs. s was essentially linear, we obtained a good fit to our model, making estimates for ED very precise.

The constant term α is significant in the EDO expression In the linear case, the SBB discretization had a higher order compared to any WBB_λ. However, due to the high constant α, $1-F_\nu(s)$ for SBB is greater than for WBB_λ. Thus, despite its higher order, SBB performs worse at explaining variability than WBB_λ.

Chapter 6

Numerical Results

The previous chapter established that, by modifying the standard Brownian bridge discretization to account for the properties of the integrand, one can achieve better dimension reduction. The purpose of this chapter is to investigate the relationship between dimensionality reduction and convergence rates of Monte Carlo integration methods. Caflisch and Moskowitz (1995), Paskov and Traub (1995), Morokoff (1998) and Caflisch, Morokoff, and Owen (1997) have shown that the QMC method with Brownian bridge discretization used to reduce the effective dimension have consistently outperformed the straightforward MC and QMC methods, although Papageorgiou (2002) reported otherwise, producing an example of a digital option on which the results obtained using the Brownian bridge discretization were considerably worse than the ones obtained using the standard discretization.

The MATLAB v6.5 R13 built–in random number generator *rand* was used to generate pseudo–random sequences. The low–discrepancy sequence generators used in the simulations are the NAG C Library Mark 7 Sobol and Niederreiter sequence generators. These generators are available through calls to the *g05yac* and *g05ybc* functions for uniform and normal distributions respectively. One of the constraints of the NAG Library is that the dimensionality of quasi–random sequences can be no greater than 40. A simple method to overcome this restriction is to generate the first 40 dimensions of the sequence using a quasi–random generator and the other 320 dimensions using a pseudo–random generator. Although, as is evident from the numerical results, this procedure is not very effective for standard path generation methods, the Brownian bridge discretization makes the most use of the first 40 quasi–random numbers by concentrating most of the functional variance within the initial dimensions, making the pseudo–random numbers much less significant.

First, we'll compare the standard Monte Carlo method using the pseudo–random numbers (denoted MC on Figures A.1 and A.2), Quasi–Monte Carlo (QMC) method (using quasi–random numbers for the first 40 dimensions and pseudo–random for the rest) with the standard path discretization (denoted SD), and Quasi–Monte Carlo method using the same sequence as QMC but incorporating the standard Brownian bridge discretization (denoted SBB). Second, we'll consider the performance of the weighted Brownian bridge discretization (denoted WBB_λ on Figures A.3, A.4, A.5 and A.6) using different parameters and compare it to the standard Brownian bridge discretization technique (denoted SBB).

When plotting the log of the relative error, every time the computed value is exact, the plot ap-

proaches ∞ and makes estimating the convergence rate quite challenging. One cannot simply sample the value of the error at a subset of points and fit a straight line using a least–squares method. The error at some of the points in the subset may be quite small even though the method has not completely converged. To achieve an accurate estimate for the order of the method we fit a straight line that, roughly speaking, skims the top of the error curve. This is done by sampling the error at a logarithmically distributed set of points, but taking the error value as a maximum error in a small neighbourhood around each point. The convergence rates displayed in the graph legends represent these estimates and allow us to assess various discretization methods.

For the antithetic computations, N is the number of times the antithetic value of the integrand $(f(x) + f(1-x))/2$ is evaluated, this requires $2N$ function evaluations. This is a significant penalty on the antithetic method not reflected in the graphs, since function evaluation accounts for most of the CPU time. However, it is not the purpose of this paper to discuss the efficiency of the antithetic variance reduction method, rather to judge the efficiency of the WBB_λ discretization method compared to SBB when used by itself or in conjunction with other variance reduction techniques.

6.1 MC, QMC, and QMC with SBB

First, consider the numerical results for the nearly linear problem in Figure A.1. The solid line (MC) in all plots represents the straightforward Monte Carlo method using pseudo–random sequences. The error behaves as expected, decreasing at a rate of $N^{-.47}$ for the simple case and $N^{-.57}$ for the antithetic run, which is comparable with the theoretical rate of $N^{-.5}$. Also, the constant term of the antithetic run is lower: 0.0577 vs 0.0017. The dashed line (SD) in all plots represents the Quasi–Monte Carlo method with the standard path discretization. As expected, SD with both Sobol and Niederreiter quasi–random sequences demonstrated increased convergence rates compared to MC for both the nearly linear and the nonlinear integrands. Still, MC had smaller error on the relevant domain for Niederreiter SD and Niederreiter antithetic SD by virtue of having a smaller constant term. SBB, however, is able to improve the respective SDs by attributing most of function variance to the first dimensions of the quasi–random sequence and achieve superior error rates. Sobol sequence, compared to Niederreiter, exhibited a more significant improvement in the performance of SBB method, reducing the error rate from $N^{-.59}$ to $N^{-.76}$ using standard evaluation and from $N^{-.51}$ to $N^{-.71}$ using antithetic computation. The respective rates for Niederreiter are from $N^{-.63}$ to $N^{-.69}$ using standard evaluation and from $N^{-.59}$ to $N^{-.68}$ using antithetic variables.

In Figure A.2 we show the error for the Nonlinear problem using the same methods as above. Here the MC method performed worse than in the linear case, achieving $N^{-.44}$ and $N^{-.34}$ for standard and antithetic computations respectively. Similarly, SD has a higher order than MC, although outperformed by MC for small sample sizes on Sobol antithetic, Niederreiter, and Niederreiter antithetic cases. While in the linear antithetic case the order of the SBB was only slightly larger relative to SD, the Brownian bridge discretization in the nonlinear problem using antithetic computation was able to significantly reduce the effective dimension and increase the order of convergence (from $N^{-.43}$ to $N^{-.78}$ for Sobol sequence and from $N^{-.57}$ to $N^{-.81}$ for Niederreiter).

6.2 SBB and WBB

Next, we consider the effect of modifying the standard Brownian bridge discretization, as described in chapter 4, on the performance of WBB_λ discretization. Figures A.3, A.4, A.5, and A.6 contain the test results for nearly linear and nonlinear problems using Sobol and Niederreiter sequences. In each figure, we consider the SBB representation vs. WBB_λ for various parameters λ as well as the effect of antithetic sampling. From the results of ANOVA analysis in Figure 5.2, we would expect the WBB_λ discretization to be most effective on the nonlinear problem for $\lambda \in [0.01, 0.015]$. Although for the nearly linear problem the same values of λ enable WBB_λ to account for considerably more variability than SBB with a limited number of dimensions ($s < 12$), the advantage becomes negligible for $s = 24$. As previously mentioned, comparing convergence rates is not a very precise method to judge effectiveness of various discretizations, yet it allows us to make several conclusions regarding the performance of WBB_λ. It is important to note that these results match the ones we've obtained through ANOVA decomposition analysis.

Generally, we can conclude that WBB_λ outperforms SBB on both examples using both low–discrepancy sequences. Any choice of $\lambda \in [0.005, 0.02]$ improves upon the convergence rate of SBB, with $\lambda \in [0.01, 0.015]$ as the most effective. This suggests that the modified Brownian bridge parametrization is able to capture the structure of the integrand more effectively, which corresponds to the general conclusion in the chapter on ANOVA decomposition analysis. One may note that the WBB_λ method using a Sobol sequence benefited more on the nearly linear problem (estimated convergence rate increase of 0.04 for Sobol sequence vs 0.02 for Niederreiter sequence), while the one based on a Niederreiter sequence benefited more on the nonlinear problem (estimated convergence rate increase of 0.06 for a Sobol sequence vs 0.08 for a Niederreiter sequence). Although the error reduction is not as dramatic as one might expect, it is significant enough to show a clear relationship between ANOVA and our numerical results results.

When combining any Brownian bridge discretization method with antithetic sampling we achieve a significant reduction in error size. For the nearly linear problem, SBB with antithetic sampling was able to retain the convergence rate ($N^{-.76}$ vs $N^{-.71}$ for the Sobol sequence and $N^{-.69}$ vs $N^{-.68}$ for the Niederreiter sequence) and dramatically improve the constant term. For the nonlinear problem, antithetic sampling improved the convergence rate ($N^{-.71}$ vs $N^{-.78}$ for the Sobol sequence and $N^{-.65}$ vs $N^{-.81}$ for the Niederreiter sequence) while slightly reducing the constant term. Thus we can conclude that the antithetic sampling is able to capture some high–dimensional antisymmetric structure of the nonlinear integrand. Unlike for the standard computation, WBB_λ was unable to improve the convergence rate of SBB when antithetic sampling was used. Although the error is smaller over the relevant sample size, by removing one dimensional linear elements of the integrand, antithetic sampling decreased the convergence rate of WBB_λ.

Chapter 7

Distributed Environment

The main advantage of moving a computational process to a distributed environment is to achieve speedup by leveraging the CPU power of multiple processors in a clustered system. There are numerous parallel computing tools and libraries available for various platforms, primarily designed to be used with Fortran and C. However, new technologies and standards are being developed and implemented that enable communication between remote objects written in high level languages such as C#, C++, Visual Basic, and Java. In fact standards such as SOAP and CORBA allow objects written in different languages to communicate with each other.

One of the emerging technologies in the high performance computing (HPC) arena is Microsoft .NET. Initially thought of as unable to compete with traditional message passing libraries such as Message Passing Interface (MPI) and Parallel Virtual Machine (PVM), HPC environments developed within Microsoft .NET framework are gaining recognition as promising alternatives.

Microsoft Application Center 2000, a cluster configuration application for .NET framework, allows users to create HPC clusters and provide seamless access to these resources. Application Center 2000 provides load balancing and fault tolerance by steering new applications away from busy processes and being able to recover from errors and node failures. One particular feature of interest is Component Load Balancing (CLB), which can spread the execution of COM+ components across multiple servers. The overhead incurred by the Application Center is offset by ease of programming and system integration. Application Center 2000 was extensively studied by Lifka, Walle, Zaloj, and Zollweg (2002) and their conclusion was that the Application Center 2000 is best used for independent applications that do not require interprocess communication.

Similar conclusions were drawn by Solms and Steeb (1998), who developed an application using the object–oriented middleware standard CORBA and Java. Although one can achieve slightly faster results using PVM with C, this advantage is severely offset by the complexities of sending and receiving abstract data types.

7.1 Microsoft .NET Remoting

7.1.1 .NET Remoting

Microsoft .NET Remoting provides a very powerful yet easy to use framework that allows objects to interact with one another across application domains in a seamless fashion. .NET Remoting provides the necessary infrastructure that makes calling methods on remote objects and returning results very transparent. It is a major improvement upon Microsoft's previous distributed applications platform MTS/COM+. The programming model was greatly simplified with the addition of C#.NET and the architecture itself now supports a variety of protocols and formats and is easily extensible to include new ones.

This transparency is achieved via *proxy objects* — when a client activates a remote object, it receives a proxy to the remote object which ensures that all calls made on the proxy object are forwarded to the appropriate remote object instance. The proxy object acts as a local delegate of the remote object. To facilitate communication between proxies and remote objects, .NET Remoting framework uses *channels* as a means of transporting messages to and from remote objects. When a method on a remote object is invoked, all of the necessary information regarding the call is transported to the remote object via channel. .NET Remoting comes with two built–in transport channels: an HTTP channel and a TCP channel. Even though these channels cover most of the possible application needs, the framework allows customization of existing or building of new channels to achieve higher performance and flexibility.

Activation of remote objects is achieved in .NET Remoting via *server activation* or *client activation*. Server–activated objects (SAO) are instantiated by the server and normally do not maintain their state between function calls. In order for clients to be able to access these objects, they must be registered with the remoting framework. Client–activated objects (CAO), or so–called stateful objects, are instantiated by the client and store information between method calls.

In summary, Microsoft .NET Remoting provides the necessary tools to build distributed application with great ease and flexibility. By hiding the details of communication between objects residing on different processes / physical machines, the framework allows the developer to concentrate on algorithms rather than on object interaction intricacies.

7.2 .NET Remoting Implementation — VSDMC Project

The main purpose of the VSDMC project was to develop an application which allows organizations/labs to leverage the computational resources currently available to them in the form of clustered systems. For example many financial institutions have access to numerous processors running under homogeneous environments (such as idling desktop computers). The VSDMC project allows for seamless integration of these resources into a single high performance computing (HPC) cluster. The VSDMC project had several important goals — flexibility, ease of configuration and deployment, and most important of all, efficiency. In order to objectively judge the performance of a distributed framework, it was important to design a lightweight application that is comparable in functionality to traditional parallel processing packages. Although configuration software packages such as Microsoft Application Center 2000 can

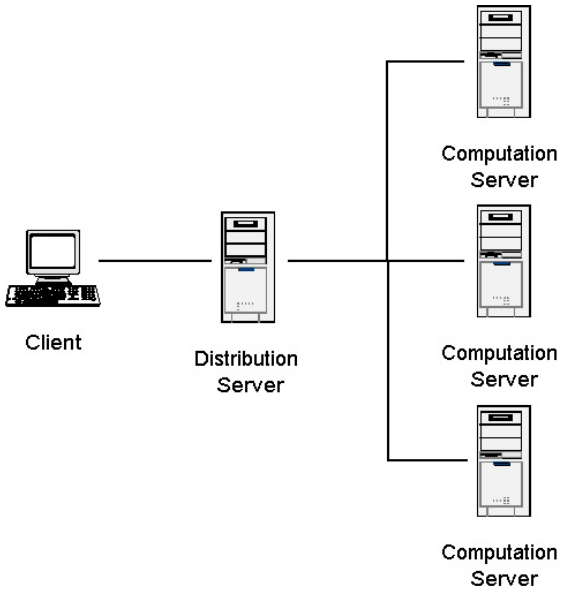

Figure 7.1: Client and Servers layout in a VSDMC cluster

achieve some of the goals stated above, they lack flexibility and performance.

The entire VSDMC application can be broken down into three distinct components: *Computation Server*, *Distribution Server*, and *Client*. This modularity permits quite flexible deployment. In fact the components are so lightweight that the Computation Server, Distribution Server, and Client can all reside on the same physical machine. Each component utilizes XML configuration files and can be easily redeployed in a different cluster. The part of the VSDMC code that facilitates communication between various components across a network was written in C# .NET, leveraging .NET Remoting technologies. The part that performs the actual computation was written in MATLAB.

7.2.1 Computation Server

The Computation Server is the application responsible for performing the computational task. Given the fact that the remoting part of application is written in C# it would be natural to develop the computational part in C# too. On the other hand, invoking a computational engine such as MathWorks MATLAB allows users to leverage readily available financial and scientific modules. Thus it was important for the Computation Server to be able to perform computational tasks written in MATLAB.

The MATLAB computational engine supports COM Automation, which allows other COM components to control MATLAB execution environment via an 'exposed' interface. Thus the Computation Server COM component can invoke MATLAB methods, store results in workspace variables, and retrieve the results by calling the interface methods. Even if there are several Computation Servers running on the same machine, each server launches its own instance of the MATLAB engine to avoid data corruption.

The Computation Server provides a single service — execute a simulation by calling a MATLAB function. In order for remote objects to be able to use this service, it has to be published in an interface file listed below.

```
public interface IServerComputation {
    // Execute the simulation via the MATLAB Automation engine
    SimulationResults RunSimulation(SimulationParameters p_objSimulationParameters);
}
```

This service provided by the Computation Server can be broken down into the following subtasks:

- Launch MATLAB Automation engine on the first call to the server. The engine is shut down when server is terminated — this saves the instantiation overhead on each call to the Computation Server.

  ```
  m_appMatlabApplication.Init();
  ```

- Invoke a method in the MATLAB Automation engine by calling the *Execute* function of the engine interface. The method name and arguments to be invoked are passed to the server as function parameters. The simulation name and parameters are stored `p_objSimulationParameters` inside

  ```
  m_appMatlabApplication.Execute("VSDMCSimulation = " +
                          p_objSimulationParameters.GetSimulationExpression());
  ```

- Retrieve the results from the MATLAB engine via the *GetFullMatrix* function.

  ```
  dblSimulationResults = m_appMatlabApplication.GetFullMatrix("VSDMCSimulation");
  ```

- Return the results to the calling object.

7.2.2 Distribution Server

The Distribution Server is the component responsible for managing the Computation Servers in a threaded environment. For each Computation Server in the cluster, the Distribution Server creates a separate Computation Server Thread (CSThread) object and the CSThread objects communicate with the underlying Computation Server in a threaded fashion. When a new task has to be computed, each CSThread accesses a pool of subtasks that comprise that task and acquires a single subtask to be computed on the Computation Server. Then CSThread invokes the MATLAB engine indirectly by triggering the *RunSimulation* method on Computation Server. When the results are returned, the CSThread will update the results in the Distribution Server and acquire a new subtask. This functionality is encapsulated into a single method which also has to be made available to remote objects via an interface file:

```
public interface IServerDistribution {
    // Start simulation on Computation Servers
    SimulationCollection RunSimulationCollection(SimulationCollection p_objSimulationCollection);
}
```

An alternative approach would be to use asynchronous function calls. The Distribution Server would asynchronously call *RunSimulation* methods on the Computation Server and wait for results. The Microsoft .NET framework provides a way of calling functions in an asynchronous fashion called *asynchronous delegates*. A delegate, a pointer to the function being called, allows invoking the function and retrieving results using *BeginInvoke* and *EndInvoke* methods. Although this approach seems more intuitive at first, using threads offers greater flexibility with features like thread synchronization, interaction, and management of common resources.

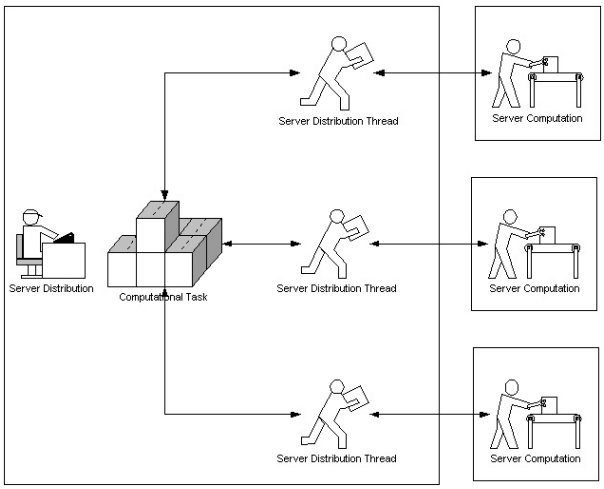

Figure 7.2: The Distribution and Computation Servers layout in a VSDMC cluster

7.2.3 Client

The Client is the component that can connect to the Distribution Server and start the simulation by calling the aforementioned method of the interface.

- Acquire a pointer to a remote Distribution Server. The *Activator.GetObject* method creates a proxy (reference) to a remote object specified by its type and URL:

```
// Get Distribution Server address from configuration file
strFullServDistAddress = "http://" + ConfigurationSettings.AppSettings["ServerDistributionAddress"] +
                         "/ServerDistribution.soap";
// Get reference to Distribution Server
m_objServerDistribution = (IServerDistribution) Activator.GetObject(typeof(IServerDistribution),
                         strFullServDistAddress);
```

- Launch the simulation on the Distribution Server

```
// Start simulation collection on Distribution Server
m_objServerDistribution.RunSimulationCollection(objSimCollection);
```

Simulation definitions and parameters are stored in an XML file, which is parsed and passed to the Distribution Server.

7.3 Baseline Test

By performing large scale computations on a cluster of workstations we can dramatically improve the performance of the algorithm by distributing the computational task among the workstations. Even though MC methods are thought to be very easily parallelized, their performance is dependent on two factors: the quality of the parallel random number generator, which affects the quality of the obtained

	N=1024	N=16384	N=32768	N=65536	N=1048576
M=1	1.584	4.374	7.248	13.201	188.547
M=2	0.942	2.316	3.677	6.619	94.413
M=4	0.776	1.487	2.114	3.439	47.418
M=8	0.869	1.156	1.572	2.330	24.690
M=16	0.860	0.819	0.992	1.380	12.721

Table 7.1: Timing results $T_{N,M}$ for $N = 2^{10}, 2^{14}, 2^{15}, 2^{16}, 2^{20}$, and $M = 1, 2, 4, 8, 16$

results, and the environment/algorithm that facilitates the interprocess communication, which affects the speed with which the results were transmitted over the network and the incurred overhead.

To obtain the most benefit from the distributed MC simulation and attain unbiased estimates, one needs to avoid using the same set of pseudo–random numbers on different machines. Most parallel random number generators are based on either leapfrog or sequence–splitting approaches, where by knowing beforehand the number of machines involved in a computation, one can easily distribute the underlying sequence of pseudo–random numbers among these machines. So a stream of random numbers used in a simulation by a single process can be distributed among M clustered processes without incurring the penalty of increased network traffic or CPU resources ((Pauletto 2000)). If $\{X_i\}_{i=1}^{N}$ is the underlying sequence and there are M clustered workstations, then the m^{th} workstation will use the subsequence $\{X_{m+Mj}\}_{j=0}^{N/M-1}$ for its computation and thus avoid intersections with other processes. While parallel pseudo– and quasi– random number generation is a topic of great interest in itself, the purpose of this section is only to evaluate the performance .NET Remoting framework as a cluster computing environment.

To conduct the test, a single computational task of evaluating the value of an MBS using the MC method with $N = 1024, 16384, 32768, 65536, 1048576$ pseudo–random numbers, was distributed among $M = \{1, 2, 4, 8, 16\}$ workstations. The total time taken to complete the execution on all workstations and return the result back to the caller is recorded. The entire task is broken down into 32 blocks to simulate a dynamic distribution algorithm that aims to reduce fault tolerance of the system via checkpointing. If for any reason a node fails (due to hardware malfunction or packet loss), only the blocks that were assigned to that workstation and not yet computed have to be distributed among other workstations.

The HPC cluster that was used to run the simulation consists of workstations and a server connected to a 3Com 100MBit switch. Each workstation is a Windows XP Professional SP1 with 1.20GHz AMD Duron processor and 1.00GB RAM and the server is a Windows 2003 Server with a Dual 2.4GHz processor and 1.00GB RAM. Each workstation ran a single Computation Server while a separate worstation ran a Distribution Server.

7.4 Results

Table 7.1 contains the test results, which show a definite decrease in total computational time for all simulations. As expected, the longer computations benefited the most from the added computational power while the shorter computations exhibited diminishing returns for the investment of additional

	N=1024	N=16384	N=32768	N=65536	N=1048576
$T_{N,*}$	0.181	2.988	5.887	11.666	186.908

Table 7.2: Serial timing results $T_{N,*}$ for $N = 2^{10}, 2^{14}, 2^{15}, 2^{16}, 2^{20}$

	N=1024	N=16384	N=32768	N=65536	N=1048576
M=1	1.000	1.000	1.000	1.000	1.000
M=2	0.841	0.944	0.986	0.997	0.999
M=4	0.510	0.735	0.857	0.960	0.994
M=8	0.228	0.473	0.576	0.708	0.955
M=16	0.115	0.334	0.457	0.598	0.926

Table 7.3: Efficiency results $E_{N,M}$ for $N = 2^{10}, 2^{14}, 2^{15}, 2^{16}, 2^{20}$, and $M = 1, 2, 4, 8, 16$

workstation. It is important to note that the number of checkpoints was kept constant at 32. For short simulation we can see how the frequent communication with the master process will affect the overall speed of the computation.

To evaluate a distributed system we'll use a measure of *efficiency* defined as $E_{N,M} = T_{N,1}/(T_{N,M} \cdot M)$,where $T_{N,M}$ is the execution time of N iterations on M processors. As previously mentioned, $T_{N,1}$ is not the serial execution time since it includes 32 checkpointing calls to the Distribution Server. The serial execution times were recorded separately and are given in Table 7.2. Notice that $T_{N,1} \neq T_{N,*}$. $T_{N,1} - T_{N,*}$ is the time it takes to make 32 checkpointing calls to the Distribution Server. The values of $E_{N,M}$ lie in $[0, 1]$ and measure the proportion of the total time actually spent on computation, while $1 - E_{N,M}$ is spent on network communication and distribution algorithm. In a best–case scenario, where doubling the number of processors corresponds to halving computational time, $E_{N,M}$ is close to 1, while in real life $E_{N,M} \to 0$ as $M \to \infty$ since the amount of time spent on communication stays constant yet takes a larger proportion of the total computation time. While it is impossible to achieve constant efficiency for a fixed–size problem, we can try to achieve *scalability*, constant efficiency as problem size and number of processors both increase.

Bibliography

Akesson, F. and J. Lehoczy (2000). Path generation for quasi-monte carlo simulation of mortgage-backed securities. *Management Science 46*, 1171–1187.

Caflisch, R. and B. Moskowitz (1995). Modified monte carlo methods using quasi-random sequences. *Lecture Notes in Statistics 106*(1), 1–16.

Caflisch, R. E., W. J. Morokoff, and A. B. Owen (1997). Valuation of mortgage backed securities using brownian bridges to reduce effective dimension. *J. Comp. Finance 1*(1), 27–46.

Kelman, A. (2002). Mortgage-backed securities & collateralized mortgage obligations: Prudent cra investment opportunities. *Community Investments 14*(1), 20–23.

Lifka, D., L. Walle, V. Zaloj, and J. Zollweg (2002). Increasing the accessibility of parallel processing with microsoft .net. *PowerSolutions*.

Liu, R. and A. B. Owen (2003). Estimating mean dimensionality. *Dept. of Statistics, Stanford Univ, Technical Reports 14*.

Morokoff, W. J. (1998). Generating quasi-random paths for stochastic processes. *SIAM Review 40*(4), 765–788.

Morokoff, W. J. and R. E. Caflisch (1994). Quasi-random sequences and their discrepancies. *SIAM Journal on Scientific Computing 15*(6), 1251–1279.

Owen, A. B. (1998). Monte carlo extension of quasi-monte carlo. *Proceedings of the 1998 Winter Simulation Conference*.

Owen, A. B. (2003). The dimension distribution and quadrature test functions. *Statistica Sinica 13*(1), 1–18. In press.

Papageorgiou, A. (2002). The brownian bridge does not offer a consistent advantage in quasi-monte carlo integration. *Journal of Complexity 18*(1), 171–186.

Paskov, S. H. (1997). New methodologies for valuing derivatives. In S. Pliska and M. Dempster (Eds.), *Mathematics of Derivative Securities*, pp. 545–582. Cambridge: Cambridge University Press.

Paskov, S. H. and J. F. Traub (1995, Fall). Faster valuation of financial derivatives. *J. Portfolio Management 22*(1), 113–120.

Pauletto, G. (2000). Parallel monte carlo methods for derivative security pricing. In *Computing in Economics, Finance 2000*, pp. 650–657.

Solms, F. and W.-H. Steeb (1998). Distributed monte carlo integration using corba and java. *International Journal of Modern Physics C 9*(7), 903–915.

Wanga, X. and K.-T. Fang (2003). The effective dimension and quasi-monte carlo integration. *Journal of Complexity 19*, 101–124.

Appendix A

Graphs of the Numerical Results

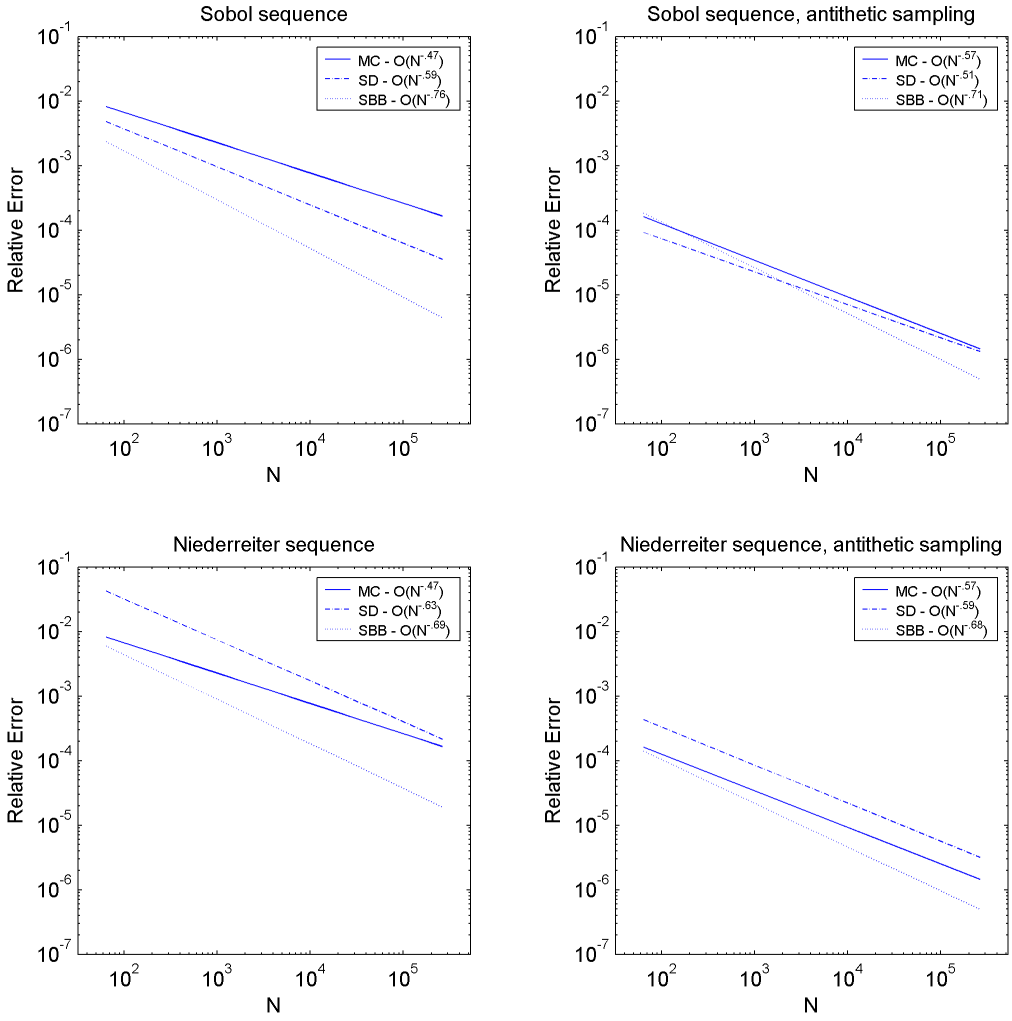

Figure A.1: MC vs. QMC for the Nearly Linear problem

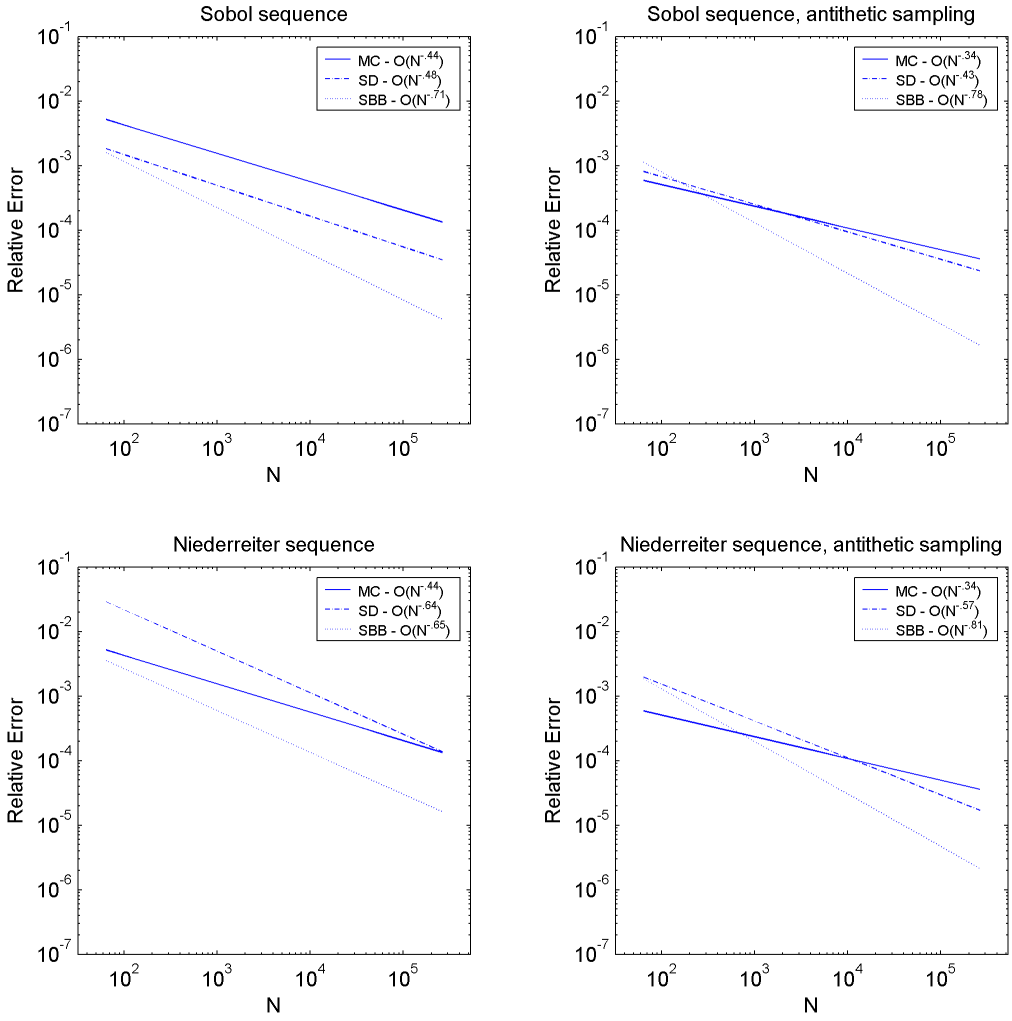

Figure A.2: MC vs. QMC for the Nonlinear problem

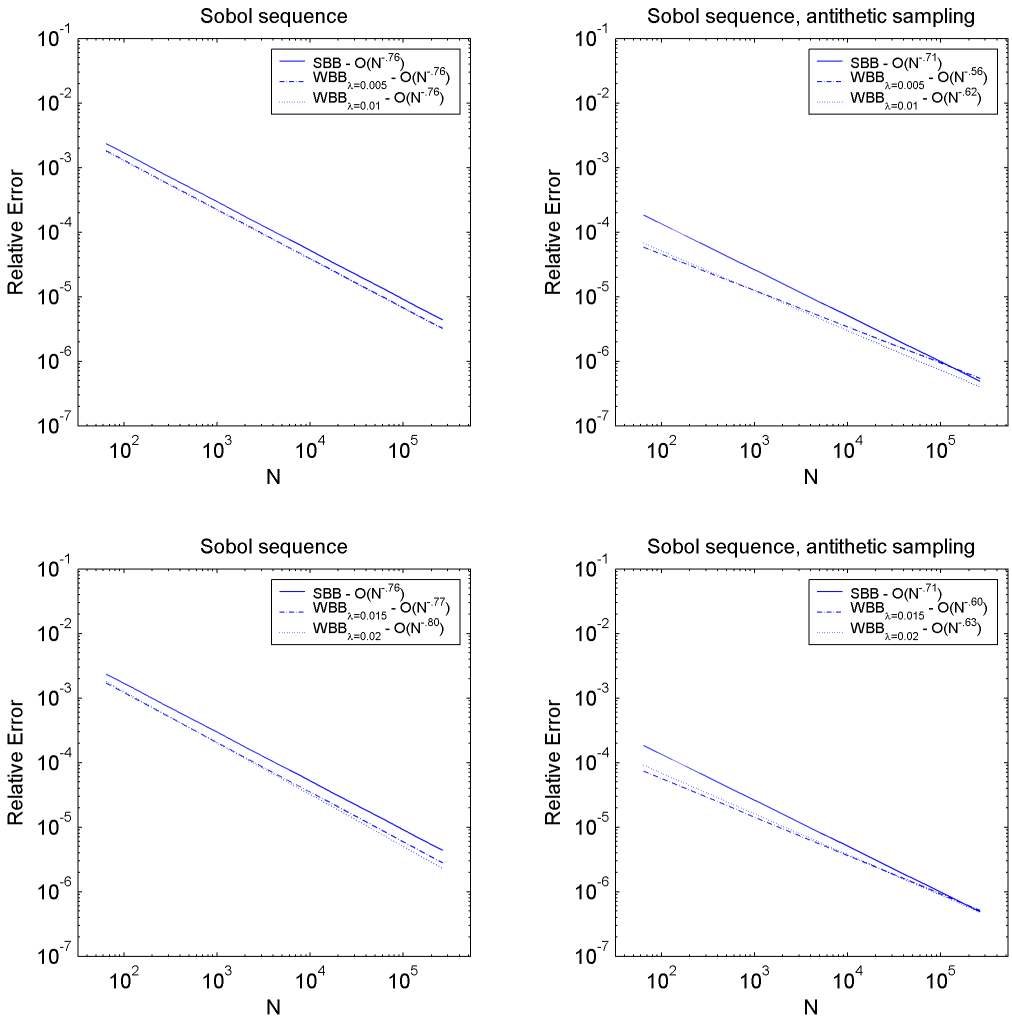

Figure A.3: SBB vs. WBB_λ using Sobol sequence for the Nearly Linear problem

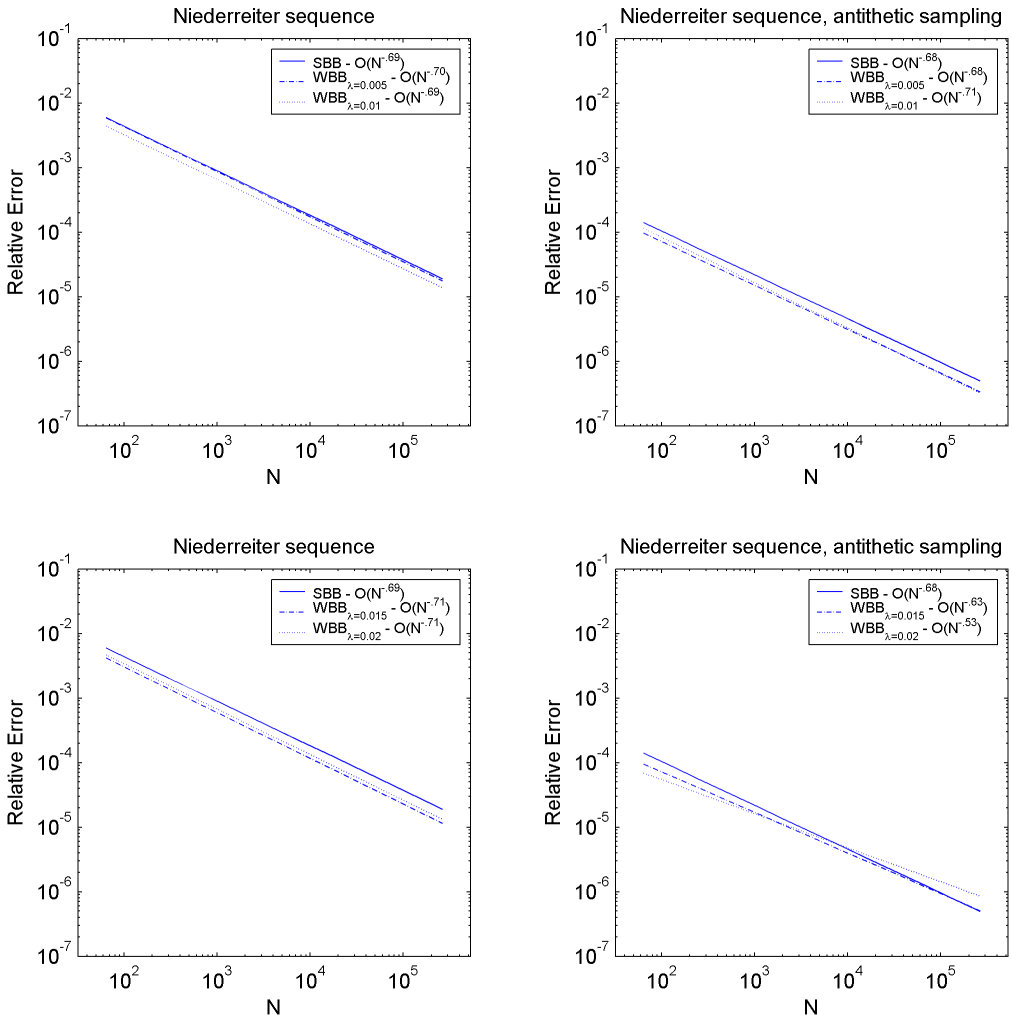

Figure A.4: SBB vs. WBB_λ using Niederreiter sequence for the Nearly Linear problem

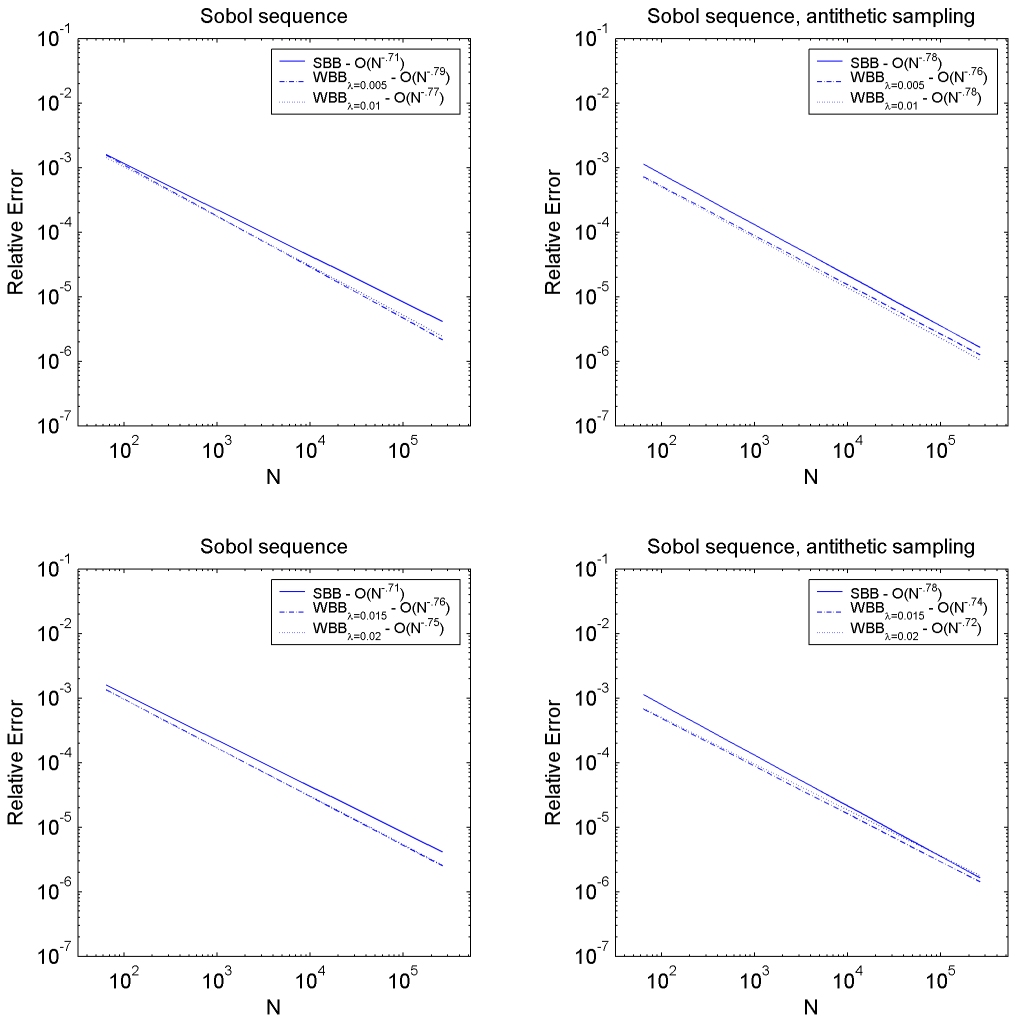

Figure A.5: SBB vs. WBB_λ using Sobol sequence for the Nonlinear problem

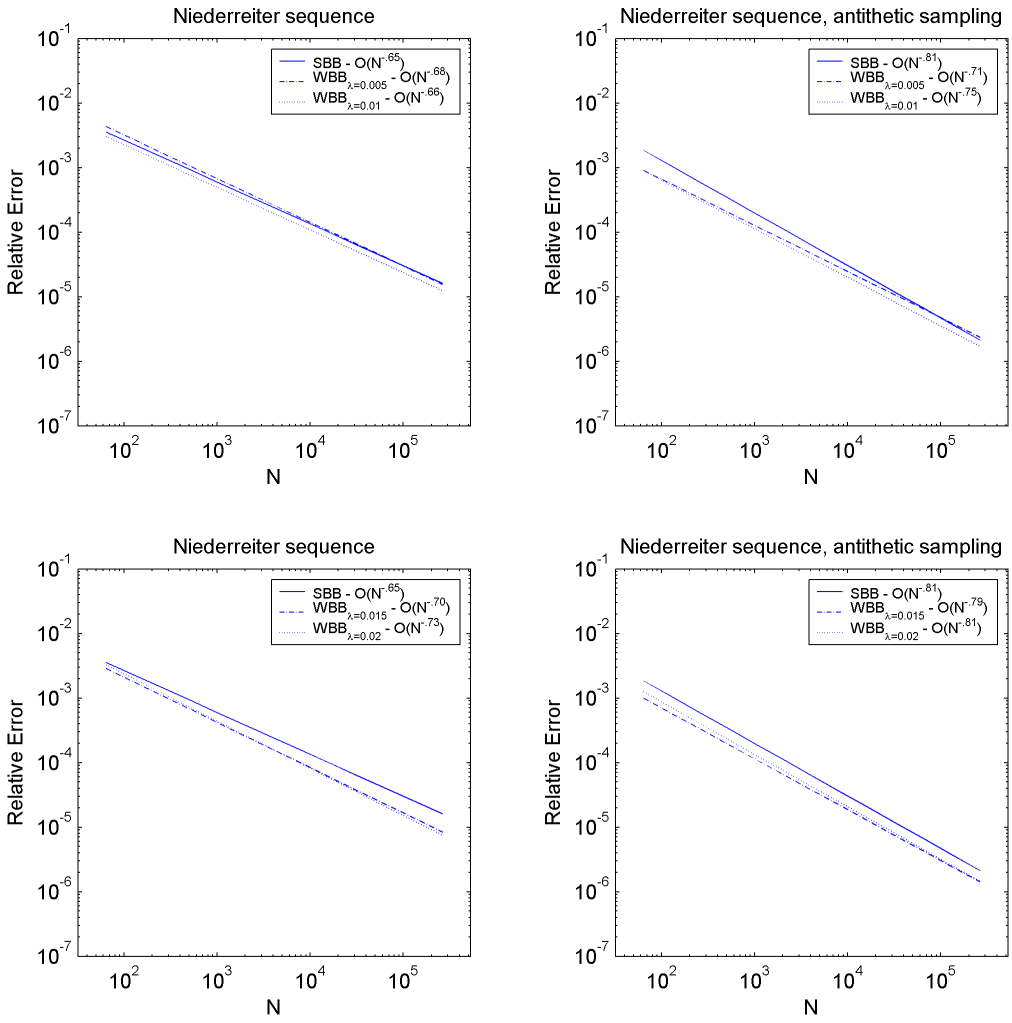

Figure A.6: SBB vs. WBB_λ using Niederreiter sequence for the Nonlinear problem